A Domain-Independent Agent Framework for Modelling, Simulating and Solving Tasks in Internet of Things (IoT) Systems

Zur Erlangung des akademischen Grades eines
Doktors der Ingenieurwissenschaften

(Dr.-Ing.)

von der KIT-Fakultät für Wirtschaftswissenschaften
des Karlsruher Instituts für Technologie (KIT)

genehmigte

DISSERTATION

von

Dipl. Informatik (FH) Nicole N. Merkle

Tag der mündlichen Prüfung:	30.01.2024
Referent:	Prof. Dr. Andreas Oberweis
Korreferent:	Prof. Dr. Anne Koziolek

Karlsruhe 30.01.2024

Danksagungen

Ich möchte meine Dissertation zum Anlass nehmen, um eine wahre Anekdote zu erzählen und mich bei den Menschen, die mich auf meinem Weg zum Doktor Grad begleitet und unterstützt haben zu bedanken.

Die Anekdote lautet wie folgt: Zu meinem 40. Geburtstag bekam ich von meinen ehemaligen Kolleg*innen und dem ehemaligen "Doktorvater" aus grauer Vorzeit eine Karte mit einer Schnecke darauf abgebildet und über der Karte stand der Text: "Besser spät als nie". Die Karte sollte wohl auf vermeintlich "witzige" Art einen gewissen Spott darüber ausdrücken, dass ich in fortgeschrittenem Alter promovieren wollte, und wie es den Anschein erweckte, wurde mein Tempo im Promovieren mit dem einer Weinbergschnecke verglichen. Ich vermute, dass auch Wetten darauf abgeschlossen wurden, ob und wann ich die Promotion schaffen würde, da sich diese auf Grund unliebsamer Umstände und Bestrebungen gewisser Personen so ziemlich in die Länge zog. Nur war mir das damals zu dem Zeitpunkt noch nicht bewusst.

Was die ehemaligen Kolleg*innen und der damalige "Doktorvater" in ihrem Spott und ihrer Ignoranz, jedoch nicht bedacht haben, ist, dass "Schnecken" interessante Tiere sind, die zwar klein, unscheinbar und langsam, aber dafür zäh und geduldig sind. Eine Schnecke hinterlässt ihre Spuren und überwindet Hindernisse, die ihr immer wieder Mal auf ihrem Weg begegnen. Sie geht ihren Weg in ihrem eigenen Tempo und lässt sich dabei nicht davon abbringen, und dabei streckt sie immer wieder ihre Fühler aus, um sich neu zu orientieren und die nächste Etappe zu verfolgen.

Diese Geburtstagskarte wurde für mich zum Sinnbild und Ansporn es der Weinbergschnecke gleich zu tun. Auch wenn ich in meinem Leben und meiner Promotion langsame Schritte gegangen bin und dabei Umwege gemacht habe, so habe ich trotz aufkommender Hindernisse nie meine Zuversicht und mein Ziel aus den Augen verloren und bin entgegen aller Steine, die mir durch so manche Person in den Weg gelegt wurden, nicht davon abgehalten worden, meinen Weg zum erstrebten Ziel zu gehen. Auf diesem Weg habe ich sehr viel gelernt, vor allem über mich selbst und es haben mich einige Menschen begleitet und denjenigen, die mich menschlich bereichert, unterstützt bzw. gefördert haben, genau diesen Menschen möchte ich aus ganzem Herzen danken.

Allen voran möchte ich meiner Doktormutter Prof. Dr. Anne Koziolek, die mich von dem damaligen Lehrstuhl, aus dem ich auf eigenen Wunsch ausgetreten bin, "adoptiert" hat, ganz herzlich danken, dafür, dass sie den Anstand, die menschliche Größe und Offenheit besessen hat, mich ohne Vorbehalte bzw. unvoreingenommen als Doktorandin zu übernehmen. Sie war letztendlich die einzige Person unter den angefragten

Professor*innen, die, trotz bzw. aufgrund der problematischen Hintergründe meines Lehrstuhlwechsels, gesehen hat, in welcher schwierigen und ungerechtfertigten Situation ich mich befand und auch erkannt, dass ich auf der Zielgeraden zur Promotion Unterstützung benötige und das Potential besitze, diese zu erreichen. Es war für mich immer sehr angenehm mit ihr über meine Doktorarbeit und die inhaltlichen Themen zu sprechen und ich habe einiges über wissenschaftliches Arbeiten von ihr gelernt und erfahren, dass eine Betreuung, im Gegensatz zu früheren Erfahrungen, auch angenehm und unkompliziert verlaufen kann. Zudem bewundere ich, dass sie Familie und Beruf sowie zusätzliche Doktorand*innen wie mich unter einen Hut bekommt. Ein großes Dankeschön, dass ich mit der Unterstützung von Frau Prof. Dr. Koziolek meine Dissertation erfolgreich beenden durfte.

Meine Entscheidung den "Doktorvater" zu wechseln, habe ich jedenfalls zu keiner Zeit bereut. Im Gegenteil, ich habe mittlerweile auch ein unterstützendes Arbeitsumfeld mit sehr netten Kolleg*innen und Doktorand*innen gefunden, in dem Menschen nicht auf Grund persönlicher Merkmale diskriminiert und ausgegrenzt werden.

Weiter möchte ich Prof. Dr. Oberweis danken, der sich bereit erklärte, als Erst-Gutachter meine Doktorarbeit zu bewerten. Ich habe ihm und Frau Koziolek ziemlich viel zum Lesen zugemutet, da ich mich einfach nicht kurz fassen kann. Ich hoffe, dass meine Arbeit trotz Umfang, interessante Einblicke in mein Thema geben konnte und beide Gutachter*innen einen gewissen Unterhaltungswert darin finden konnten, auch wenn das Thema meiner Dissertation nicht ihr wissenschaftliches Interessengebiet umfasste.

Ich möchte auch meinen ehemaligen Kolleg*innen Dr. Tom Zentek, Nadia Ahmed, Prof. Dr. Stefan Zander sowie Dr. Viliam Simko danken.

Tom hatte meine Diplomarbeit betreut und war der erste Kollege mit dem ich nach meinem Diplom Abschluss am FZI zusammen arbeiten und Projekt Erfahrung sammeln durfte. Dabei habe ich sehr viel darüber gelernt, wie es in der Arbeitswelt zugeht. Zudem habe ich mich in der damaligen Abteilung sehr wohl gefühlt und das Arbeitsklima war sehr angenehm und familiär. Mir ist auch mit der Zeit klar geworden, wie wichtig und leider nicht selbstverständlich, ein gutes Arbeitsklima, offene Kommunikation, Teamfähigkeit, verträgliche Kolleg*innen und Chefs sind.

Meine Kollegin Nadia, mit der ich später ein Büro teilen durfte, hatte mich damals überredet an einer "Semantic Web Summer School" teilzunehmen und hatte mich ermuntert die Promotion, zu versuchen, da ich damals als FH Abgängerin mit Diplom Abschluss überhaupt keine Ambitionen hatte zu promovieren. Auf der Summer School wurde ich dann auch von Prof. Dr. Maria-Esther Vidal ermuntert die Chance einer Promotion zu nutzen, da, laut ihren Aussagen, meine Themen, die ich dort präsentiert hatte, interessant seien und Potential für eine Promotion hätten. Sie hat mir auch bei meinem aller ersten Paper wertvolles Feedback gegeben, so dass es dann auch tatsächlich auf einer Konferenz akzeptiert wurde und das für mich den Ausschlag gegeben hat es wirklich anzugehen.

Mir ist klar geworden, dass es manchmal einzelne Menschen sind, die zum richtigen Zeitpunkt durch Zuspruch und gezeigtes Zutrauen einem den entscheidenden Impuls geben, einen neuen Weg einzuschlagen. Deshalb ein großes Danke an Nadia und Prof. Dr. Vidal, denen ich ihren damaligen Impuls und Zuspruch hoch anrechne.

Nun komme ich zu Stefan und Viliam. Beide haben mich damals am FZI als Postdocs betreut. Mit Stefan habe ich die meisten Paper geschrieben bzw. auch veröffentlicht und er hat mir einiges über das Schreiben von wissenschaftlichen Papern und den akademischen "Betrieb" beigebracht. Zudem war er ein sehr freundlicher und angenehmer Kollege, mit dem ich mich gut verstanden habe und der den allzu treffenden Begriff "Wissenschafts-Zirkus" in mein Vokabular brachte. Im Verlauf meiner Promotion habe ich öfter daran denken müssen, dass es zum Glück auch andere Menschen außer mir gibt, die es so empfinden, wie der Begriff es suggeriert.

Nachdem Stefan Professor wurde und das FZI verließ, übernahm Viliam die Betreuung meiner Arbeit. Zu Viliam ist zu sagen, dass er ein angenehmer Kollege und Zeitgenosse ist, mit dem ich sehr gerne zusammengearbeitet und von dem ich auch einiges technisches Know-how zu Software Technik, Frameworks und APIs sowie R gelernt habe. Vor allem haben mir unsere langen und amüsanten Diskussionen zu JavaScript, funktionaler Programmierung, Fotografie, den Bau von Flipper-Automaten, sowie den Nutzen- bzw. Nicht-Nutzen von "Semantic Web" als auch das Ansehen von skurrilen und witzigen YouTube Videos über Arbeitsunfälle Spass gemacht. Ich finde man sollte mehr solcher Kollegen wie Viliam haben, mit denen man auf Augenhöhe reden und arbeiten kann. Er hat, im Gegensatz zu manch anderen, nie den überheblichen oder besserwisserischen Postdoc raus hängen lassen, obwohl er sehr viel Ahnung und technisches Know-how besitzt, sondern war ein Kollege, den ich sehr schätze.

Zu guter Letzt möchte ich meinen Eltern Birsen Merkle und Manfred Merkle danken, ohne deren Zutun ich nicht auf diesem Planeten wäre. Sie haben mich auf all meinen (Um-)Wegen und Sackgassen begleitet und mir Kraft, Mut, Zuversicht, Hoffnung und Selbstvertrauen gegeben. Ich bin dankbar solche Eltern zu haben, die mich so akzeptieren und lieben wie ich bin.

Acknowledgements

I would like to take my doctoral thesis as opportunity to tell a true anecdote and to thank the people who have accompanied and supported me on my way to my doctoral degree.

The anecdote is like this: For my 40th birthday, I received a card with a snail on it from former colleagues and the former "Ph.D. supervisor". Above the card was the text: "Better late than never". The card was supposedly meant to be "funny" and should obviously express a certain mockery of the fact that at this advanced age of 40 at the time, I still wanted to do a doctorate and presumably my speed in doing a doctorate was compared to that of a vineyard snail. I suspect that bets were also placed on whether and when I would be able to complete my doctorate, as this took quite a long time due to unpleasant circumstances and the endeavours of certain people. But I didn't realise that at the time.

What the former colleagues and "Ph.D. supervisor" did not consider in their mockery and ignorance, however, is that snails are interesting animals that may be small and slow, but patient and tough. A snail leaves its traces and overcomes obstacles that are placed in its path from time to time. A snail goes its own way at its own pace and does not allow itself to be diverted from its path, and in the process it always puts out feelers to reorient itself in order to pursue the next stage.

This birthday card became a symbol and incentive for me to do the same as the snail. Even though I have taken slow steps and detours in the course of my dissertation and my life, I have never lost sight of the goal and have not let myself be deterred from reaching the desired goal despite all the obstacles that some people have put in my way. On this path I have learned a lot, especially about myself and there have been people who have accompanied and supported me, especially these people I would like to thank here.

First and foremost, I would like to thank my doctoral supervisor Prof. Dr. Anne Koziolek, who "adopted" me from the former chair I left at my own initiative, for the fact that she had the decency, human size and openness to accept me as a doctoral candidate without reservations or bias. Ultimately, she was the only person among the professors approached who, despite or because of the problematic background of my change of research chair, saw the difficult and unjustified situation I was in and also recognised that I needed support on the home stretch to my doctorate and that I had the potential to achieve it. It was always very pleasant for me to talk to her about my doctoral thesis and the content-related topics and I learnt a lot about scientific work from her and experienced that, in contrast to previous experiences, supervision can also be pleasant and uncomplicated. I also admire the fact that she manages to juggle family and career as well as additional doctoral students like me. A big thank you for allowing me to successfully complete my doctorate with the

support of Prof. Dr. Koziolek.

I have never regretted my decision to change my doctoral supervisor. On the contrary, I have now also found a supportive working environment with very nice colleagues and doctoral students in which people are not discriminated against or marginalised on the basis of personal characteristics.

Furthermore, I would like to thank Prof. Dr. Oberweis, who agreed to be the first reviewer of my thesis. I expected him and Mrs. Koziolek to read quite a lot, as I simply cannot be brief. I hope that, despite its length, my thesis was able to provide interesting insights into my topic and that both reviewers were able to find some entertainment value in it, even if the topic of my dissertation did not cover their scientific field of interest.

I would especially like to thank my former colleagues Dr. Tom Zentek, Nadia Ahmed, Prof. Dr. Stefan Zander and Dr. Viliam Simko.

Tom had supervised my diploma thesis and was the first colleague with whom I was able to work and gain project experience at the FZI after graduating. I learnt a lot about how things work in the world of labour. I also felt very comfortable in the department at the time and the working atmosphere was very pleasant and informal. Over time, I also realised how important a good working atmosphere, open communication, team spirit and friendly colleagues and bosses are, which unfortunately cannot be taken for granted.

My colleague Nadia, with whom I later shared an office, had persuaded me to take part in a "Semantic Web Summer School" and encouraged me to try for a doctorate, as I had no ambitions at all to do a doctorate as a graduate with a degree from a university of applied sciences at that time. At the summer school, I was also encouraged by Prof. Dr. Maria-Esther Vidal to take the opportunity to do a doctorate because, according to her, the topics I presented there were interesting and had potential for a doctorate. She also gave me valuable feedback on my very first paper, so that it was actually accepted at a conference and that was the deciding factor for me to really go for it.

I realised that sometimes it is individual people who give you the decisive impetus to take a new path at the right time through encouragement and confidence. So a big thank you to Nadia and Prof. Dr. Vidal, who I greatly appreciate for their impetus and encouragement at the time.

Now I come to Stefan and Viliam. Both of them supervised me as post-doctoral researchers at the FZI. I wrote and published most of my papers with Stefan and he taught me a lot about writing scientific papers and the academic "business". He was also a very friendly and pleasant colleague with whom I got on well and who brought the all too apt term "scientific circus" into my vocabulary. During the course of my doctorate, I often had to think about the fact that, fortunately, there are other people besides me who feel the way the term suggests.

After Stefan became a professor and left the FZI, Viliam took over the supervision of my work. I have to say that Viliam is a pleasant colleague and contemporary with whom I really enjoyed working and from whom I learnt a lot of technical know-how about software technology, frameworks and APIs as well as R. Above all, I enjoyed our long and amusing discussions about JavaScript, functional programming, photography, building pinball machines and the benefits or non-benefits of the "Semantic Web", as well as watching bizarre and funny YouTube videos about accidents at work. I think we should have more colleagues like Viliam with whom we can talk and work at eye level. In contrast to some others, he never let the arrogant or know-it-all post-doc hang out, even though he has a lot of knowledge and technical expertise, but was a colleague I really appreciated.

Last but not least, I would like to thank my parents Birsen Merkle and Manfred Merkle, without whose help I would not be on this planet. They have accompanied me on all my journeys, (re)paths and dead ends, and have given me strength, courage, optimism, hope and self-confidence. I am grateful that I have such parents who accept and love me for who I am.

List of Referenced Publications

This Thesis is based on the following publications:

[157] Nicole Merkle and Stefan Zander. “Improving the Utilization of AAL Devices through Semantic Web Technologies and Web of Things Concepts”. In: Procedia Computer Science 98.Supplement C (2016). The 7th International Conference on Emerging Ubiquitous Systems and Pervasive Networks (EUSPN 2016)/The 6th International Conference on Current and Future Trends of Information and Communication Technologies in Healthcare (ICTH-2016)/AffiliatedWorkshops, pp. 290–297. issn: 1877-0509.

[156] Nicole Merkle and Stefan Zander. “Agent-Based Assistance in Ambient Assisted Living Through Reinforcement Learning and Semantic Technologies”. In: On the Move to Meaningful Internet Systems. OTM 2017 Conferences. Ed. by Hervé Panetto et al. Cham: Springer International Publishing, 2017, pp. 180–188. isbn: 978-3-319-69459-7.

[159] Nicole Merkle, Stefan Zander, and Viliam Simko. “A Semantic Use Case Simulation Framework for Training Machine Learning Algorithms”. In: Knowledge Engineering and Knowledge Management. Ed. by Catherine Faron Zucker et al. Cham: Springer International Publishing, 2018, pp. 243–257.

[158] Nicole Merkle and Stefan Zander. “Using a Semantic Simulation Framework for Teaching Machine Learning Agents”. In: Procedia Computer Science 137 (2018). Proceedings of the 14th International Conference on Semantic Systems 10th, 13th of September 2018 Vienna, Austria, pp. 78–89. issn: 1877-0509.

[155] Nicole Merkle and Patrick Philipp. “Cooperative Web Agents by Combining Semantic Technologies with Reinforcement Learning”. In: Proceedings of the 10th International Conference on Knowledge Capture, K-CAP 2019, Marina Del Rey, CA, USA, November 19-21, 2019. Ed. by Mayank Kejriwal, Pedro A. Szekely, and Raphaël Troncy. ACM, 2019, pp. 205–212.

[152] Nicole Merkle and Ralf Mikut. "Context-Aware Composition of Agent Policies by Markov Decision Process Entity Embeddings and Agent Ensembles". Semantic Web Journal (SWJ), IOS Press. 2024. pp. 1-29.

Abstract

Considering the digitisation and trends in heterogeneous domains (e.g. healthcare, robotics, smart home, automated web agents), a growing number of tasks and processes are outsourced to intelligent software agents that are able to perform and solve process oriented tasks and make recommendations based on sensed observations and contextual information. For instance, the Internet of Things (IoT) and artificial intelligence (AI) algorithms help intelligent software agents to perform their tasks in *cyber-physical and IoT systems* [174, 28, 10]. Meanwhile, smart watches, medical bands, mobile applications as well as smart home platforms move mainstream and are available for affordable costs.

The mentioned digitisation in every day life leads to the generation of valuable data and further to social implications such as outsourcing tasks to intelligent software agents. By handling these valuable data through software agents, informative insights can be gained that allow making smart decisions, while costs and (human) resources can be saved.

Thereby, the objective of agents is to imitate–based on domain knowledge, learned strategies (policies) and their made observations–as good as possible the decisions of human domain experts. Agents also have to exhibit adaptive and well-defined behaviour to neither cause harm in their environment nor harm themselves. However, this brings many challenges for software agents they have to deal with, especially in *IoT* systems.

Due to the aforementioned requirements, agent frameworks or platforms are necessary that cover the life cycle and process steps of software agents from development to deployment, training, execution and maintenance. However, such frameworks are usually developed for game environments or specific domains and tasks, or only fulfil the named requirements of software agents to a limited extent.

The objective of the presented agent framework is to enable agents making smart and well-defined decisions by modelling and combining semantic technologies and machine learning in order to solve tasks in IoT systems. Thereby, the simulation of guidelines and mathematical models, representing phenomena and tasks, shall guide agents in their learning process.

For this reason, this work aims for providing a domain-independent, conceptual reference framework that allows a simplified formalisation of decision process oriented tasks in order to simulate/imitate them in a generic way so that software agents are enabled to train task and process characteristics and apply the resulting policies for solving IoT tasks adequately in a domain-independent and user-centric way. Furthermore, the framework aims for allowing agents the adaptation and formal description of trained policies so that these can be shared and are reusable by rule-based agents for novel tasks and individual

users.

This work is intended to contribute to the support of people by intelligent software agents within heterogeneous IoT environments and domains (e.g. healthcare, service robotics, web agents, ambient assisted living, etc.).

Contents

Vorwort . . . . . **i**

Acknowledgements . . . . . **v**

List of Figures . . . . . **xv**

List of Tables . . . . . **xix**

1. Motivation **1**
1.1. The Cold-Start Problem of Agents 2
1.2. Context-Awareness and Personalisation of Agents 6
1.3. Mining, Sharing and Reusing of Produced Expert Knowledge 10
1.4. Structure of the Thesis 12
1.5. Summary of this Chapter 13

2. Foundations **15**
2.1. Agent-based Systems 16
2.1.1. Properties of Intelligent Agents 17
2.1.2. Properties of the Task Environment 19
2.1.3. Simple Reflex Agent 20
2.1.4. Model-Based Reflex Agent 20
2.1.5. Goal-Based Agent 21
2.1.6. Utility-Based Agent 21
2.1.7. Learning Agent 22
2.2. Semantic Technologies 23
2.2.1. RDF(S), Turtle, SHACL, JSON-LD and RDFa 24
2.2.2. Description Logic 32
2.2.3. SPARQL 36
2.2.4. Semantic Web Rule Language (SWRL) 37
2.2.5. Notation3 (N3) 38
2.2.6. The Internet of Things (IoT) 39
2.2.7. The Web of Things 40
2.3. Machine Learning 42
2.3.1. Learning 42
2.3.2. Supervised Learning 43
2.3.3. Unsupervised Learning 44
2.3.4. Over- and Under-fitting 45
2.3.5. Classification versus Regression 45

2.3.6. Feature Engineering and Data Preprocessing . . . 46
2.3.7. Evaluation of ML Models . . . 47
2.3.8. Multinomial and Gaussian Naive Bayes Classifier . . . 50
2.3.9. Markov Decision Process . . . 52
2.3.10. Reinforcement Learning . . . 54
2.3.11. Hyperparameter Optimisation . . . 58
2.3.12. Word/Entity Embeddings . . . 62
2.3.13. Genetic Algorithms . . . 64
2.4. Decision Rule Mining . . . 69
2.4.1. Frequent Item sets . . . 69
2.4.2. Association Rules . . . 70
2.4.3. OneR . . . 71
2.4.4. Sequential Covering . . . 71
2.5. Summary of this chapter . . . 75

3. Related Work . . . 77
3.1. Cold-Start Problem . . . 78
3.1.1. Recommender Systems . . . 79
3.1.2. Robot Systems . . . 82
3.1.3. Human Computer Interaction Systems . . . 84
3.1.4. Summary . . . 85
3.2. Context-Aware Policies Adaptation and Combination . . . 88
3.2.1. Reinforcement Learning Approaches . . . 88
3.2.2. Language Models for Policies Combination . . . 90
3.2.3. Rule Mining and Process Mining . . . 92
3.2.4. Summary . . . 98
3.3. Retrieval and Reuse of Policies . . . 101
3.3.1. Summary . . . 104
3.4. Summary of this Chapter . . . 106

4. Approach . . . 109
4.1. Overview of Roles and Framework Phases . . . 111
4.2. Framework Ontology: Formal Representation of Domain-Independent Knowledge . . . 118
4.2.1. The Simulation of Phenomena by Mathematical Models . . . 119
4.2.2. Concepts and Properties . . . 120
4.3. Simulation of MDP Tasks . . . 133
4.3.1. Message exchange . . . 134
4.3.2. Reasoning of States . . . 136
4.3.3. Reasoning of Rewards . . . 138
4.4. Adaptation and Mining of Policies . . . 141
4.4.1. Selection of Rule Mining Algorithm . . . 142
4.4.2. Dataset Structure of Input . . . 142
4.4.3. The Relation between Data Samples and Rule Items . . . 143
4.4.4. Data Preprocessing and Rule Mining Algorithms . . . 143

4.5. Retrieval of Policies . . . 155
4.5.1. Retrieval of Reusable Task Policies . . . 155
4.5.2. Context-Dependent Combination of Task Policies . . . 158
4.6. Conceptualisation of Agents . . . 178
4.6.1. General Structure and Flow of the Agent Program . . . 178
4.6.2. Worker Agent . . . 183
4.6.3. Consumer Agent . . . 184
4.7. Assumptions and Limitations . . . 186
4.8. Summary of this Chapter . . . 189

5. Use Cases . . . 193
5.1. UC1: Automated Execution of Web Tasks . . . 194
5.1.1. Flight Booking Task . . . 195
5.1.2. Click Check-box by Label or Synonymous Label . . . 196
5.1.3. Click Button and Button Sequence . . . 196
5.1.4. Click List Option by Label . . . 197
5.2. UC2: Virtual Coaching based on Clinical Pathways . . . 201
5.2.1. Cardiac Disease Pathway . . . 202
5.3. UC4: COVID-19 Measures for Public Health and Economy . . . 206
5.3.1. The SIR Model . . . 207
5.3.2. The SEIR Model . . . 208
5.4. UC5: Domestic Activities . . . 213
5.5. Summary of this chapter . . . 217

6. Evaluation . . . 219
6.1. RQ1: Cold-Start Problem . . . 219
6.1.1. General Goals addressing RQ1 . . . 220
6.1.2. Metrics . . . 221
6.1.3. Set-up . . . 222
6.1.4. Results and Discussion . . . 225
6.2. RQ2: Context-Aware Policy Adaption and Disambiguation . . . 237
6.2.1. General Goals addressing RQ2.1 and RQ2.2 . . . 238
6.2.2. Metrics . . . 240
6.2.3. Set-up . . . 241
6.2.4. Results and Discussion . . . 247
6.3. RQ3: Share and Reuse of Policies . . . 257
6.3.1. General Goals addressing RQ3 . . . 257
6.3.2. Metrics . . . 258
6.3.3. Set-up . . . 260
6.3.4. Results and Discussion . . . 267
6.4. Threats to Validity . . . 287
6.4.1. Conclusion Validity . . . 288
6.4.2. Internal Validity . . . 290
6.4.3. Construct Validity . . . 292
6.4.4. External Validity . . . 294

6.5. Summary of this chapter 296

7. Summary and Outlook 299
7.1. Discussion about Contributions 299
7.1.1. RQ1: The cold-start problem of agents 300
7.1.2. RQ2: Context-awareness and personalisation of agents 301
7.1.3. RQ3: Mining, sharing and reusing of produced expert knowledge 302
7.1.4. The simulation of phenomena by mathematical models 303
7.1.5. Summary of Contributions 303
7.2. Future Work 305
7.2.1. Evaluation of Framework Usability 305
7.2.2. Security concerns and trust 305
7.2.3. Incorporating fuzzy states by fuzzy logic 305
7.2.4. Open-source and platform-independent framework distribution . 306
7.2.5. Environmental design and Generative Models for varying Simulations 306
7.2.6. Large language models for transforming natural language into semantic task entity descriptions and policy instructions 306
7.2.7. A Wikidata knowledge graph for agent programs 308

Bibliography 309

A. Reward Function Example 335

B. State Update 339

C. Data Preprocessing 343

D. List of all Scientific Publications 347

List of Figures

2.1. The interaction of an agent with its environment. 18
2.2. The graph representation of the Peanuts example. 25
2.3. RDFa graph representation of the resource *me*. 32
2.4. Supervised machine learning process steps 44
2.5. Illustration of a ROC/AUC curve of a synthetic generated and classified dataset 49
2.6. Function plot of cumulative reward per execution step 51
2.7. Screenshot of the SMAPPI game [232] and corresponding MDP created as part of this work to illustrate an example of an MDP. 55
2.8. Learning cycle in RL 56
2.9. An exemplary deep neural network with a word embedding layer. 64

4.1. Big picture of the presented approach. 113
4.2. The definition of a task instance with the guidance of a chat bot web application. 114
4.3. The simulation process between agent and simulator [155]. 133
4.4. The UML sequence diagram illustrates the flow of the simulation and the interaction between agent and simulator. 134
4.5. Process flow of the rule mining approach for deriving policy rules. 141
4.6. Stages, goals and actions of a flight booking task. 144
4.7. UML class diagram representing the datasets that serve as input for the discussed rule mining algorithms. 144
4.8. The automated selection of the rule mining approach. 145
4.9. Example of rule-learning from CART. 147
4.10. Data flow between the presented function. First, the collected data samples are preprocessed, then item sets and ordered action sequences are derived and based on these frequent item sets and ordered action sequences the *Best Fit Sequence* algorithm derives for each sequential task stage the corresponding action sequence. 148
4.11. Retrieval of reusable policies by Entity Embeddings. An agent sends its task entity description to the agent framework. The corresponding DNN is re-trained with the new task entity description. The encoded entity embedding of the new task is stored in the database and used to find the most similar task embeddings in the database using similarity metrics. The top-n of the most similar tasks are returned to the requesting agent who had provided the new task entity description. 156
4.12. Deep Neural Network Architecture for training task and action embeddings. 158
4.13. Process flow and architecture of policies combination approach [152]. . . 160

4.14. The second derivative at single points (here x) indicates whether the underlying function is increasing, decreasing or constant. E.g.: (a) Shows an increasing, i.e. *concave upward*, slope of a function with a *local minimum*. (b) Illustrates that the slope of the function is constant, i.e. *inconclusive*. This may indicate that an *inflection point* is reached. (c) Illustrates a decreasing, i.e. *concave downward*, slope of a function with a *local maximum*. 166
4.15. The sampling of states by inflection points. The dashed lines mark and delimit the samples in the time series data that represent different states of a given feature (y-axis) among time steps (x-axis). A total of 4 states (e.g *SusceptibleDecreased, SusceptibleConstant, RecoveredIncreased, Recovered-Constant*) are sampled in the example, i.e. 2 for each of the two observation features. 168
4.16. A (hidden) Markov model that represents hidden states, state transitions and state observations [152]. 169
4.17. Bimodal data distribution (in blue) and different estimated density curves (in red) by different kernel density estimators and with bandwidth 2. . . 171
4.18. Arrangement of the deep neural network for training entity embedding vectors that represent activity contexts [152]. 173
4.19. The trained embedding vectors of activities, states and actions are merged and visualised in a 3-dimensional space in the Tensorflow Projector. The colouring of the embedding vectors (shown as dots) indicates how spatially far away other embedding vectors are from a selected embedding vector (here, e.g. in light red Wash_drinking_glass_1_Done). The darker the hue of a neighbouring embedding vector, the closer it is to the selected vector. The distances in this illustration were measured with the Euclidean distance [152]. 174
4.20. The general programme to which an agent of the framework is obliged to adhere. 179
4.21. Simulator-based online training of a worker (e.g. DQNN) agent [155]. . . 183
4.22. Adaptation and reuse of policies for performing novel tasks conducted by the consumer agent [155]. 185

5.1. The flight-booking task execution sequence 196
5.2. The two variants of the check-box task. 197
5.3. Button related web tasks. 197
5.4. List Option Task 198
5.5. Transition model of the SEIR [1] categories. 209

6.1. Set-up for the evaluation of automated web tasks. The trained RL agent was previously trained with the simulation component of the proposed agent framework. 224
6.2. Every agent's cumulative reward for every evaluated web task. 230
6.3. Performance of each agent w.r.t. the select list option task [155]. 231
6.4. Performance of each agent w.r.t. the select click buttons task [155]. . . . 232
6.5. Performance of each agent w.r.t. the select click button sequence task [155]. 233

6.6. Performance of each agent w.r.t. the click checkboxes by synonym task [155]. 234
6.7. Performance of each agent w.r.t. the click checkboxes by label task [155]. 235
6.8. Performance of each agent w.r.t. the flight booking task [155]. 236
6.9. Evaluation set-up regarding the policy adaptation scenario. 243
6.10. Evaluation set-up of the agent ensemble approach [152]. 244
6.11. Distribution of the action sequence length (y-axis) among all 52 activities (x-axis) evaluated [152]. 245
6.12. Density plot and histogram of the search radius value ranges (x-axis) in which the most suitable and related action embeddings for submitted, i.e. observed states, were found [152]. 247
6.13. Occurrence of actions and assigned rewards during and after training. . . 248
6.14. Distribution of successfully trained (right bar) and unsuccessfully learned (left bar) activities by the ensemble agent approach after 1 episode and the RL, i.e. DQNN, agent after 1, 10 and 100 training episodes. The colour shades indicate the action sequence length of the respective activities. In contrast to the RL agent, every task performed could be successfully completed by the proposed agent ensemble approach [152]. 249
6.15. Cumulative reward (y-axis) for each activity (x-axis) obtained during combination by ensemble agents (blue dots) and during training by RL agent (red dots) after 1, 10 and 100 Episodes [152]. 250
6.16. Required steps (y-axis) of ensemble agents (blue dots) and RL agent (red dots) per activity (x-axis) until policies are composed or trained. The turquoise dots show the number of required policies among each activity. The plots are scaled and displayed in logarithmic scale to make overlapping values visible in the plots [152]. 251
6.17. Number of incorrect decisions (y-axis) made for each activity (x-axis) during a) policy combination by ensemble agents (blue dots), and b) policy training by the RL agent (red dots) [152]. 251
6.18. Distribution of successfully finished (true value at x-axis, right bar) and unfinished (false value at x-axis, left bar) tasks by the RL and ensemble approach. The RL agent had 3 different training times, i.e. 1, 10 and 100 episodes while the proposed ensemble approach required only 1 episode. 254
6.19. Development of cumulative reward (y-axis) for each performed and numerically encoded activity (x-axis) during training for the ensemble approach (blue dots) in 1 episode and the RL approach (red dots) in 1, 10 and 100 episodes. 254
6.20. Required steps (y-axis) for each performed and numerically encoded activity (x-axis) during training for the ensemble approach (blue dots) in 1 episode and the RL approach (red dots) in 1, 10 and 100 episodes. The turquoise dots show how many actions an activity requires in order to be solved. The plots are scaled and displayed in logarithmic scale to make overlapping values visible in the plots. 255

6.21. Number of wrongly executed actions (y-axis) for each performed and numerically encoded activity (x-axis) during training for the ensemble approach (blue dots) in 1 episode and the RL approach (red dots) in 1, 10 and 100 episodes. 255
6.22. Set-up for rule mining evaluation 262
6.23. A generated MDP for the *Turn on light 15* task. 264
6.24. Setup for task embeddings evaluation 268
6.25. Performance of agents that reused for the web tasks the mined rules. . . 269
6.26. Histograms of rewards that were achieved by utilising the mined rules. . 270
6.27. Virtual Home dataset tasks, executed with mined rules. 271
6.28. Performed actions and rewards of Virtual Home dataset, executed with mined rules. 272
6.29. COVID-19 measures: RL agent performance versus mined rules performance. 274
6.30. COVID-19 measures: Performed actions of each algorithm and received rewards per action. 275
6.31. Development of the SEIR categories based on the imposed measures. Since the exposures overlap with the infections, they cannot be seen in the plots. 276
6.32. COVID-19 measures: GA agent performance versus mined rules performance. 278
6.33. COVID-19 measures: Performed actions of each algorithm and received rewards per action. 279
6.34. Development of the SEIR categories based on the imposed measures. The exposures (yellow) in Fig. 6.34b and Fig. 6.34c overlap with the infections and therefore cannot be seen in the plots. 280
6.35. Embedding Projector showing trained task embeddings. 283
6.36. Go toilet 28 policies trained and applied to Go toilet 60 task. 284
6.37. Turn on light 15 policies trained and applied to Turn on light 16 task. . . 285
6.38. Read book 7 policies trained and applied to Read book 25 task. 286

7.1. Generative adversarial models for task simulation and policy combination. 307

List of Tables

2.1. Confusion Matrix 47

3.1. Comparison of related works and agent framework w.r.t. the cold-start problem. 87
3.2. Comparison of reinforcement learning approaches 91
3.3. Comparison of rule mining approaches. 100

4.1. State concept and its properties. 121
4.2. ObservationFeature concept and its properties. 123
4.3. Topic concept and its properties. 123
4.4. Action concept and its properties. 124
4.5. Effect concept and its properties. 125
4.6. The *Transition* concept and its properties. 125
4.7. VirtualInfluencer concept and its properties. 126
4.8. Strategy concept with its properties. 126
4.9. Policy concept and its properties. 128
4.10. Param concept and its properties. 128
4.11. Equation concept and its properties. 128
4.12. Algorithm concept and its properties. 129
4.13. Agent concept and its properties. 130
4.14. UseCase concept and its properties. 130
4.15. Task concept and its roles respectively properties. 132
4.16. Selection of rule mining algorithm based on task characteristics. 143
4.17. Ontology concepts and properties of the meta data describing log records. 163
4.18. Data Structure with the combined actions resp. policies. The rank determines the quality of the actions based on the reward values obtained [152]. 176

5.1. Characteristics of the use case tasks. 194
5.2. Features of heart disease pathway. 203
5.3. States and rules of the heart disease pathway. For implementation purposes, *InitialState* and *FinalState* have been introduced as states, although they violate the requirement that an MDP cannot be in two different states at the same time. However, it can be said that a context or state consists of all partial states defined here and all partial states are observable in each time step and specify an overall state and context. 204
5.4. Actions and their effects in the heart disease pathway. 205
5.5. COVID-19 Measures with decreasing restriction level. 207
5.6. First configuration of reward assignment based on hospital beds and taken measure. 211

5.7. Second configuration of reward assignment based on hospital beds and taken measure. 211
5.8. Third configuration of reward assignment based on hospital beds and taken measure. 212

6.1. Performance table for every agent and web task with respect to the number of episodes, cumulative reward and mean reward per episode. The bold values show the best results. 227
6.2. The table shows the confidence intervals of the rewards received by the agents per web task. The confidence intervals represent a 95% coverage of the reward values. It can be seen that the confidence intervals do not overlap and thus the results differ significantly. 228
6.3. The table displays the Cohen'd effect size between the different distributions of reward values obtained by the evaluated agents. The interpretation of effect size values is as follows: +/-0.2 till 0.5 small effect, +/-0.5 till +/-0.8 medium effect, > +/-0.8 strong effect. 228
6.4. Average number of incorrect actions performed for each activity within 1, 10 and 100 episodes (see **metric M.2.2.5**). A total of 52 activities were evaluated for each of the approaches considered. The agent ensemble approach required only 1 episode for each task, thus values for 10 and 100 episodes are not given. 251
6.5. Evaluation results of the 10-fold cross validation for the proposed ensemble approach. Here the results of all tasks were considered, i.e. also the ones that were aborted after a certain time because no solution could be found within a predefined radius. 253
6.6. Evaluation results of the 10-fold cross validation for the proposed ensemble approach. Here only the results of completed tasks were considered. . . . 253
6.7. Baseline approaches compared to the proposed policy combination approach in terms of their feasibility and conformance to the Virtual Home dataset activities. It is important to note that the experimental set-ups are different. For this reason, the results are only comparable to a certain extent. 256
6.8. Configuration of initial conditions for the COVID-19 measures task. . . . 262
6.9. Initial model parameters for the COVID-19 measures task. 262
6.10. Comparison of most similar tasks by their embedding vectors. 282

1. Motivation

The progressing evolution of *cyber-physical systems* [10, 174] and *Internet of Things (IoT) technologies* [286] as well as the evolution of artificial intelligence technologies and applications lead to the ubiquity of *intelligent software agents* [33] providing services that support humans in daily life activities within heterogeneous domains, such as e.g. logistics, smart home, entertainment, social life, healthcare, well-being, ambient assisted living, service robotics and many more [260]. These agents take over tasks so that humans can be relieved in their everyday life and work [198, 49].

Different research fields in computer science (e.g. machine learning, data science, data mining, recommender systems, knowledge engineering and discovery) deal with strategies and approaches in order to enable agents to perform such complex tasks. For this reason, agents require different skills for *observing* and *recognizing* their environment in order to *decide* which action to *perform* next so that pre-defined goals can be achieved step by step [198]. This implies that agents have to interpret their observations (data) and have to distinguish if the observed information is relevant or not and how made observations are related to appropriate decisions and actions [210].

An awareness of context (state space) and agent capabilities (action space) is required which means that agents have to know which situations (states) can occur within a task and which actions they are able to perform in a given environment [261, 29]. The mentioned skills are usually provided by machine-interpretable formalisations summarised under the term *knowledge representation* [128, 9]. Moreover, intelligent software agents require the ability to learn strategies, i.e. policies, from their observations, which means that their skills or strategies are based on their made experience [169]. To gather experience and learn from this experience, *machine learning* (ML) techniques are required as a means.

The aforementioned requirements indicate what an agent framework is expected to provide or address in order to support software agents during their runtime. In the following motivational sections, the different challenges that an agent has to deal with are discussed in detail by providing an overview of *how* these challenges are addressed by the contributions of this work.

Section 1.1 discusses the *cold-start problem* [129, 140] that occurs when an agent has no domain-specific prior knowledge at the beginning of its runtime. This problem is particularly relevant in IoT systems where new tasks are deployed on-the-fly or new, unknown users are introduced during the runtime of the system. The interaction and use of services and applications by different people requires that agents can adapt their services and policies at runtime, so that different requirements of tasks and preferences of the users

can be served.

Section 1.2 sets the target users, their context and characteristics of tasks into the focus of consideration and illustrates that context-aware adaptation and personalisation are important, since an agent has to serve for heterogeneous target users and deal with different and changing environments and task characteristics.

Section 1.3 reflects the utility of produced process data and elaborates the benefit of mining and deriving rules from data as well as sharing and reusing the derived rules for solving immediately upcoming tasks.

The research questions (RQ1-3) that are posed and addressed by this work, base on the challenges and problems discussed in the corresponding mentioned sections. Therefore, the stated hypothesis and the corresponding research question (RQ) are presented at the end of each section to summarise the problem. Section 1.4 provides an overview about how this work is structured. Finally, Section 1.5 summarises the key considerations of this chapter.

1.1. The Cold-Start Problem of Agents

Numerous scientific works deal with the life-cycle management of software agents [16, 87, 31]. Every software agent passes a certain lifespan from the beginning of its development and deployment to its execution of tasks and its adaptation to new situations and requirements [295]. Thus, from the beginning of their life cycle, agents should display a well defined behaviour in order to appropriately solve the tasks assigned to them. Therefore, an agent requires an internal knowledge-base that provides, adapts and maintains the required knowledge about the environment, the solvable task and the capabilities of the agent [214, 290]. This domain knowledge is usually provided by domain experts, guidelines, demonstrations or by according use case specific datasets in order to delegate tasks to intelligent software agents [240, 241, 99, 165, 250].

In case this prior knowledge is not available in a formal and machine-understandable representation to an agent, the agent faces the *cold-start* problem which is problematic, especially if unpredictable or critical situations come up. Use cases that often face the *cold-start problem* and require immediately well-defined strategies, concern the personalisation of services, applications and recommender systems, the Internet of Things (IoT) or service robotics [264]. A concrete example of this is when new devices and tasks are deployed during runtime and an agent needs to be able to cope with the new devices and tasks immediately. The rapid emergence of new contexts and environments, or switching between them, may also require an agent to be able to respond adequately to these new contexts and environments at runtime. For example, a service robot in a domestic environment or a factory floor is confronted with different tasks, spaces and possibly people and should be able to react to sudden events and changing contexts in an appropriate way [32, 115]. In order to achieve a well-defined, safe and desired behaviour and task execution

of the agent, a formal (machine-understandable) representation of the required domain knowledge should be provided to the agent.

Domain knowledge depends heavily on the use cases, (see Sec. 5), and goals of the domain experts or the target users. Information about dynamic changes in the environment or sudden context and state changes may sometimes only be observable at runtime of the agent, whereas they are unknown at design time. Moreover, the state and action space of the task may be so huge or even infinite so that not all possible states and best possible actions can be determined and pre-programmed in advance. For instance, if an agent has many alternative actions to choose from in a given state, it sometimes cannot know in advance which of the possible and alternative actions will lead to a desired state, since many unconsidered factors can unpredictably influence the outcome of an action.

Since use cases and objectives can vary due to unknown and changing user preferences, task requirements and stochastic environments, synthetic data, representing significant information to start and learn a task, is required in order to overcome the cold-start problem. Machine learning (ML) strategies empower agents to learn a certain behaviour (sequence of policies) depending on acquired data.

However, before an intelligent software agent is able to learn a desired behaviour or strategy (policy), it requires existing and preprocessed datasets in classical (un-)supervised ML settings and recommender systems. Depending on the application domain, the required datasets might be sparse or either not (immediately) available in a structured and machine-interpretable way or they might lack significance and representativeness for a given problem [178, 204]. In many cases, data sovereignty restrictions and privacy policies prevent the collection of valuable data [111, 271]. Considering for instance the healthcare domain, there are usually only synthetic data available or data that were acquired during a medical field study [53, 41, 257, 281]. However, field studies are costly and it requires time to collect and pre-process the appropriate data. Moreover, if certain tasks have varying preferences, requirements and objectives (for instance due to personalisation), such data are not sufficient significant and useful, because it just represents a tendency inherent to the given field study.

Generalisability and transferability however is a crucial condition for predictions and within evidence-based, process and decision oriented tasks [263, 268, 76]. Trained ML policies require to be applicable and adaptable to changing observations within a given task whereas they allow a well-defined behaviour. Many works [129, 27, 140] address the cold-start problem, especially concerning recommender systems. The objective is to provide for agents a matching default strategy, so that they are enabled to act immediately within an environment in an adequate and intended way.

Other approaches such as *learning by demonstration* [219, 278] are applied with the help of (human) teachers who demonstrate the desired behaviour for a task to the agent, while the agent imitates the performed behaviour [123, 266, 274, 121]. This implies that the agent perceives its environment by demonstrated observations and acts according to the

teachers behaviour. Therefore, usually an interaction between human domain expert and agent is required, as well as a task representation (perception of the task) that the learning agent can process. However, one problem of *learning by demonstration* is that implicit or exceptional state occurrences that are not influenced directly by the demonstrating teacher, are not covered by this kind of learning and cause uncertainty [42], which means that it cannot be guaranteed that the entire state space is covered during the learning phase and that the demonstrations are of high quality [39]. Moreover, teaching an agent without having prior knowledge can be very lengthy and daunting for the teacher, since a long time span can be required until the agent achieves a desired behaviour [24, 288].

Active Learning [222, 225, 56] that propagates a *human-in-the-loop* [177] approach is a further attempt of numerous works [252, 36, 223, 93, 108, 291] for dealing with the cold-start problem. Active learning utilises different, efficient strategies (e.g. dealing with uncertainty and data diversity) for dataset selection and annotation by humans. Thereby, data is iteratively filtered, preprocessed and labelled in order to train and improve ML algorithms. For instance, *crowd-sourcing* [70, 107] platforms facilitate active learning.

However, active learning faces similar problems as learning by demonstration. One problem is given if annotators are uncertain regarding the correct annotation of datasets or if there are contradicting opinions, i.e. disagreements about the right annotation of the data [8, 68]. A further problem is, that annotating data is costly regarding human and computational resources. Furthermore, it is not guaranteed to have experts that correctly annotate the data [67]. Tasks and data that is provided to human annotators have to be described and explained in a way that human annotators can understand the given problem. Furthermore, the motivation of users in annotating data plays a crucial role in getting high qualitative labelled data that can be utilised for training and improving supervised ML algorithms. The focus of active learning lies in supervised learning that is basically tailored towards classification and regression problems. Processes and dynamic changes of task environments are hardly considered by active learning approaches. Nevertheless, active learning can be utilised as basis for personalisation of policies as discussed in Section 1.2.

To tackle the cold-start problem and accelerate policy learning, a formal representation of prior knowledge that can be used to simulate tasks might be helpful in addition to the ML approach. Based on the previous considerations, hypothesis H1 is formulated as follows:

H1 Simulating tasks based on a thorough semantic formalisation of task processes allows, for software agents, a warm-start in terms of a fast adaptation and positive feedback that reflects the performance of the RL agent.

The previous considerations regarding the cold-start problem lead to RQ1:

RQ1 Can the presented agent framework train different kinds of software agents by formal specification and simulation of tasks in such a way that they show a well-defined and beyond state-of-the-art behaviour in solving different tasks from the beginning of their runtime or at least within a short period of time?

This work proposes an approach that helps to avoid the cold-start by means of a semantic model representation comprising guideline rules, semantic task representations and agent profiles that are utilised by a dedicated task simulation framework in order to simulate these processes and decision oriented tasks and train the appropriate agent. Therefore, human domain experts provide their domain knowledge as formal representation of tasks that serves as prior knowledge for the corresponding simulation framework of the proposed agent framework. With this simulated prior knowledge, an agent is enabled to accelerate its learning phase and to overcome the cold-start as this work demonstrates. Contribution **C1.1** addresses **RQ1**:

C1.1 A task simulation framework that allows agents to train policies based on process datasets and domain knowledge.

1.2. Context-Awareness and Personalisation of Agents

Adaptation and Personalisation Valuable expert and process knowledge is hidden in guidelines and (unstructured) data and is reproduced by software agents that learn from human experts and the corresponding data. Since humans can be fallible, ambiguous or biased, this can lead to distorted, incomplete domain knowledge and incorrect, ambiguous rules, as well as undefined and undesirable behaviour from a running agent [151, 258, 259]. Another difficulty in learning policies arises when dynamic, unexpected task requests arise at agent runtime. This mainly plays a role in personalisation, e.g. when the user's preferences and goals change or alternative goals and states occur that require an adaptation of the agent's behaviour [45, 235]. Furthermore, the uncertainty, heterogeneity and complexity of environments, tasks and their immanent processes can make it difficult or even impossible to specify during design time the appropriate interdependencies and requirements [60, 117, 77, 89]. Learned ambiguity in domain knowledge and biased, incorrect rules need to be adjusted or corrected through current feedback. Furthermore, generally applicable guidelines (policies) should be refined and adapted to the specificity of the task and the users.

Personalisation or customisation within certain domains (e.g. advertising, entertainment, healthcare, well-being, service robotics, social networks) is important since provided software services are tailored to individual target users. This implies that individual user characteristics (e.g. preferences, user skills, demographic information) are considered by an agent during the adaptation of policies to user needs [113, 224].

This leads to some challenges for agents because among general characteristics and requirements of a process-oriented task, they have to consider as well the individual user characteristics and to adjust a general policy representation to an individualised policy representation that meets the user requirements, context and task characteristics. However, since user characteristics (profiles) can be diverse or as well have similarities, a requirement is to adapt general policies to individual ones without retraining a new strategy for every new task.

From a developer perspective, it might be cumbersome to re-program or re-align an agent's operational logic and internal decision-making processes for every specific task and context. Moreover, it is impossible to consider all situations and exceptions, which might occur during an agent's life cycle, at its design time. Since a stochastic environment can dynamically change its behaviour due to external parameters (e.g. changing context, changing target user, unknown environmental influences) [112, 185, 189, 7], agent developers cannot hard-code all possible and necessary agent strategies to solve a task during design and implementation time.

Considering the perspective of agent developers, it might be a relief providing a framework that allows agents training their own new policies and adjusting them to changing observations. This implies that developers do not need to implement hard-coded programs that are restricted to certain tasks and environments because the agents are themselves

responsible for training and adapting their policies to their made observations. The underlying requirement and precondition is that agent developers need tools for defining and initialising agents for all kind of tasks. Therefore, it is important to provide a general implementation process that allows a simplified integration into any kind of environment that an agent has to face with.

Hypothesis H2.1 summarises the assumptions that are considered regarding the adaptation of policies:

H2.1 RL agents can deal with incorrect, ambiguous guidelines and adapt their policies (behaviour) either by changing the task representation or by feedback (e.g. datasets) that provide user and task specific information.

The previous considerations lead to RQ2.1:

RQ2.1 Can a RL agent be empowered to correct wrong, ambiguous or unspecific guidelines (i.e. policies) it has learned and adapt them to individual task and user requirements and contextual needs?

This work addresses the discussed requirement by defining *general valid procedures* that every agent requires to implement in order to be framework compliant, interchangeable and applicable for different tasks.

If a strategy (ie. sequence of policies) is learned by the agent, it can share its learned policies with other agents within a multi-agent system by a central database or semantic collaboration framework e.g. *Semantic MediaWiki (SMW)*[1] or even the *Semantic Web* [101]. Other agents can retrieve, adapt and reuse these policies given that they have to solve similar tasks. The adaptation and reuse of policies can speed up and improve the task execution because new policies do not have to be retrained from new and the more agents refine given policies the more tailor-made and optimised strategies for various tasks are generated.

Therefore, this work proposes methodologies for sharing policies and their provenance information in a formal way so that agents can look up these policies and their provenance information. Agents are enabled by this creating formal representations of their trained policies and assessing them in order to collaborate with each other within a multi-agent system.

The following points list the **contributions for RQ2.1**:

C2.1.1 A methodology that combines *reinforcement learning* [242] and rule mining techniques in order to adapt policies and derive adapted policy rules.

C2.1.2 A methodology for the selection of suitable rule mining algorithms that is based on the corresponding task description.

[1] `https://www.semantic-mediawiki.org/wiki/Semantic_MediaWiki`

C2.1.3 An algorithm, for data preprocessing so that the applied approaches can process and mine the given datasets.

C2.1.4 An approach, i.e. *Best Fit Sequence*, for deriving rules for sequential tasks that are divided in stages.

Context-Awareness of Agents Agents operating in complex and stochastic environments also have to be able to cope with different contexts. This means that they require to react immediately and adequately to context changes. In addition, an agent has to choose in every step the action that promises the best, i.e. the most reward-maximizing, result from a set of possible actions. Most often in many approaches [171, 170, 230, 61, 167, 79, 73], reinforcement learning is used to teach agents reward-maximizing policies. However, learning policies for a single task through reinforcement learning may require the agent to perform several thousand training episodes with several thousand action steps until it has developed a strategy for a single task. To act in a complex multi-context environment, an agent would then have to learn all possible policies in advance, which can be very time-intensive or even hardly feasible depending on the size of the state and action space at hand. Furthermore, actions can belong to different contexts and activities, making it more difficult to learn unambiguous and context-dependent policies.

The aim of this work is to enable agents to adapt their trained policies and act appropriately across contexts without having to learn all possible policies in advance. Agents should be enabled to decide at runtime and context-dependently through knowledge already gained or through feedback provided which actions are required.

One consideration of this work is that already learned context-related activities, perceived states and performed actions can be represented by their spatial distribution in an n-dimensional entity embedding space and their context can be determined by their neighbourhood. In order to accelerate the combination of optimal policies, the simulation and parallel execution of potential actions by agents enables the exploration of optimal policies.

The following research question R2.2 and hypothesis (H2.2) summarise the previously discussed considerations about context-awareness of agents.

H2.2 The agent framework can numerically represent the context, i.e. the state and action space of tasks or activities, through entity embedding vectors and, based on the spatial distance of the vectors, narrow down the agent's context as well as speed up policy provision. This is done by selecting action candidates that are semantically close to the currently observed state of the agent and running them in parallel in a simulation to determine which action promises the best result for the current state.

RQ2.2 Can the agent framework compose reward-maximising policies across contexts, i.e. across activities, that contribute to the fulfilment of activities and speed up the provision of policies in terms of less training episodes and steps, compared to agents that apply reinforcement learning, i.e. deep-q-learning neural networks (DQNN), for policies learning?

The **contributions for RQ2.2** are:

C2.2.1 A methodology for representing and transforming task entity descriptions into a processable representation that can be used to calculate similarities between task, state and action entities and to represent and delimit task contexts.

C2.2.2 A methodology for the context-dependent composition of policies that address tasks and activities.

1.3. Mining, Sharing and Reusing of Produced Expert Knowledge

In computer science as well as in other domains, the share and re-use of expert knowledge and strategies is a common method to avoid reinventing the wheel. In order to preserve experience and knowledge, it is necessary that expert knowledge is shared openly and understandably with future generations of people who want to become experts in a particular problem area. Transferring this common principal of sharing and re-using knowledge into the application of agent-based systems leads to the consideration that also agents that learn by experience and by training certain strategies (i.e. policies) can share their strategies to future agent generations that face the same problem domains and have to solve the same or very similar tasks.

For instance, in agent-based systems, during runtime and within real-time environments, agents do not only observe empirical data but also produce valuable empirical data [208] by their policy-driven actions that can be used to derive corresponding valuable and representative agent-based models and rules [118, 125], determining which actions to perform. However, one challenge is to extract useful knowledge from a huge amount of data that contains this hidden knowledge [52] while another challenge is to relate problems and tasks with matching strategies and required (agent) *competencies* [273, 85, 12, 13]. The competencies or capabilities [188, 3, 194, 195] of an agent are determined by its environment and the set of actions it is able to perform. A strategy has to be adequate to the given problem domain (e.g. empathy in social interaction) [150] or task. Thus, a methodology is required that enables agents to be interoperable [193] and compare task properties between different tasks (e.g. in various environments) [90] in order to find policies that fall into the given problem domain.

These considerations lead to hypothesis H3 and RQ3.

H3 Rules derived and formalised from recorded demonstration data can enable agents in a multi-agent environment to solve given tasks on demand without the requirement and effort of training policies from scratch.

RQ3 Are rules and policies obtained via mining techniques from the collected data reflecting behaviour validly applicable and as accurate and effective as the originally executed behaviour inherent to the data, and do the rules derived from the data actually mirror the intended behaviour and enable the agent to achieve a reward maximising behaviour?

This work proposes a methodology that allows agents the comparison of task characteristics with respect to the properties of tasks. The assumption is that similar tasks require similar policies to solve a task. This information in turn can be made available to other agents (for example on the World Wide Web (WWW) or other networked infrastructures). In this way, trained agent knowledge can be shared, reused and adapted to task requirements as needed.

Learned knowledge that is hidden in data can be useful for agents that want to process and solve tasks on-the-fly without cumbersome and daunting pre-processing, parametrisation and learning procedures, because usually, data is not directly usable as source of information due to noise in the data, or missing and incomplete data samples. Therefore, pre-processing, cleaning, feature engineering, dimensionality reduction and feature aggregation are essential prior steps before collected data from different sources can be utilised for machine learning applications. In order to avoid for a newly deployed and starting agent these pre-processing and training steps, the previously trained RL policies can be transferred into concise and significant rules that reflect the essential trained policies of the appropriate RL agent. Moreover, the competencies that are provided by the policies can be formalised so that agents can search for competencies that they can adopt by the appropriate derived rules.

Considering the value of generated operational data, a methodology for the agent framework shall be devised that empowers agents to exploit their produced and observed real-world data in order to derive decision rules and to provide them subsequently to other agents in a formal representation together with provenance data.

In this work, provenance data are considered as meta information that enable agents to request and assess the derived and provided policy rules with respect to their performance. Provenance data mainly contains statistical information about performance and training conditions (e.g. mean reward, standard deviation from the mean reward, training iterations, etc.) as well as information about the authorship and extraction process of the derived rules. Since policies are tailored towards tasks and their requirements, also task descriptions can serve as indicator for the suitability of policies to a given task. Thus, a methodology is required that allows the representation and comparison of task characteristics in order to determine the similarity between them.

On the web, the previously discussed meta information can then be made available together with the rules in a machine-readable form, enabling rule-based web agents to assess and reuse the trained and provided agent knowledge. This way, rule-based web agents no longer have to go through long learning processes and can directly apply the published knowledge. In addition, decision rules are also comprehensible to humans, which is important for the *explainability* and user acceptance of made decisions.

The following points list the **contributions for RQ3**:

C3.1 A methodology that allows the mining of representative decision rules from behavioural, operational and task-specific data.

C3.2 A methodology that allows the formal specification and provision of provenance data and knowledge that can be utilised by agents to assess the suitability of policies.

1.4. Structure of the Thesis

The remainder of this thesis is structured as follows.

Chapter 2 outlines conceptual models and technological foundations upon which this work is based.

Chapter 3 presents and discusses related State-Of-The-Art (SOTA) approaches that have been considered with regard to the contributions of this thesis in order to differentiate the approaches of this thesis from other existing approaches.

Chapter 4 provides for every posed RQ, software-based solutions and demonstrates step by step the agent framework that has been conceptualised, implemented, evaluated and presented by this work.

Chapter 5 illustrates heterogeneous use cases of different application domains (e.g. healthcare, automated web tasks, epidemiology) that were considered for conceptual considerations and later for evaluation purposes.

Chapter 6 presents and discusses the set-up and results of the performed evaluation that was conducted with the presented use cases of Chapter 5.

Finally, Chapter 7 concludes the presented work and achieved results and gives an outlook to future works that cannot be addressed within the scope of this work.

1.5. Summary of this Chapter

This chapter provided a motivation for agent-based solutions and an overview regarding the problems and challenges that intelligent software agents nowadays have to deal with. In detail, diverse problems such as the *cold-start* problem, the *programming* and *on-the-fly integration* of intelligent software agents into heterogeneous infrastructures and environments as well as the *adaptation* of agent strategies to context, user-centric needs and task characteristics have been outlined.

Furthermore, this chapter provided considerations about the requirements (e.g. formalisation and simulation of tasks and phenomena, share of knowledge, etc.) that an appropriate agent framework should address in order to overcome the discussed challenges and problems. It has been discussed that different strategies and paradigms (e.g. ML and knowledge representation and rule mining approaches) are required and have to be combined in order to provide a holistic solution that supports domain experts as well as agent developers and serves for context-aware and user-centric needs. In order to do so, the entire lifespan of a software agent has been considered.

In summary, this doctoral thesis presents a reference architecture for an agent framework that supports the entire lifespan of software agents, i.e. deployment, learning, execution and provision of strategies to other agents. The aim is to (semi-)automate all the steps that an agent has to go through with the help of the presented framework. The focus is on the acquisition, adaptation and provision of openly available and machine-interpretable knowledge for agents. The agent framework is intended to promote the expansion and improvement of process knowledge and strategies as well as the cooperation between agents in IoT systems.

2. Foundations

This work builds upon different foundational concepts and technologies comprising in particular three research fields, i.e. data mining, machine learning (ML) and Semantic Web technologies. The main objective of this chapter is to provide a basic understanding about the used technologies within the presented approach.

Since this work serves for agent-based solutions, this chapter outlines the definition of an agent and its environment and considers different agent programs that follow different implementation paradigms. Section 2.1 introduces the commonalities and differences of agent programs in order to provide a categorisation of agents that are considered in this work.

The conceptualised and implemented agent framework consists of different software components that utilise semantic technologies such as RDF(S), SHACL, JSON-LD and RDFa for annotating, representing and sharing the required domain knowledge, comprising the different model representations (i.e. agent profile, task profile, observation topics, policies, etc.).

Section 2.2.1 gives a basic introduction into RDF(S), SHACL and RDFa which are W3C recommendations for serialising semantic knowledge graphs.

The formal description of concepts was partially represented within scientific publications by description logics (DL). Section 2.2.2 gives a basic introduction into DL constructs that allow concisely representing concepts and deducing implicit knowledge.

As it has been outlined in the introduction, the devised simulation framework utilises domain specific guideline rules in order to generate numerical state representations of tasks that can be processed by ML algorithms. Furthermore, agents shall be enabled to describe their policies by means of rules that can be reused by other agents within a multi-agent system. For this reason, this work applies interchangeably the W3C recommendations Semantic Web Rule Language (SWRL), Notation3 (N3) and SPARQL in order to represent and reason rules that serve for the definition of states and task policies. Sections 2.2.3, 2.2.4 and 2.2.5 discuss the characteristics of SPARQL, SWRL and N3.

The considered agents communicate with the environment via sensing and acting devices and messages. Therefore, communication means are required that allow a shared understanding and interoperability between different devices and software components. This work considers IoT and WoT concepts and infrastructures in order to introduce appropriate

communication technologies in Section 2.2.6 and 2.2.7.

Since the proposed agent-framework applies ML in combination with semantic technologies, Section 2.3 gives a brief overview regarding common ML paradigms and approaches to lay the foundations for a better understanding regarding the made design decisions and to outline why RL, compared to other ML approaches, is predestined for agent-based systems.

In the area of ML, tasks are described by process models, called *Markov Decision Process (MDP)*. This decision process model among others is an abstract representation of complex task processes by probabilistic state-action diagrams. Section 2.3.9 provides the formal specification of MDPs.

Especially, intelligent agents that apply RL in order to train task-specific policies, are enabled to process tasks by means of an internal MDP model representation. Section 2.3.10 illustrates the general idea of the RL paradigm.

Since ML algorithms require to be parametrised preceding to their execution, *hyper-parameter optimisation* [287] is an important topic that is applied within the framework in order to find optimal parameters for the utilised RL algorithm. Section 2.3.11 deals with two hyper-parameter optimisation methods that are grid- and random search.

As outlined in Section 1.3, the numerical representation of contexts and the similarity measurement of task specifications provides a means for policy search. For this reason, word embeddings are a foundational approach that are introduced in Section 2.3.12.

Agents may implement various algorithms in order to learn strategies and make decisions. Genetic Algorithms (GAs) are one of these algorithms that are applied in the agent framework and are discussed in Section 2.3.13.

The extraction of decision rules for sharing and reusing policies is conducted by different decision rule mining approaches. Section 2.4 discusses four approaches (e.g. frequent item sets, association rules, OneR and sequential covering) that allow the derivation of rules from huge datasets.

Finally, Section 2.5 summarises what has been outlined in this chapter.

2.1. Agent-based Systems

The following definitions and classification of agents and environments are adopted from the AI books [213] and [198].

2.1.1. Properties of Intelligent Agents

Russell and Norvig define the terms *rational* and *intelligent* agent and use them synonymously. *Rational agents* are agents that perform the *right* or intended actions. In order to assess what is right or wrong, an agent requires a *performance measure* that evaluates the entire sequence of environmental states that were generated by the agent's actions. It is important to notice that not the agent's internal states are assessed by the performance measure but the environmental states, because an agent could assess its performed actions always as right in order to maximise its performance measure. Therefore, a performance measure has to be independent from the agent's internal states and has to follow the intended behaviour that is expected from an agent. According to Russell and Norvig, *rationality* depends on four criteria [213]:

- The performance measure that defines the success criterion.
- The prior knowledge about the environment.
- The actions that an agent can perform.
- The sequence of agent perceptions.

Considering these criteria, [213] provides the following definition regarding rational agents: *"A rational agent shall select for every sequence of perception, an action which is expected to maximise the performance measure if the sequence of perception as well as the existing knowledge of the agent are considered."*

According to Russell and Norvig, intelligent or rational agents can be categorised into five different types: *simple reflex agents, model-based agents, goal-based agents, utility-based agents* and *learning agents* [213]. All agents have in common the way in that they perceive and interact with their environment.

Intelligent agents perceive their environment via sensors (e.g. cameras, infra-red sensors, etc.) and act within this environment via available actuators while they influence their environment via their actuators. For instance, *software agents* [198] perceive their environment by the peripheral equipment (e.g. keyboard, mouse, file content, network packages, etc.) of a computer and impact their environment by updating or generating file contents, making outputs to the user interface (UI) or sending network packages.

The agent's *sequence of perception* is considered as course of all perceptions that the agent has made during its life time. The selection of an action depends only on the made observation. This leads to the implication that the *behaviour* of an agent is determined by an *agent function* that specifies all possible courses of perceptions and assigns them to the appropriate executable actions. The agent function is usually implemented by an *agent program* [213]. However, both terms have to be distinguished. While, the agent function is a mathematical abstraction, the agent program is a concrete implementation of the agent function that is executed within a *physical system* [198, 213].

Figure 2.1 illustrates a closed interaction loop between an agent and its environment. First, the agent observes its environment via its sensors, then decides via an agent function its next action and finally performs via its actuators the appropriate action that impacts again the environment and leads to new state changes and due to these to new observations.

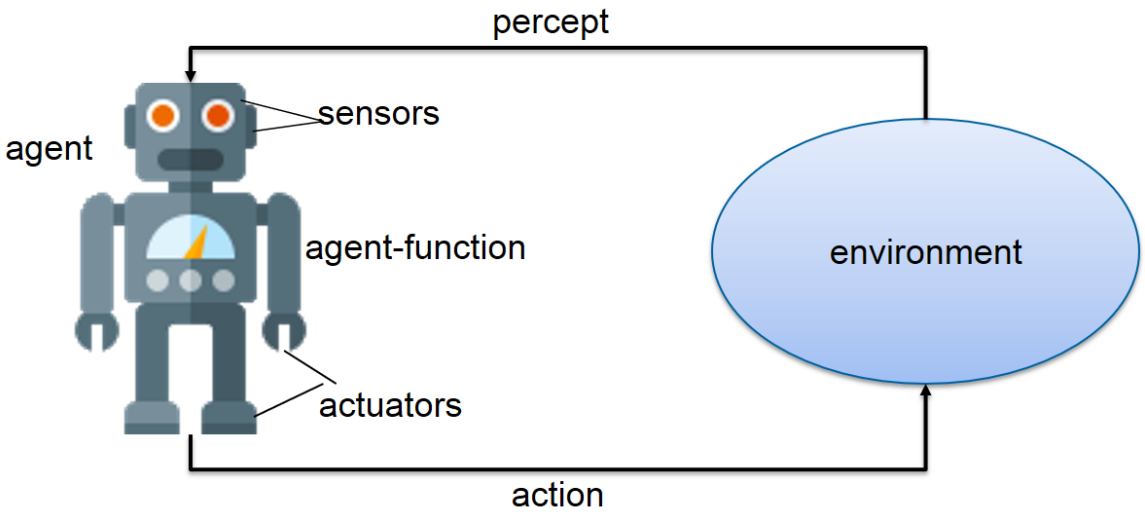


Figure 2.1.: The interaction of an agent with its environment.

Russel and Norvig make in their AI book a distinction between *rationality* and *omniscience*. In the real world omniscient agents are not possible because this would mean that agents know in advance what effects all their actions will have to an environment. Since this is not possible, agents perform actions in order to *explore* environmental states and the effects their performed actions might have to the environment. *Exploration* is an important means for agents, to collect information and learn a useful behaviour. This implies that rational agents *learn* from their perception based on their explorations.

If an agent only relies on the knowledge and experience of its developer and does not consider its own perceptions and experience, then it has a lack of *autonomy*. However, a rational agent should always be autonomous in order to compensate incomplete or erroneous knowledge [213]. Incomplete and erroneous knowledge can result among others, from unavailable domain knowledge, noisy sensors and environmental characteristics that hinder the agent to observe all relevant information [213, 198, 186, 54, 285].

It depends highly on the environment and agent program, how an agent is implemented [198]. However, the different agent programs may have, nevertheless, overlapping characteristics as it will be outlined later.

According to Russell and Norvig, an agent consists of an *architecture* and an *agent program* that implements the *agent function* for the assignment of observations to actions. An architecture is the platform (e.g. computer, robotic platform, etc.) with its periphery (e.g. sensors, actors). The agent program and architecture have to match to each other because the program requires to apply the appropriate architectural components. Later in

this chapter, the different fundamental agent programs (simple reflex, model-based reflex, goal-based, utility-based and learning agent) will be described.

2.1.2. Properties of the Task Environment

Since an agent acts within an environment that is determined by the assigned task, we consider now the *task environment* and its characteristics. According to [213], an environment can be evaluated by the following *PEAS* description. PEAS stands for *Performance, Environment, Actuators* and *Sensors*. *Performance* defines the criterion for evaluating the agent's performance. *Environment* describes the agent's environment. The *sensors* and *actuators* specify the means for observing and acting within the environment. In the conceptualisation of an agent-based system, the PEAS description can be used by agent developers to determine characteristics of the considered agent tasks [213].

Task environments are classified by different criteria. One of them is the *observability* of an environment. Environments can be either *fully observable*, *partially observable* or *not observable* [213]. It depends on the utilised sensors or the lack of them what kind of observability is given. For instance, sensors can be noisy, missing or provide just partial or unreliable information.

Another criterion is the *determinism* of an environment. An environment can be either *deterministic* if it is given that the next state is determined by the current environmental state and performed actions or *stochastic* if uncertainty is given regarding the performed action and next state. This implies that in a stochastic or indeterministic environment, the probabilities of action and state occurrences are unknown to the agent and its developer [213].

Furthermore, an environment is determined by the number of agents that act within this environment in order to solve a task. Given that only one agent acts within an environment, it is a *single-agent environment*, otherwise if more than one agent act within that environment, it is a *multi-agent environment*. In a multi-agent environment, agents can collaborate with each other in order to solve a task and influence mutually the states of the environment.

The challenges of multi-agent environments are different to the challenges of single-agent environments. In multi-agent environments the communication between the agents is important. Moreover, it has to be distinguished if there are *cooperating* agents or *competing* agents. It depends on the objectives that the agents might have and if their actions maximise or minimise the performance measure of each other. Cooperating agents maximise mutually their performance measures, while the actions of competing agents impact diametrically their performance measure.

Additional properties are utilised for the classification of task environments. For instance, an environment is *undefined*, if the environment is either stochastic or not fully observable.

In *episodic* environments the perceptions and decisions of an agent are independent from previous perceptions and decisions, while in *sequential* environments, decisions and perceptions can influence subsequent perceptions and decisions. Episodic environments are easier to implement and maintain because the agent does not require the ability to look ahead regarding the effects its actions might have in future.

An environment is *dynamic* if it changes, while the agent is making a decision regarding the next action, otherwise the environment is *static*. Given that the environment does not change but the agent's performance measure changes according to the performed actions, then the environment is *semi-dynamic*.

Regarding environments, there is also a distinction between *continuous* and *discrete* environments. In order to make this distinction, the states of the environment, the time, the perceptions and actions have to be considered. Continuous environments provide usually an infinite number of states. The states themselves are described by continuous values, while in a discrete environment, a finite number of states and actions is given. Moreover, the state and action value representations are as well discrete (e.g. pixels of an image).

The knowledge level of agents about an environment and its physics determines if an environment is *known* or *unknown* for agents. Given that an environment is unknown, the agent requires to learn how the environment works and how to act within this environment.

Finally, an environment can be *simulated* in order to train agents within heterogeneous environment classes. A simulation, which can be considered as an environment generator, implements the physics and characteristics of the environment, so that agents can learn to act useful within that environment.

2.1.3. Simple Reflex Agent

Simple reflex agents are the simplest implementation of an agent program. They select their actions based on their current observation and do not consider the previous course of perceptions. Simple reflex agents react reflexive by following condition-action rules, represented by simple *if-then* clauses. The program of a simple reflex agent can be considered as a collection of condition-action rules that are triggered by certain observations, matching certain conditions. It is important to note that only the current observed state is relevant for the agent. Due to their simplicity they are limited in their intelligence. Furthermore, it has to be taken into account that simple reflex agents can act only in *fully observable* environments because all perceptions and according action-assignments have to be known beforehand by the agent.

2.1.4. Model-Based Reflex Agent

Model-based reflex agents are agents that maintain an internal model representation that is influenced by the agent's course of perception. This allows model-based reflex agents to act within *partially observable* environments, since the model representation provides

the required knowledge about how the world they act in, changes independent of their actions. In addition, this model representation reproduces the knowledge about the effects that the agent's actions have to the world in question. It is important to notice that the agent cannot exactly determine the current state of the world in a partially observable environment. Therefore, it has to follow its internal model in order to make assumptions about the current state of the world and decide based on these.

2.1.5. Goal-Based Agent

In order to solve a task, an agent requires a goal that is formalising the intended outcome of the given task. A goal-based agent can combine this goal information with a given model representation in order to select the actions that contribute to the according goal state. Finding a solution (sequence of actions) that lead to the objective can be either easy or complex depending on how many actions have to be considered and performed in order to a achieve an appropriate goal. Goal-based agents have to apply planning or search algorithms to find certain action sequences. They require to know which effects their actions might have in future and this distinguishes them fundamentally from simple reflex agents which only consider the current perception. Goal-based agents are much more flexible because their knowledge can be explicitly represented and updated so that an adaptation of behaviour patterns is possible. In order to do this, it is only required to change the goals of the agent.

2.1.6. Utility-Based Agent

Goals can be achieved on different ways. However, not every way is desirable and it might be as well important *how* a goal was achieved. That is the point when utility-based agents come into play. A utility-based agent, evaluates by a *utility function* the utility of its actions depending on the perceived environmental states. The objective of the agent is to maximise its utility by performing actions that lead to desirable states.

The utility of the agent is determined by the states the agent faces during its execution. It is required that distinct states can be assessed based on a given performance measure. This performance measure usually provides a numerical value that is computed by evaluating the performed sequence of actions and achieved sequence of states. In this way, more or less desirable action paths can be distinguished. The internal utility function and external performance measure require to match so that an agent can be classified as a rational utility-based agent [213].

Utility-based agents have the advantage that they allow different interim solutions and trade-offs, in the case that a task has two or more conflicting goals. Moreover, utility-based agents enable the estimation of success probabilities versus the importance of goals. These both features allow utility-based agents to decide under uncertainty. The success of utility-based agents depends on various capabilities. They have to be able to represent internally a model, to perceive and track their environment, to reason and learn their behaviour in order to maximise their utility.

2.1.7. Learning Agent

The ability to learn enables an agent to act in unknown environments and to improve its behaviour in relation to the initial behaviour that is based on its initial knowledge. To equip an agent with the capability to learn, four conceptual components that fulfil different tasks, are required. The *learning component* has the task to improve the agent's decisions, while the *performance component* has the task to select actions based on the agent's perceptions. The *actor-critic component* provides feedback regarding the outcome of a performed action. Based on this feedback, the learning component changes the performance component in order to improve the outcomes of the agent. The actor-critic component is required because the perception all alone is not sufficient in assessing the success of the agent. The last component, the *problem generator* is responsible for suggesting the next actions in order to explore new information that contributes to the learning and improvement of the agent. *Learning* means in this context a process that allows changing all mentioned agent components in order to align the performance of the agent with the received actor-critic feedback. The objective is to improve the agent's overall performance.

2.2. Semantic Technologies

The idea of the Semantic Web was first proposed and introduced by Tim Berners-Lee, the inventor of the World Wide Web (WWW).

The objective of the Semantic Web is to enable machines (computers) to interpret and understand the meaning of annotated web content. Programs are enabled to use formalised real world knowledge in order to solve tasks in heterogeneous domains and fields of application. This is conducted by semantic vocabularies and ontology languages that allow devising and annotating resources on the web with meta data in order to add machine-interpretable meaning to these web resources.

Moreover, the provision of web standards leverage many means to create and publish formal knowledge representations that contribute to a shared conceptualisation and understanding between different hardware and software components. For this reason, the World Wide Web Consortium (W3C) standardises these ontology languages to facilitate the growth of the *Web of Data* or *Linked Open Data (LOD)*, formerly and still known as Semantic Web.

Basically, the idea of the Semantic Web is to interlink web resources by properties in order to provide facts or descriptions about these resources. Both the resources as well as the relations between them are used to build knowledge graphs, infer the different kinds of relationships (e.g. subsumption, instance-of-relationship, etc.) and make implicit knowledge explicit.

In this work, knowledge representation within the proposed agent framework is implemented by ontologies and semantic knowledge graphs. In order to devise ontologies and knowledge graphs, several serialisations and web APIs (e.g. URI, HTTP, XML, JSON, etc.) are available and recommended by the W3C. This section deals with the foundations of different utilised semantic technologies such as the Resource Description Framework (RDF) that serves for representing knowledge graphs by linking facts. RDF can be formalised and combined with different serialisations, such as the Terse RDF Triple Language (Turtle), the Shape Constraint Language (SHACL), JSON-LD and many more.

In this work, Turtle and SHACL are proposed for defining a vocabulary that represents relevant knowledge for agents and the dedicated agent framework. SHACL is a means for defining shape definitions with constraints for relevant concepts and their properties. This feature of SHACL assures in contrast to plain RDF a *closed world assumption* that is required in order to avoid undefined factual circumstances on the A-Box level. The shape constraints of SHACL allow beforehand to evaluate RDF instances regarding completeness, cardinality and typisation. In this way, missing facts and constraint violations can be reported to the creator of RDF instances.

In this approach, the considered agents exchange messages during their interaction with the agent framework and within heterogeneous task environments. For this purpose,

the agents communicate via JSON-LD messages in order to have a shared understanding between all participating components.

Moreover, agents of this approach publish their policy provenance information by different semantic serialisations. One of them is RDFa, that is used for annotating HTML pages semantically. In this way, the appropriate provenance information can be shared by the agents with other semantic agents that crawl the web in order to search for policies that match to their task requirements.

Since an ontology was defined for the proposed agent framework, a tool was needed to concisely describe the semantic concepts of the developed vocabulary. Description logics (DL) were used because DL in its formalism allows for a higher expressiveness (e.g. conjunctions, disjunctions, negations of terms) than simple RDF constructs.

The policies that the agents learn for a certain task, are shared by them via alternative rule and query languages. The agent framework uses different languages such as SPARQL CONSTRUCT queries, SWRL and N3 rules interchangeably, as heterogeneous agents should be provided with alternatives for sharing, reusing and reasoning about task policies.

The Web of Things (WoT) that is planned to become a W3C recommendation, can be considered as a semantic layer for the Internet of Things (IoT). The WoT allows to abstract relevant IoT device functionalities and properties, so that heterogeneous hardware and software components of an IoT infrastructure can cooperate. The proposed approach applies the concepts of the WoT in order allow agents to interact interoperable with their environment and independently of proprietary protocols and APIs.

The foundational description of the considered semantic technologies relies on the books [100, 102] of Hitzler et al. as well as on the W3C specifications that are openly accessible on the web. The objective of this section and its subsections is to provide a basic understanding of the Semantic Web concepts and technologies that this work builds on.

2.2.1. RDF(S), Turtle, SHACL, JSON-LD and RDFa

The basic building block for semantic technologies is RDF. RDF is a framework for representing resources or entities by Uniform Resource Identifiers (URIs) or Uniform Resource Locators (URL). These web constructs allow to assign a unique identity to resources on the web in order to make their semantic representation by the Hyper Text Transfer Protocol (HTTP) accessible. RDF triples or statements consist of subject, property and object constructs that allow to express facts about states of the real world by building relations between resources. In addition, properties allow to assign literals to resources in order to characterise these resources. The literals may be typed (e.g. string, float, double, boolean, etc.) or untyped. By means of type specifications of the XML Schema Definition, types can be assigned to literals. It has to be taken into account that literals have no external existence within the web. They are just locally related to a resource and therefore not

represented by publicly accessible URIs.

Figure 2.2 depicts a graph representation and Listing 2.1 shows the corresponding Turtle example of RDF triples that express the facts that Snoopy is a beagle (line 4), that has a height of 60 centimetres (line 5). Snoopy's height is described by a literal of the XSD type integer. Further, Snoopy's owner is Charlie Brown who is a cartoon character (line 6 and 9). Snoopy's friend is Woodstock, a canary bird (line 7-8). In this way factual knowledge about entities of the real (or in this case fictional) world can be provided. Every triple element despite literals and blank nodes, is represented by URIs. The prefix tags allow to define abbreviations of name-spaces. Instead of writing the entire domain name-space, the abbreviation tag is taken for representing the domain of the entity. In the depicted example (line 1-3), we have three different name-spaces: rdf, xsd and the base or local name-space that does not require a reference tag. The *rdfs:label* property allows to assign among the URI of a resource also a human readable name in a certain language (e.g. English), which is indicated for instance by the *@en* tag (line 10).

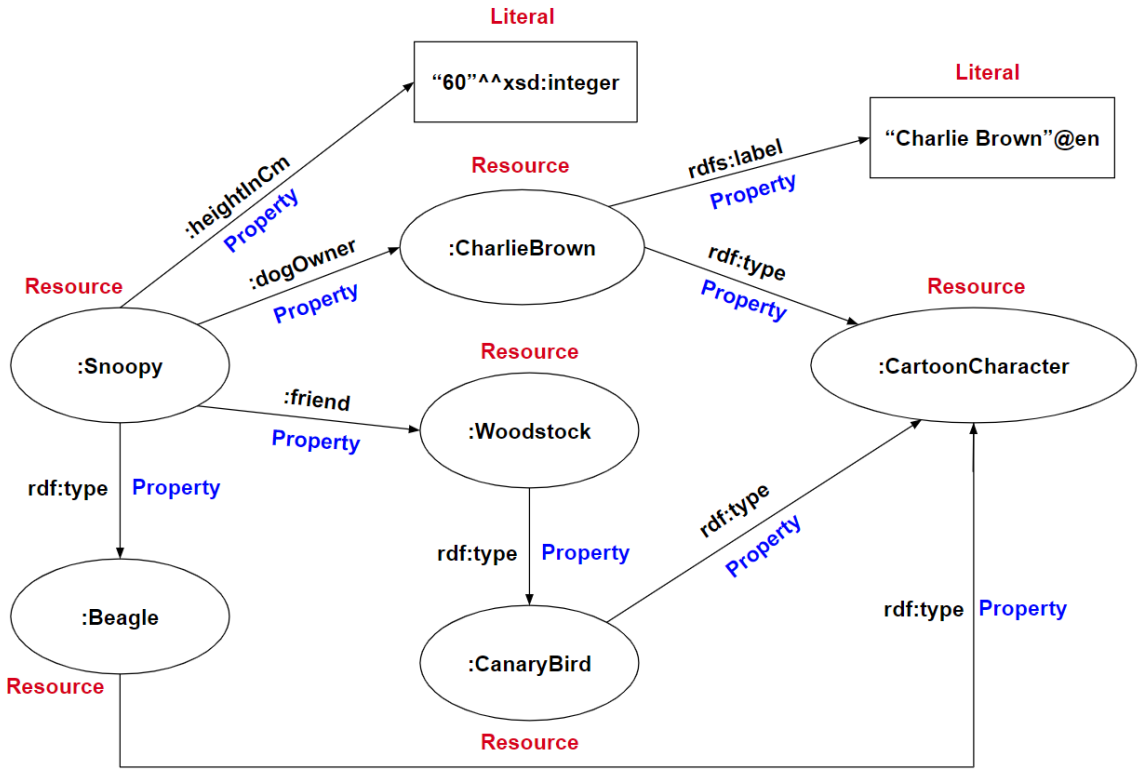


Figure 2.2.: The graph representation of the Peanuts example.

Listing 2.1: RDF triples in Turtle format.

```
1 @prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
2 @prefix xsd: <http://www.w3.org/2001/XMLSchema>.
3 @prefix : <http://peanuts-ontology.org>.
4 :Snoopy rdf:type :Beagle;
```

```
5       :heightInCm "60"^^xsd:integer;
6       :dogOwner :CharlieBrown;
7       :friend :Woodstock.
8  :Woodstock rdf:type :CanaryBird;
9  :CharlieBrown rdf:type :CartoonCharacter;
10           rdfs:label "Charlie Brown"@en.
```

In order to model and deduce what meaning and implications the relations between resources have, RDF Schema (RDFS) is required. It allows to create hierarchical structured concepts and properties of light-weight ontologies by class and property definitions. The following Listing 2.2 illustrates how the model schema, which means the concepts and properties of the previous Snoopy example, can be specified by RDFS.

Listing 2.2: RDF Schema of the Peanuts example.

```
1  @prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
2  @prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
3  @prefix : <http://peanuts-ontology.org>.
4  :CartoonCharacter rdf:type rdfs:Class.
5  :Beagle rdfs:subClassOf :CartoonCharacter.
6  :CanaryBird rdfs:subClassOf :CartoonCharacter.
7  :friend rdf:type rdf:Property;
8        rdfs:domain :CartoonCharacter;
9        rdfs:range :CartoonCharacter.
10 :heightInCm rdf:type rdf:Property;
11          rdfs:domain rdfs:Class;
12          rdfs:range rdf:XMLLiteral.
13 :owner rdf:type rdf:Property;
14       rdfs:domain :CartoonCharacter;
15       rdfs:range :CartoonCharacter.
16 :dogOwner rdfs:subPropertyOf :owner.
```

First, a class is defined and then the sub-class relationships (line 4-6). For instance, *CartoonCharacter* is of type *rdfs:Class* (line 4) and *Beagle* is a sub-class of *CartoonCharacter* (line 5). The same principle is applied with the property definitions (line 7-16), whereby a property can be sub-property of other properties (line 16) and have in addition, domain-range relationships.

The *rdfs:domain* property specifies which kind of subjects a property has, while the *rdfs:range* property specifies which type of objects are assigned to the property. By means of this kind of conceptualisation already simple logical inferences can be applied. For instance, the class membership of an entity can be derived from the domain or range of its properties. Moreover, the super-class membership of an entity can be deduced by the specified class hierarchy. Additionally, new facts can be derived by sub-property relationships. However, the possibilities are limited and therefore, depending on the complexity of the domain model, more expressive languages such as OWL-Lite, OWL-DL, SHACL or

rules might be required.

As already outlined, the proposed framework utilises RDF(S) because the agent-framework's model representation is not that complex and requires only light-weight ontology language constructs. This is due to the fact that complex semantic reasoning is not applied within the framework. In this case, the main purpose of the ontology is to provide task representations in order to simulate the states that occur within a task. Basic reasoning tasks are only performed within the task simulator in order to recognise and update states and their according desirability after actions have been performed by agents. The designated agents only predict after their training the appropriate actions that they have to perform according to their trained policies.

Reification is a further helpful means that is supported by RDF. It allows to define statements about statements. Reification can for instance be utilised for providing provenance information about resources. Listing 2.3 shows an example about how reification can be applied with the help of RDF.

Listing 2.3: Reification example.

```
:AgentX :proposes :Policy1.
:Policy1 a rdf:Statement;
       rdf:subject :CoffeeMachine1;
       rdf:predicate :switch;
       rdf:object "OFF".
```

In Listing 2.3, an agent (AgentX) instance proposes via a policy (Policy1) statement to switch off a certain coffee machine (CoffeeMachine1). Reification is as well applied within this approach, when the agents provide provenance data about their trained policies.

We have seen now the basic building blocks of RDF. In the next paragraphs SHACL, JSON-LD and RDFa are discussed.

2.2.1.1. SHACL

The Shapes Constraint Language (SHACL) is a W3C recommendation[1] of July 2017. Its main purpose is to enable the validation of RDF graphs by means of individually defined constraints. This implies that SHACL allows the definition of constraints on RDF resources and their properties by means of shape constraint definitions. According to the referenced W3C specification, SHACL shapes can as well be utilised for data integration, code generation and user interface building.

On the one hand, SHACL is an alternative to RDF/OWL and on the other hand it can be utilised together with these both formats. Furthermore, the SHACL core function can be

[1] https://www.w3.org/TR/shacl/

extended by extension-mechanisms[2] such as SPARQL and JavaScript[3] constraints that provide additional functionality that can be not covered by the SHACL core concepts.

SHACL provides as well the opportunity to define rules that allow the reasoning of new facts. The rules can be described in three different formats: SPARQL, JavaScript and triple rules.

The basic principle of SHACL is that shapes specify targets for graph nodes. A shape can be considered as a collection of constraints that are applied to certain graph nodes. The shape constraints can for instance determine the type of properties. There are built-in constraint types and user-specified constraints by SPARQL or JavaScript. By means of the stipulated constraints, it is possible to validate beforehand if user-driven inputs and instantiations of data are correct.

Furthermore, SHACL enables to search for RDF or OWL instances that match the shape definitions. The shape constraints can be defined in a declarative manner, and the appropriate shape instances can be shared as linked data so that they are reusable by other applications.

It is also possible to define and implement custom functions that are designated to modularise and solve certain (micro) tasks (e.g. conversion of currencies and temperatures, etc.). The appropriate shape functions can then be queried by different applications on the web. Listing 2.4 shows an example of a SHACL shape definition.

Listing 2.4: SHACL Shapes.

```
1  ex:StudentShape
2    a sh:NodeShape ;
3    sh:targetClass ex:Student ;
4    sh:property [
5      sh:path ex:registerNo ;
6      sh:maxCount 1 ;
7      sh:datatype xsd:string ;
8      sh:pattern "^\\d{3}-\\d{2}-\\d{4}$" ;
9    ] ;
10   sh:property [
11     sh:path ex:studies ;
12     sh:class ex:University ;
13     sh:nodeKind sh:IRI ;
14   ] ;
15   sh:closed true ;
16   sh:ignoredProperties ( rdf:type ) .
```

In the given listing, a student shape is defined. In line 2, the *StudentShape* is declared as being of type *sh:NodeShape*. The RDF target class is *ex:Student* that is defined in line 3.

[2] https://www.w3.org/TR/shacl-af/

[3] https://www.w3.org/TR/shacl-js/

Then the *sh:property* is defined where all constraints are listed for the appropriate property *registerNo* (line 4-9). The *registerNo* property has a cardinality of one which means that only one value can be assigned to the *registerNo* property. The data type of the property is string and the according value has to comply to a regular expression pattern that is assigned to (*sh:pattern*). In this way SHACL is compliant to RDF(S) and provides means for enabling constraints and validating the conformance of RDF instances to shape definitions.

For in-depth details regarding the definition of SHACL shapes, please refer to the appropriate web resources[4].

2.2.1.2. JSON-LD

JavaScript Object Notation for Linked Data (JSON-LD) is a serialisation format for Linked Data (LD) that was recommended in the version 1.0[5] by the W3C. In the meantime the version 1.1 is available as W3C editor's draft[6]. JSON-LD can be considered as an extension of JSON.

The purpose of JSON-LD is to transmit documents between different (web) applications and to utilise and support LD. JSON documents consist of key-value pairs. In JSON-LD additional keys or tags are introduced (i.e. @context, @id, @type, @value, @language, @graph, etc.) that support the LD principles.

Listing 2.5 shows a JSON-LD document that provides appropriate keys. First, within the document, the context is specified (line 2-22) through the appropriate key words (person, name, company_de, role_en) and their *id* and *type* definitions. Then, a concrete person instance (i.e. Nicole Merkle) is defined that is described by the previously introduced keys and their value initialisations (line 23-27).

As Listing 2.5 shows, JSON-LD also supports internationalisation by the @language tag (line 15 and 20). For instance, the keys *company_de* and *role_en* reference German and English string values. In this way, different languages can be supported. Furthermore, JSON-LD allows key aliasing. Line 7 shows an example for aliasing the id key as url.

Listing 2.5: JSONLD example.

```
1 {
2   "@context": {
3     "Person": {
4       "@id": "http://schema.org/Person",
5       "@type" : "@id"
6     },
7     "url": "@id",
8     "name": {
```

[4] https://www.topquadrant.com/technology/shacl/tutorial/ and https://www.w3.org/TR/shacl/
[5] https://www.w3.org/TR/json-ld/
[6] https://w3c.github.io/json-ld-syntax/

```
9       "@id": "http://schema.org/name",
10      "@type": "xsd:string"
11    },
12    "company_de": {
13      "@id": "http://schema.org/company",
14      "@type": "xsd:string",
15      "@language": "de"
16    },
17    "role_en": {
18      "@id": "http://schema.org/role",
19      "@type": "xsd:string",
20      "@language": "en"
21    }
22  },
23  "@type": "Person",
24  "url": "http://www.fzi.de/former_employee/NicoleMerkle",
25  "name" : "Nicole Merkle",
26  "company_de" : "FZI Forschungszentrum Informatik",
27  "role_en": "PhD Student"
28 }
```

The previous example illustrates that within the @context specification the utilised object-keys and property-keys are defined by their @id and @type definitions in order to enable short key names without having the necessity to list long URIs.

The type of a key can be either an object type (e.g. Person, Affiliation, etc.) or a data type (string, date, integer, etc.). Data types can be specified with a concrete value (e.g. 2019-06-17) by the @value tag.

According to the LD principles, the URIs help to reference for instance RDF resources on the web and disambiguate similar keys, having a different meaning. Therefore, different property and object definitions from different web resources can be utilised together which contributes to interoperability between different applications.

The context specification can be provided in a separate JSON-LD document and can be included into other JSON-LD documents by referencing the URI of the context specification file in the @context tag. Therefore, the appropriate context definition has to be publicly accessible on the web.

There are two ways of referencing LD objects, either by *embedding* the objects directly into the referencing document or by *referencing* an URI that links to the JSON-LD document, containing the object specification.

In this section, a basic introduction into JSON-LD was given. JSON-LD provides many more features. For further details, please refer to this web resource[7] or the previously mentioned W3C pages.

2.2.1.3. RDFa

Resource Description Framework in attributes (RDFa) is a markup extension of HTML5, XHTML and XML-based (e.g. SVG) web pages for annotating them semantically in order to provide structured meta data regarding their content. Web browsers, web services and web crawlers of search engines can utilise this semantically enriched data in order to gather useful and valuable information and answer for instance specific search inquiries.

RDFa in version 1.1[8] is a W3C standard of March 2015. The objective of RDFa annotations is to provide machine-readable data within human-readable web pages. Therefore, RDFa provides a set of mark up tags (i.e. @vocab, @typeof, @resource, @property, etc.) that can be applied for exactly this purpose. Listing 2.6 shows a snippet of a HTML page annotated with RDFa.

First, in the body-tag the used schema.org vocabulary is referenced (line 3). Then within a div-tag the referenced resource (me) is specified with the *typeof* property as type *Person* (line 4). Every RDFa property tag within the span-tags stipulates the properties (i.e. fullName, birthdate, url) of the previously defined resource (line 5-7). The RDFa tags can be utilised together with every kind of HTML tag. In this way arbitrary complex RDF graphs can be generated that provide the appropriate knowledge for web applications such as e.g. web agents.

Listing 2.6: RDFa example.

```
1  <html>
2    <body vocab="http://schema.org/">
3    <div typeof="Person" resource="#me">
4      <span property="fullName">Nicole Merkle</span>
5      <span property="birthdate">1978-12-04</span>
6      <a property="url" href="https://nmerkle.github.io">
7        <span>Homepage</span>
8      </a>
9    </div>
10   </body>
11 </html>
```

The resulting RDF graph representation of the previous RDFa example is depicted in Figure 2.3. The basic principles of RDFa should have become clear. For more details please refer to the corresponding web resources[9].

[7] https://json-ld.org
[8] https://www.w3.org/TR/rdfa-core/
[9] e.g.https://rdfa.info

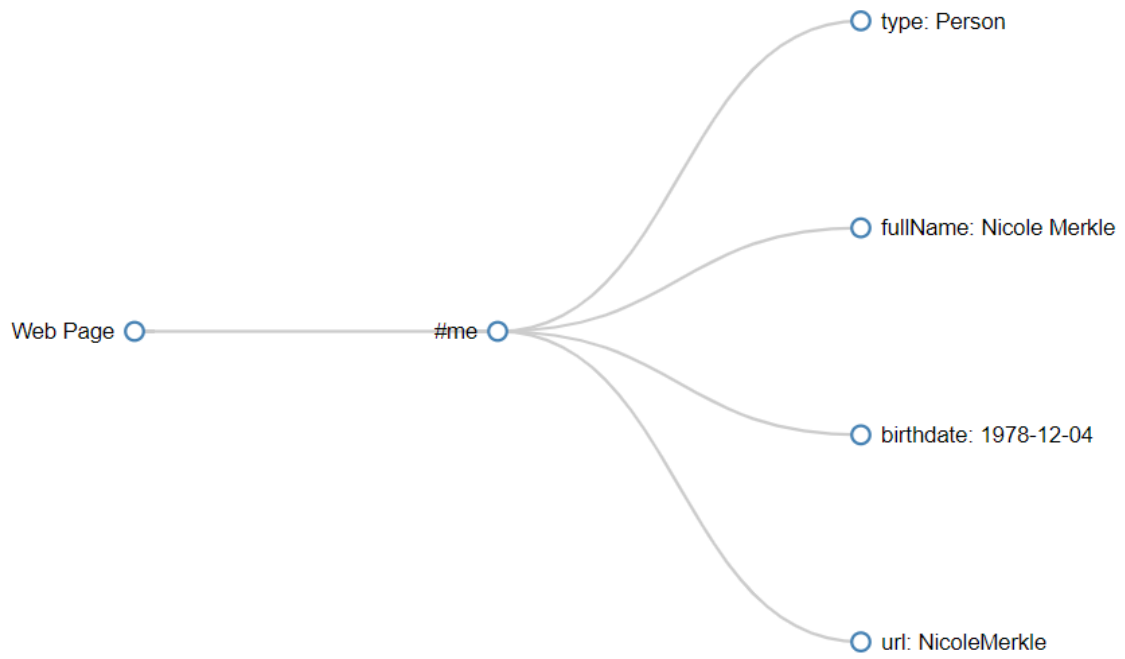


Figure 2.3.: RDFa graph representation of the resource *me*.

2.2.2. Description Logic

Before providing a basic introduction to DL, the term *ontology* has to be clarified. According to Gruber, an ontology is an *"explicit, formal specification of a shared conceptualisation"* [88].

If we deconstruct Gruber's definition, it becomes evident that an ontology has to meet different criteria to be an ontology. For instance, the ontology requires to be *explicit*, which means that the meaning of all relevant concepts has to be defined within the ontology. Further, it has to be *formal* or in other words, machine-understandable so that computers are enabled to interpret it. An ontology is, according to Gruber, a *conceptualisation*, which means that it is an abstract model of relevant concepts and their relations, and last there has to be a consensus given about the appropriate ontology in order to be a *shared conceptualisation* that can be utilised by every human or computational instance.

The main building blocks of an ontology are classes (concepts), relations (roles) and individuals (entities). Classes are templates or collections for objects (individuals) that specify which attributes (properties) an individual of this class might have. Furthermore, attributes are name-value pairs and classes can be related via attributes to other classes. Moreover, constraints (e.g. disjointness of two classes, cardinalities between attribute and class, etc.) can be defined on the attribute relations. In this way, assertions and axioms can be build in order to enable reasoning.

Considering the building blocks of an ontology, we have to distinguish *terminological (T-Box), assertional (A-Box)* and *role-centric (R-Box)* knowledge. The terminological knowl-

edge describes how ontology concepts are related to each other and which constraints apply on these relations. The assertional knowledge represents the individuals (concrete and real objects) of an ontology that implement the terminological knowledge, while the role-centric knowledge determines properties and their constraints.

The main objective of ontologies is to provide knowledge by drawing inferences, whereby implicit encoded knowledge is made explicit by applying basic inference types. The following listing introduces briefly the appropriate inference types:

- Subsumption – Inference about whether a class C is subclass of class D, i.e. $C \sqsubseteq D$.
- Class equivalence – Inference about whether class C is equivalent to class D, i.e. $C \equiv D$.
- Class disjointness – Inference about whether a class C and a class D are disjoint/disjunctive, i.e. $C \sqcap D \sqsubseteq \bot$.
- Global (in)consistency of a knowledge base (KB) – Inference about whether a KB is globally (in)consistent by showing if $KB \models \bot$.
- Class (in)consistency – Inference about whether a class is (in)consistent, if $C \sqsubseteq \bot$ is not a logical consequence of the given knowledge base.
- Instance checking – Inference about whether individual x is instance of class C, i.e. C(x).
- Instance retrieval – Inference of all individuals x of class C, i.e. $(\forall x)$ C(x).

After, it was outlined what an ontology is and what basic inference types are, the characteristics of DL will be discussed now.

DL is a family of formal languages that base on logical formalisms in order to represent knowledge. The building blocks of DL models consist of *concepts, roles, individuals* and their *relationships* to each other. Different ontology *constructors* (e.g. transitivity, inversity, number restrictions, qualified number restrictions, etc.) define the expressibility of DLs. For instance, OWL2DL is an implementation of DLs that supports the following constructors *SROIQ(D)*[10]. The OWL2DL constructor indicates that OWL2DL supports the attribute language with complement *(ALC)*, the transitivity of roles *(S)*, role constructors *(R)*, nominals *(O)*, inverse roles *(I)*, qualified number restrictions *(Q)* and data types *(D)*.

Furthermore, DLs provide logical operators such as, universal $\forall$ and existential $\exists$ quantifiers, conjunction $\sqcap$, disjunction $\sqcup$ and negation $\neg$, universal concept $\top$ and bottom concept $\bot$. By means of these logical operators it is possible to define logical axioms that represent *class inclusion, class equivalence, strict binding* and *open binding* regarding roles.

[10] `https://www.w3.org/TR/2004/REC-owl-guide-20040210/#OWL_DL_tag`

Axiom 2.1 depicts an example for class inclusion. According to axiom 2.1, a dolphin is subclass of a mammal.

$$\text{Dolphin} \sqsubseteq \text{Mammal} \tag{2.1}$$

Axiom 2.2 illustrates a class equivalence in which classes are equivalent to concepts and vice versa.

$$\text{Concept} \equiv \text{Class} \tag{2.2}$$

Also more complex class relations are possible. Axiom 2.3 depicts an example for complex class relations. For instance, a *Mushroom* is subclass of a class that either is of type *EatableFood* and *Tasty* or *Toxic* and the negation of *EatableFood*.

$$\text{Mushroom} \sqsubseteq (\text{EatableFood} \sqcap \text{Tasty}) \sqcup (\text{Toxic} \sqcap \neg\text{EatableFood}) \tag{2.3}$$

In DL exist different binding types (strict and open) for roles. The task of a binding type is to restrict by quantifiers the domain and range of a role. *Strict binding* in DL is achieved by the universal quantifier.

Axiom 2.4 shows an example that specifies that all *Car* concepts need to have a component that is of type *Wheel*.

$$\text{Car} \sqsubseteq \forall\text{hasComponent.Wheel} \tag{2.4}$$

Open binding in DL is defined by the existential quantifier. Axiom 2.5 depicts an example in that every *Paper* should have at least one *Author* that is of type *Person*.

$$\text{Paper} \sqsubseteq \exists\text{hasAuthor.Person} \tag{2.5}$$

Number restrictions allow defining the cardinality of roles. Axiom 2.6 determines that a *hasValidAge* role should have only integer values greater than or equal eighteen.

$$18 \geq \text{hasValidAge} \tag{2.6}$$

It is also possible defining qualified number restrictions (cardinalities) for roles. The difference to the previously discussed number restrictions is that the range of the role is known and the number restriction applies on the given range concept. Axiom 2.7 defines that the role *hasWheel* requires to have greater than or equal four *Wheels*.

$$4 \leq \text{hasWheel.Wheel} \tag{2.7}$$

Concrete domain restrictions stipulate a concrete data type and its value range. Axiom 2.8 illustrates that the role *hasIncome* is a numeric value and that all values of this role have to be greater than or equal three thousand five hundred.

$$\text{hasIncome.}(\geq 3500) \tag{2.8}$$

Classes can be defined by enumerations of nominals. For instance, a class of ball games could be defined by the given nominals in axiom 2.9.

$$\{\text{Soccer}, \text{Basketball}, \text{Handball}, \text{Volleyball}\} \tag{2.9}$$

Inverse roles are stipulated as in axiom 2.10. The inverse of the role *isParent* is the role *isChild*.

$$\text{isParent}^{-} \equiv \text{isChild} \tag{2.10}$$

The transitivity of roles can be expressed as illustrated in axiom 2.11.

$$\text{isPartOf} \sqsubseteq +\text{isPartOf} \tag{2.11}$$

Role compositions–also known as property chains–can be as well defined in DL. The concatenation of roles (properties) establishes a new role. Axiom 2.12 depicts an example, expressing that the sister of a father is an aunt.

$$\text{hasFather} \circ \text{hasSister} \sqsubseteq \text{hasAunt} \tag{2.12}$$

It is important to know that two assumptions are made in DL. On the one hand the *no unique name assumption (UNA)* and on the other hand the *open world assumption (OWA)*. The *no unique name assumption* stipulates that an individual can have more than one name or ID. This means that it is never assured that different named individuals are not the same one.

The *open world assumption* defines that every concept and relation is possible until it is restricted by the DL. This implies that incomplete information and uncertainty about the world state are allowed and that unknown facts about the world are not false but undefined within the knowledge base that contains the T-, A- and R-Box.

This circumstance can have advantages but as well disadvantages depending on the field of application. On the one hand, it may be required that uncertain facts about the world are allowed, since the world and its states are not always transparent and asserted. Open world ontologies can be arbitrarily extended and reused depending on their utilisation. On the other hand, domain applications may require well-defined states and certainty in order to ensure an intended behaviour of the application. Therefore, restrictions are required and since in a DL every restriction has to be specified explicitly, this can be a drawback, which means that every possible circumstance requires to be considered and restricted by the domain experts who constitute the ontology.

In contrast in the *closed world assumption (CWA)*, every relevant knowledge has to be specified and everything that is not true is in turn false. Therefore, there is no undefined knowledge given in the knowledge base. This has the benefit, that it allows the validation of facts.

2.2.3. SPARQL

SPARQL Protocol and Query Language (SPARQL) is as the name indicates, a protocol layer and a query language for RDF graphs. It allows graph traversal, so that linked graphs entirely can be discovered. The protocol layer of SPARQL transmits SPARQL queries via HTTP, while the output format specification of SPARQL is XML. SPARQL provides similar to other query languages, different query (CRUD[11]) operations, such as exploring graphs by SELECT and ASK queries, transforming graphs by CONSTRUCT queries from one vocabulary to another, creating and updating new graphs and performing federated queries over several SPARQL endpoints. SPARQL queries consist of the query statement that depends on the query type and a *WHERE* clause that contains RDF Turtle triples, representing graph patterns. These graph patterns allow to filter query results that have to match to the given graph pattern.

Listing 2.7 depicts a SELECT query with appropriate variables that are represented by variable names with a leading question mark (line 1). The results are assigned to the appropriate variables and represented within a table containing for each variable a column.

Listing 2.7: SPARQL query for selecting a Pizza.

```
1  SELECT ?pizza_name ?price
2  FROM <http://pizza-house.com>
3  WHERE {
4    ?pizza rdf:type :Pizza.
5    ?pizza :hasName ?pizza_name.
6    ?pizza :hasIngredient :Olive.
7    ?pizza :hasIngredient :Salami.
8    {?pizza :hasIngredient :Artichoke.}
9    UNION
10   {?pizza :hasIngredient :Anchovies.}
11   ?pizza :hasIngredient :Mozzarella.
12   OPTIONAL {
13     ?pizza :hasIngredient :Mushroom.
14     }
15   ?price :hasValueInEuro ?value.
16   FILTER(?value < 15)
17 }
18 ORDER BY ASC(?pizza_name)
19 LIMIT 10
20 OFFSET 0
```

In this query example, the names of pizza instances are requested that match the ingredients listed in the *WHERE* clause (line 3-13). The filtering triples within the WHERE clause are conjunctively connected. Given that disjunctive queries have to be build, the *UNION* term is required (line 9). In the given example, either the requested pizzas need to have arti-

[11] Create, Update, Delete

chokes (line 8) or anchovies (line 10), while all other triples have to match the graph pattern.

Furthermore, according to the defined *OPTIONAL* (line 12-14) and *FILTER* (line 16) function, the ingredient mushroom is optional and the price of the appropriate pizza has to be less than fifteen Euro. The *OPTIONAL* statement allows to define a triple pattern as optional condition, while the *FILTER* statement allows to specify as constraints, logical and mathematical operators (e.g <, >, <=, >=, =, !=, +, -, *, /, &&, ||, etc.) and apply general SPARQL functions (e.g. LANG, STR, IN, NOT IN, isLiteral, isBlank, etc.) to filter query results. The *FROM* statement in line 2 specifies the graph to be queried. After the *WHERE* clause, further instructions can be added, such as *ORDER BY, LIMIT, OFFSET, etc.* (line 18-20). For instance, the *ORDER By* instruction defines by which variable, the results have to be ordered. The results can be ordered either ascending (ASC) or descending (DESC). The *LIMIT* statement specifies how many results are requested, while the *OFFSET* statement indicates from which position on the results should be displayed.

In this way arbitrary complex requests (also with sub-queries) can be performed.

As already mentioned, among the SELECT query, there are also other query types.

An *ASK* query for instance, indicates if at least one result regarding the queried pattern is existing in the queried RDF graph. The response of an ASK query is always either *true* for *result exists* or *false* for *a result does not exist.*

A *DESCRIBE* query is similar to a SELECT query and provides as result a RDF graph representation with data about appropriate resources that match the given graph pattern.

A *CONSTRUCT* query allows to create new RDF graphs by existing graphs. In this way, existing graphs can be transformed into new graphs of other vocabularies. In the presented work, CONSTRUCT queries are utilised in order to deduce new facts from given graph patterns. The graph patterns represent the precondition of a rule, while the constructed triple(s) represents the implication of the rule. In the approach section it is described in detail, how CONSTRUCT queries are used as rules.

SPARQL provides many more features that are not covered within the scope of this work. For more details, the reader is referred to online sources.

2.2.4. Semantic Web Rule Language (SWRL)

The *Semantic Web Rule Language (SWRL)* is a W3C submission (no standard) of 2004. It combines parts of OWL and RuleML/DATALOG[12]. SWRL exists in three different syntax variants, one concrete XML version in OWL and RDF and an abstract syntax version in

[12] Programming language for databases

RDF. SWRL rules consist of an antecedent (body) and a consequent (head) that is the implication of the antecedent. Axiom 2.13 depicts the general structure of SWRL rules.

$$\text{Antecedent} \longrightarrow \text{Consequent} \tag{2.13}$$

The elements of a SWRL rule are conjunctively connected assertions (atoms). These assertions can be built by OWL individuals, variables and concrete OWL domain elements. The following (unary or binary) constructs are possible:

- C(x) – x is element of class C.
- D(x) – x is element of data type D.
- R(x, y) – element x is in relation R with element y.
- D(x, y) – element x is in relation with data type value y.
- builtIn(x, y_1, ..., y_n) – built in properties with an arbitrary number of elements.
- sameAs(x, y) – element x is same as element y.
- differentFrom(x, y) – element x is different from element y.

A SWRL knowledge base is usually constituted by an OWL knowledge base and a finite set of rules. SWRL utilised together with OWL is undecidable[13], though it useful for different applications (e.g. rule-based agents), given that the number of possible inferences is restricted. However, it is also possible to define decidable rules in OWL, but this will not be covered in this work, since rules in OWL were not applied.

2.2.5. Notation3 (N3)

Notation3 (N3) is a formal language that was developed by Tim Berners-Lee and Dan Connolly for specifying RDF data. N3 goes beyond defining RDF, since it also allows to specify rules, formulae, functional predicates and variables. The Turtle format is a subset of N3. Therefore, triples in Turtle are expressed the same way as in N3. For rule definitions, existential (*@forSome*) and universal (*@forAll*) quantifiers as well as implications (*log:implies, =>*) are supported. Since in this thesis N3 is proposed for defining and sharing trained policies as rules, Listing 2.8 shows an example for N3 rules, variable use and formulae statements. Formulae statements are statements about statements (reification).

In line 2 of the example Listing 2.8, an existential quantifier is defined for the variables *x* and *y*. The appropriate entailment stipulates that x supervises y and since y is a student, it can be inferred that x in all cases might be a person of type supervisor (line 3-4).

If no quantifier is given, then variables that begin with a question mark are universally quantified. This means that an implicit quantification is assumed. The same is valid for

[13] `https://www.w3.org/Submission/SWRL/`

blank node variables (line 9), since blank node variables are existentially quantified and therefore an explicit quantifier is not required. In line 9, an example is given about the use of blank node variables. *Tom* has a pet that is a *gold fish*. In line 7, the individual *Sherlock* infers the formula (statement) that the individual *Gardner* has killed the individual *Alice*.

Listing 2.8: N3 rule example.

```
1  # Entailment with quantifiers and variables.
2  @forAll ?x, ?y
3    {?x :supervises ?y.}
4    {?y a :Student.}
5    => {?x a :Supervisor.}
6    # Statement about formula
7    :Sherlock :infers {:Gardner :killed :Alice.}.
8    # Use of variable _:x
9    :Tom :hasPet _:x.
10   _:x a :GoldFish.
```

It is important to note that the formal semantics of N3 is not yet defined and the different implementations of N3 may differ. The W3C team submission[14] makes only recommendations about the intended use of N3. The usual reasoners that are implemented and used for N3 are *EYE*[15] and *Cwm*[16]. For additional information regarding N3, please refer to the mentioned web resources.

2.2.6. The Internet of Things (IoT)

Agents using the proposed agent framework deal with *Internet of Things (IoT)* systems and corresponding tasks. According to the *International Telecommunication Union's* definition the Internet of Things is: *"A global infrastructure for the information society, enabling advanced services by interconnecting (physical and virtual) things based on existing and evolving interoperable information and communication technologies." [2]*, and the *Internet Society* states that the *"Internet of Things generally refers to scenarios where network connectivity and computing capability extends to objects, sensors and everyday items not normally considered computers, allowing these devices to generate, exchange and consume data with minimal human intervention."* [233]. Considering these two definitions and the goals of the proposed agent framework, it can be stated that software agents can also be considered as *"interconnected virtual things"* that *"enable advanced services through evolving interoperable information and communication technologies"* [2] while they *"consume and exchange data with minimal human intervention"*. Software agents of the framework perceive states in the environment via connected sensors but they make decisions without considering the specificity of hardware devices. In addition, the agents of the framework use *"data-accessing and data-processing services available on the internet." [174]*. These services and

[14] https://www.w3.org/TeamSubmission/n3/
[15] http://eulersharp.sourceforge.net/
[16] https://www.w3.org/2000/10/swap/doc/cwm

APIs are provided by the agent framework when training policies and sharing and reusing task entity descriptions and policies. The agent framework uses e.g. log and time series data to derive, combine and adapt decision rules and task policies while it mediates the bi-directional communication between software agents and corresponding environments. This implies that sensors and agents send in an event-driven manner information via the agent framework in order to perceive and cause environmental changes. In this sense, it can be said that the agent framework serves for software agents that act in IoT-based systems and solve heterogeneous IoT tasks.

2.2.7. The Web of Things

The Web of Things (WoT) is candidate for a W3C recommendation[17] and serves as an abstraction layer for IoT infrastructures in order to provide interoperability in different areas of application and for different use cases (e.g. smart cities, health care, industry, smart buildings, gas and oil, insurance, etc.)

The basic concept of the WoT is the abstract *Thing Description (TD)*[18] that encapsulates the semantic and virtual representation of physical and virtual *Things* in the appropriate IoT environments. *Things* are in this context relevant objects in the environment, that can be *shared, retrieved, accessed* and *composed*. Therefore, WoT implementations require to provide for every of these functionalities a layer that addresses and serves for the appropriate management of *Things* and their representation.

The TD comprises of some main building blocks, such as *meta data (property affordance)* about the *Thing*, *interaction affordance* (e.g. actions, events), *data exchange schemas* for interpretability and *web links* that associate a *Thing* to other *Things* in the environment.

In this work, the concepts of the TD were adopted for the interaction of agents, platforms and IoT devices. The considered *Things* are usually responsible for providing their TD that can be looked up by other WoT participants (Things) in order to use and communicate with the appropriate *Thing*. The specification of TDs is made by JSON or JSON-LD. Listing 2.9 was adopted from the W3C page[19] and depicts a TD of a Lamp device.

Listing 2.9: WoT description as JSON-LD.

```
1 {
2    "@context": "https://www.w3.org/2019/wot/td/v1",
3    "id": "urn:dev:ops:32473-WoTLamp-1234",
4    "title": "MyLampThing",
5    "properties": {
6       "status" : {
7          "type": "string",
```

[17] https://www.w3.org/WoT/
[18] https://www.w3.org/TR/wot-thing-description/
[19] https://www.w3.org/TR/wot-thing-description/\#simple-thing-description-sample

```
8           "forms": [{"href": "https://example.com/status"}]
9        }
10    },
11    "actions": {
12       "toggle" : {
13          "forms": [{"href": "https://example.com/toggle"}]
14       }
15    },
16    "events":{
17       "overheating":{
18          "data": {"type": "string"},
19          "forms": [{
20             "href": "https://example.com/oh",
21             "subprotocol": "longpoll"
22          }]
23       }
24    }
25 }
```

In the presented JSON-LD example, a lamp device is described by a TD. First, the context of the lamp is defined. The @context reference provides the schema definition, namely the context in which the lamp is described (line 2). Then, in line 3, the @id of the lamp is specified by a URN. The lamp supports the action *toggle* that provides a *forms* reference link for the mentioned action (line 13). Further, the given lamp may trigger the event (*OverHeating*), whose value type is a string data type. Events have as well a *forms* specification that references a link to the appropriate event form. Moreover, a sub-protocol can be defined that stipulates the kind of request protocol (i.e. *longpoll*). There are already different WoT implementations[20] that provide the infrastructure and adhere to the *Thing* representation. However, for the given work at hand, an own WoT implementation was developed, since the WoT was emerging newly and there were no mature implementations that could have been reused.

[20] https://hacks.mozilla.org/2017/06/building-the-web-of-things/, https://github.com/thingweb/node-wot

2.3. Machine Learning

In the previous sections, agent-based systems and Semantic Web technologies were introduced and discussed. Now the focus is on machine learning (ML) that enables agents to learn policies and adapt themselves to new situations. In this section, the adopted information about ML is based on the following literature [25, 169, 65, 242, 181, 213]. In the agent framework presented, agents apply reinforcement learning (RL) to learn to solve tasks.

However, before we dive into RL, some definitions regarding the challenges and nature of ML have to be clarified. In the following sections, this work outlines topics and terms such as *(un-)supervised learning, reinforcement learning, feature engineering, ML model evaluation, etc.* in order to give a basic introduction into the employed ML terminology.

2.3.1. Learning

According to [213], *learning* in the context of ML means learning a function by means of inputs and appropriate outputs in order to make predictions by a learned (non-)linear function for new unseen input data. Learning is important because not every situation that could occur, can be considered beforehand at design time of a ML task. For this reason, agents require to learn how to react to possible unconsidered situations. Moreover, it is not possible to predict beforehand over time upcoming changes in the agent environment and the last reason why *learning* is important and indispensable for agents, is that some solutions for given tasks are not programmable by fix procedures and functions, especially if the previously mentioned conditions (e.g. dynamically changing situation and environment) are given. The objective of learning is to obtain a representative mathematical model for a given problem. According to Mitchell [169], *learning* is defined as:

"A computer program is said to learn from *experience E* with respect to some class of *tasks T* and *performance measure P*, if its performance at tasks in T, as measured by P, improves with *experience E*."

In other words, an agent learns by its made experience regarding a certain class of tasks and by evaluating a performance measure, while the learning process is observable by the performance improvement, depending on gathered experience. This implies that three different components (*Class of Tasks, Experience, Performance*) are required to characterise a (machine) learning process and describe a learning agent.

If we map Mitchell's definition of *learning* to agents, the class of tasks can be considered as the agent's *problem domain* that has to be solved by the agent. The agent's experience is built by representative *training samples* (observations) and an agent *function* represents the utility (i.e. *performance measure*) of conducted actions with respect to observed states.

The agent's performance is usually measured by an *error* or *loss function* (e.g. mean squared error) that indicates the deviation (error rate) from the training sample's model representation. The fitting of the model function is conducted by different algorithms (e.g.

stochastic gradient descent, etc.).

There are many challenges in ML (e.g. data quantity and quality, feature selection and pre-processing, etc.), which are discussed in the corresponding sections.

2.3.2. Supervised Learning

The training of ML algorithms depends in particular on the datasets that are used for building a representative model. Annotated datasets that provide the corresponding outputs to given inputs contribute to *supervised learning*. The agent gets feedback by applying the input values (X-vector) to the model function and compares by the loss function, the results of its model function with the given outputs (Y-vector) of the annotated samples. At the same time, the error function shows how much the model has to be adjusted until it produces the desired results.

$$X = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \quad Y = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix}$$

The success of supervised learning depends highly on the *quantity, quality, significance* and *representativeness* of annotated datasets. The different prediction classes have to be separable by their feature representation. The better the separation of samples by their features is possible, the more are the appropriate models and predictions reliable.

Most of the algorithms require a high number of qualitative training samples in order to cover and reproduce all data characteristics inherent to the model. *Noise* in data by irrelevant features and missing or incorrect values can be a problem and lead to non-representative models and worse prediction results. Moreover, datasets can be *imbalanced* which means that not all of the predictable classes are sufficiently represented in the training samples. Given that a class is more representative than other classes, this may contribute to good predictions for this one class but to uncertain and incorrect predictions regarding the under-represented classes.

Feature engineering, cleaning and pre-processing of data samples help to overcome the above problems by converting the data into machine-processable formats. Figure 2.4 illustrates a ML pipeline that is usually processed during the training of a model. First, the (annotated) preprocessed datasets serve for the learning algorithm as input, then a hypothesis (i.e. model function) is trained and then this hypothesis is tested by test data in order recognise the error rate and to adjust it according to the error rate. The trained hypothesis is later employed for making predictions.

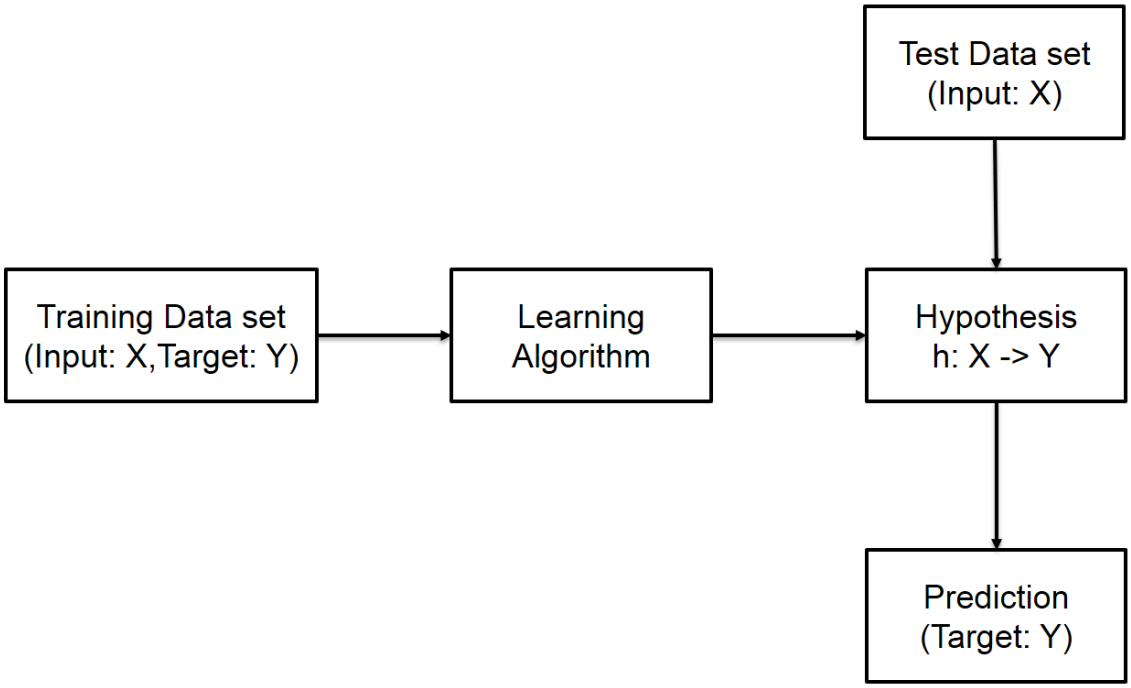


Figure 2.4.: Supervised machine learning process steps

2.3.3. Unsupervised Learning

In cases in which the training samples only consist of input but no output data, agents have to deal with an *unsupervised learning* problem. This means that the agent gets no supervision about how output results according to the given features have to look like.

$$X = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

The main objective in such cases is to find patterns (e.g. clusters) in the data, draw conclusions and get insights by these patterns. The focus is more in *exploring, interpreting* and *describing* the data. *Distance* respectively *similarity measures* (e.g. Euclidean distance, Manhattan distance, Cosine distance, Minkowski distance, Jaccard similarity, etc.) and clustering algorithms (e.g. K-Means, DBSCAN, Expectation Maximisation, etc.) as well as dimensionality reduction (e.g. Principal Component Analysis (PCA), Latent Semantic Analysis (LSA)) algorithms are applied. This kind of problems are usually more often given, since the annotation of data is costly and means some effort. Unsupervised learning, however, is more difficult to manage than supervised learning because it is not clear what the outcome will be and it depends heavily on the interpretation of the data scientist as to how conclusions are drawn from the recognised patterns. In unsupervised learning the objective is to receive new, unknown insights and knowledge.

2.3.4. Over- and Under-fitting

A hypothesis (model function) can perfectly fit the given training data by capturing the underlying relationship between independent and dependent variables. The trained hypothesis then perfectly represents the training dataset, but does not generalise well to new, unseen data samples. Thus, an over-fitted model implies high variability in predictive performance as measured by a specific loss function (e.g. mean squared error, log-loss) across different datasets, i.e. across training data and test data. Reasons for high variance in trained models include training on a small sample size or a large number of features in the data and hence parameters to train for.

Moreover, such a hypothesis is sensitive to noise. In the above cases, the model function has a *high variance*. Over-fitting or variance is a problem in ML that requires various heuristics (e.g. regularisation, bagging, boosting) that avoid over-fitting the model function.

Eq. 2.14 (ridge regression) and Eq. 2.15 (lasso regression) represent two regularisation methods where R is the regularised loss, *loss* is a model-dependent loss function, λ is the regularisation parameter that ranges between 0 and ∞, and $w_1 \dots w_n$ are the estimated model parameters, i.e. weights. The only difference between the *Lasso* and *Ridge* regressions is that the former computes the *absolute* sum of the model parameters, whereas the latter squares the sum of the model parameters. The consequence of this slight difference is that in lasso regression, unimportant model parameters that do not contribute to the prediction can be eliminated by setting the corresponding parameters to zero, whereas in ridge regression their value approaches zero but never becomes zero.

$$\mathrm{R} = \mathrm{Loss} + \lambda(\mathrm{w}_1 + \mathrm{w}_2 + \ldots + \mathrm{w}_\mathrm{n})^2 \tag{2.14}$$

$$\mathrm{R} = \mathrm{Loss} + \lambda \mid \mathrm{w}_1 + \mathrm{w}_2 + \ldots + \mathrm{w}_\mathrm{n} \mid \tag{2.15}$$

The opposite problem of over-fitting a hypothesis is under-fitting or biasing it. Thus, it can be stated that there is a trade-off between variance and bias, and the difficult task of an estimator is to find a balance between variance and bias.

When bias is present, the hypothesis does not fit the training data and yields poor or random results for each prediction. This is usually the case when the dataset is not representative of the prediction task or when the quality and quantity of the data is insufficient. Another reason for bias could be that the ML method only captures linear models, while the data is based on a non-linear and complex model. Neither bias nor variance is desirable and can be avoided by different approaches, as mentioned above.

2.3.5. Classification versus Regression

ML problems can be distinguished by whether they solve a *classification* or a *regression* problem. Regression algorithms predict numeric value outputs, while classification algo-

rithms predict distinct classes or categories of made observations. Classification problems can be separated into *binary* (two classes) or *multi-class* (many classes) classification depending on the number of classes that can be predicted.

For instance, predicting the price of a house by certain house features or making a weather forecast regarding upcoming temperatures are regression problems, since the appropriate ML algorithms predict numeric values. Examples for classification tasks are for instance image recognition, spam filtering and many more.

2.3.6. Feature Engineering and Data Preprocessing

Before training a hypothesis, a ML engineer or data scientist requires to explore and consider which data features may be of interest for the given problem. There may be cases in which features are redundant or irrelevant and then it is required that the redundancy and irrelevant features are eliminated through feature engineering. Moreover, incorrect or missing values in the data lead to poor results, and raw data can usually not directly be utilised as training data because of many different reasons such as e.g. noisiness of the data, missing or incorrect values, no machine-processable and uniform format, no numerical representation, etc. Additionally, the *curse of feature dimensionality* can be a problem, since a high-dimensional feature space in the datasets can lead to sparse vectors and insignificant data samples. This in turn, contributes to a diminished performance of the learning algorithm.

In order to overcome these problems and challenges, feature selection and dimensionality reduction (e.g. Principal Component Analyse (PCA), etc.) techniques are applied.

Assuming now that the test data is preprocessed, we have to consider how the test data can be effectively utilised for learning. Usually, the test data is divided into different partitions also called bins, such as *training set, test set, validation set*[21]. The ratio of this splitting is individual but it makes sense to take a much higher ratio of samples for training than for the other both partitions. In order to ensure the equal distribution of prediction classes in every data partition, approaches such as *sampling, (k-fold, leave-one-out) cross validation* [213] are applied. For instance, in *k-fold cross validation* the dataset is partitioned in k-partitions, whereby iteratively k-times, k-1 equal sized bins are taken as training data while the one selected bin is taken as validation data. Then the training results are averaged over the k-bins. In this way, every class has the chance to be in the training dataset.

Feature Engineering is an important aspect of ML. The right selection of features and data cleaning are crucial for the success of ML algorithms.

[21] The validation data serves for evaluating the model performance during the training, while the test data serves for evaluating the performance of the finally trained algorithm.

2.3.7. Evaluation of ML Models

After or during a ML model is trained, the prediction performance and reliability of the model can be evaluated by different metrics, depending on the ML algorithm (e.g. supervised learning, reinforcement learning), such as *accuracy, precision, recall, F1, Receiving Operator Characteristics (ROC) and Area under the ROC curve (AUC), confusion matrix, cumulative rewards, etc.* In this section, we discuss common metrics and evaluation means for supervised ML and in particular for classification problems as well as for reinforcement learning.

For evaluation purposes usually a *ground-truth* is required. The ground-truth consists of datasets whose prediction outcomes are representative correct and already known. For instance in a binary classification problem, a ground-truth is provided by annotated data samples that represent the two classes (positive, negative). According to the comparison of prediction results against a ground-truth, different rates can be measured, such as *true positive (TP), false positive (FP), true negative (TN)* and *false negative (FN).*

TP means that the trained classifier and the ground-truth have the same positive prediction result. FP means that the ground-truth result is negative but the hypothesis predicts it as positive. In turn, FN means that the ground-truth result is positive but the hypothesis predicts it as negative and TN means that the ground-truth result is negative and the hypothesis predicts it as well negative.

A confusion matrix (see Table 2.1) can be utilised for counting the number of correct and incorrect predicted classes. It enables intuitive insights into the performance of a trained hypothesis by comparing the results of a hypothesis against a given ground-truth. Based

		Prediction	
		positive	negative
Ground-Truth	positive	#TP	#FN
	negative	#FP	#TN

Table 2.1.: Confusion Matrix

on the previously mentioned rates, different metrics such as accuracy, precision, recall and F1 can be computed in order to show the prediction performance of a trained classifier. Equation 2.16 computes the accuracy of a classifier. The number of all correctly classified results (TP and TN) is divided by the size of the entire evaluation set. The accuracy however is no reliable metric, since it depends on the *balance* of the dataset, which means that all classes are equally represented within the evaluation set. In *imbalanced* datasets, using accuracy does not make any sense because it does not reflect how the accuracy would change if the occurrence of the under-represented classes would increase. Therefore, other metrics are more suitable for evaluating a classifier.

$$Accuracy = \frac{\#TP + \#TN}{\#TP + \#FP + \#TN + \#FN} \tag{2.16}$$

The precision of a classifier computes the ratio of TPs and the entire–by the classifier–as positive predicted data samples. This metric shows how precise the classifier was in its positive predictions. 2.17 shows the equation for computing the precision of a classifier.

$$Precision = \frac{\#TP}{\#TP + \#FP} \tag{2.17}$$

The recall of a classifier computes the ratio of TPs and all positive values that are given in the ground-truth. This metric expresses how many of the entire positive samples could be covered by the classifier. 2.18 depicts the equation for computing the recall rate.

$$Recall = \frac{\#TP}{\#TP + \#FN} \tag{2.18}$$

The F1 measure is the harmonic mean of precision and recall, so it combines both measures for providing one key number for estimating the reliability of the (binary) classifier. The maximum F1 rate that can be achieved is one, which means that the classifier has a perfect precision and recall of 100 percent. Eq. 2.19 shows the computation of the F1 rate. Precision and recall are equally weighted.

$$F1 = 2 * \frac{Precision * Recall}{Precision + Recall} \tag{2.19}$$

Among the outlined metrics, there are also metrics for determining the most accurate decision boundary threshold values of trained classifiers. For this reason, many different decision thresholds are tested for performance and their performance is plotted as individual points in a (ROC/AUC) visualisation. In other words, each point in the ROC/AUC represents the performance of a tested classification threshold. If multiple thresholds are assessed, the points form a curve representing the relationship between *true positive rate (TPR)* (see Eq. 2.20) and *false positive rate (FPR)* (see Eq. 2.21). The TPR corresponds to the recall probability, while the FPR determines the ratio of classes incorrectly predicted as positive that were labelled as negative to all negative results calculated in the ground-truth. The AUC allows comparison of different ML methods by calculating the area under the given ROC curve. The larger the AUC, the better the prediction method. Thus, when comparing two or more methods, one should select the method with the highest AUC value.

$$TPR = \frac{\#TP}{\#TP + \#FN} \tag{2.20}$$

$$FPR = \frac{\#FP}{\#FP + \#TN} \tag{2.21}$$

By the visual representation of ROC/AUC, TPR and FPR are set into relation. Figure 2.5 depicts an example ROC/AUC representing a synthetically generated performance estimate of a classified dataset. The more the curve tends towards the value one hundred in the top left corner the better is the reliability and performance of the classifier, since the TPR is then higher than the FPR. The AUC shows the ratio between TPR and FPR. The

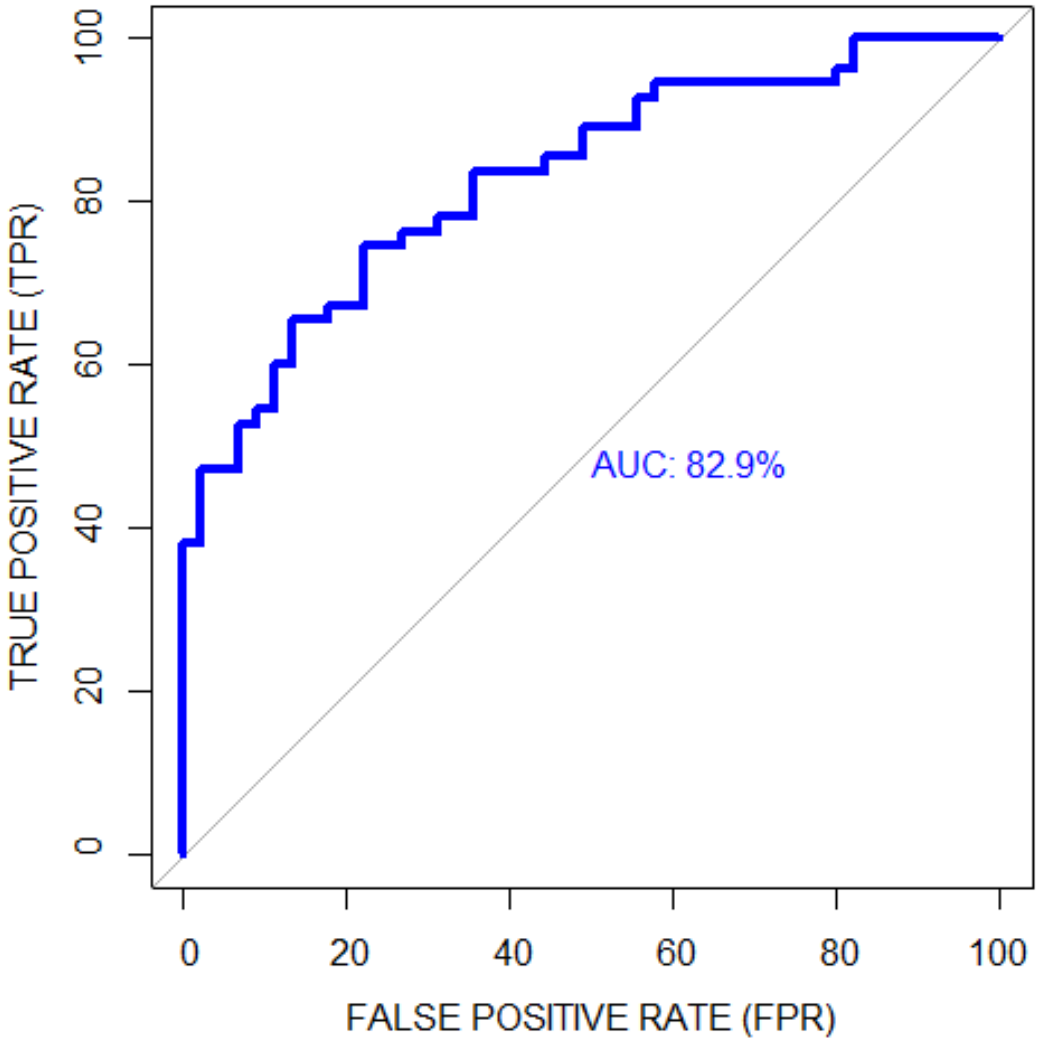


Figure 2.5.: Illustration of a ROC/AUC curve of a synthetic generated and classified dataset

higher the AUC rate, the more reliable are the made classifications. In the given case the AUC is at 82.9 % which is not bad for a classifier. However, a value towards one or one hundred is the ideal AUC for a classifier.

In reinforcement learning (RL), the evaluation of the performance of a strategy looks different than in supervised learning because a) in RL there is usually no baseline data available and b) the only feedback the RL agent receives about whether or not it is performing well is the reward value received per action execution step. Therefore, different evaluation metrics depending on the use case scenario have to be considered and applied.

According to Poole and MacWorth [198], the evaluation of RL agents is either by looking *"how good an RL agent finds policies"* or by considering the cumulative reward as a function of steps (t).

$$f(t) = \sum_{t=0}^{T} \text{Reward}_t \tag{2.22}$$

For instance, if no further exploration steps are required because the agent has time to sufficiently learn off-line final policies, then the cumulative reward function may be the metric of choice to estimate the performance of the trained policies, while an agent that has to train during deployment time, may be evaluated by how *good* and fast it finds policies. Hence, the RL agent collects and sums the reward gained per execution step. Figure 2.6 shows an example plot of a cumulative reward function. The slope of the function can be used to derive various statistics about the performance and utility of the actions performed, indicating whether the agent learns and performs desirable or useful behaviour in the course of task execution. As depicted in Figure 2.6, the slope is continuously increasing which means that the agent continuously learns a reward maximising and therefore useful behaviour.

2.3.8. Multinomial and Gaussian Naive Bayes Classifier

Agents who observe their environment can use different types of classification algorithms for decision making. The *Naive Bayes* classifier is one such classification algorithm that allows agents to compute *conditional probabilities* of actions and thus select actions depending on the currently observed state and the underlying collected data representing the agent's experience. The Bayes classifier is based on the *Bayes Theorem* (see Theorem 2.23) that was defined by Thomas Bayes, a mathematician in the 18th century. The foundation to multinomial and Gaussian naive Bayes classifiers was adopted from the books: *Pattern Classification* [65] and *Thinking Bayes* [64].

$$P(\text{action}|\text{evidence}) = \frac{P(\text{action}) \cdot P(\text{evidence}|\text{action})}{P(\text{evidence})}, \tag{2.23}$$

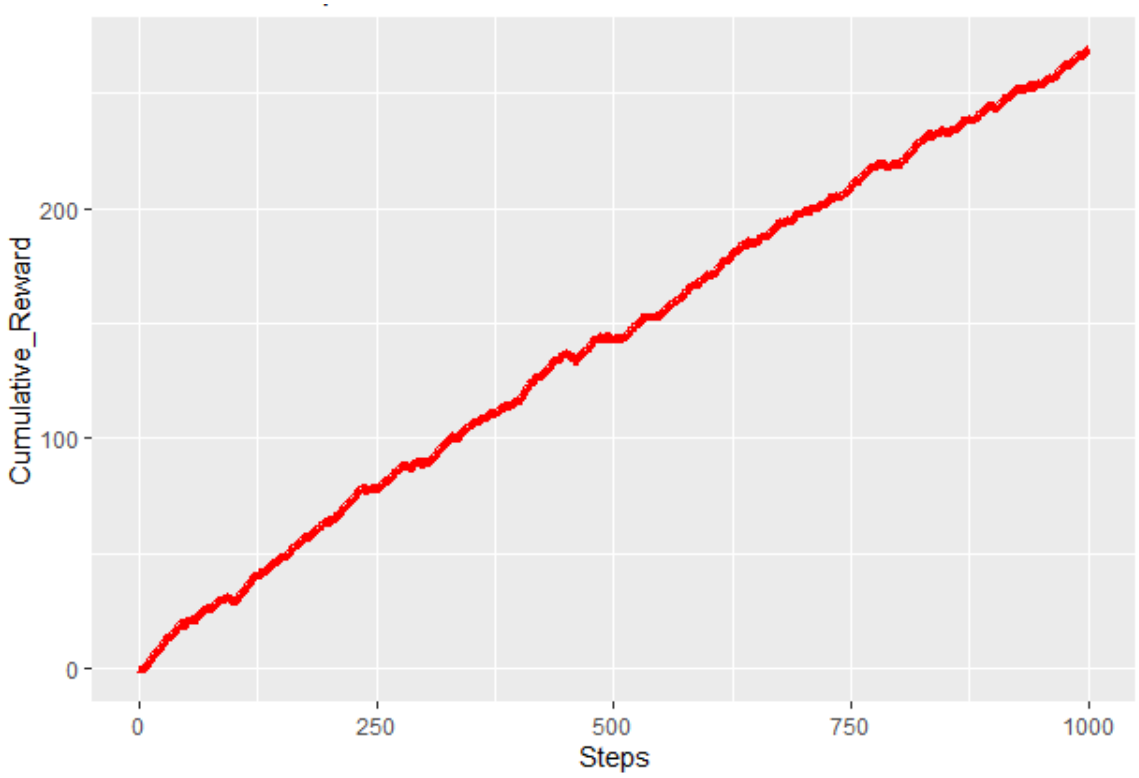


Figure 2.6.: Function plot of cumulative reward per execution step

where P(action|evidence) is the *posterior probability* of an action given an evidence, P(action) is the *prior probability* of action, P(evidence|action) is the *likelihood* of evidence and P(evidence) is the prior probability of evidences.

The mentioned theorem prescribes the calculation of conditional probabilities. It is assumed that a hypothesis H is given whose *posterior probability* is to be calculated. In the case of the agent, the hypothesis could be any action that the agent is able to perform. The conditional variables (i.e. evidences) are the observations that the agent makes before each execution step by perceiving its environment. The *likelihood* of an evidence, is defined by the probability of an evidence given a hypothesis. Thus, this means that a hypothesis and evidence co-occur in the dataset.

It is important to note that the Naive Bayes classifier is called *naive* because it assumes that all evidence is independent of each other, which is rarely the case in reality, as there are usually some dependencies between evidence variables that influence each other. Nevertheless, the Naive Bayes classifier has been shown to work very well for certain classification problems (e.g. Spam classification).

Acquired data can consist of different types of features (e.g. discrete, continuous) and thus have different probability distributions (e.g. uniform, binomial, multinomial and Gaussian). A finite set of possible feature values characterises discrete data, while continuous data

can consist of an infinite number of possible feature values. Depending on the underlying probability distribution, either the agent requires to perform a multinomial[22] Bayes classifier for discrete features distributed in a *Probability Mass Function (PMF)* or a Gaussian Bayes classifier for continuous features distributed in *Probability Density Function (PDF)*. Thus, $P(x)$ is the probability of a feature x.

Equation 2.24 defines how probabilities of multi-nomial features can be calculated.

$$P(x) = \sum_{i=1}^{N} P(C_i \cap x) = \sum_{i=1}^{N} P(x|C_i) \cdot P(C_i), \tag{2.24}$$

where x is an observed feature (e.g. door) with a certain value (e.g. open) and C_i is a predictable class in the dataset. Eq. 2.25 depicts how probabilities of Gaussian distributed features can be calculated.

$$\mathrm{P(x)} = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}, \tag{2.25}$$

where x is an observed feature value, σ is the standard deviation, σ^2 is the variance and μ is the mean of the observed feature's values.

To summarise the difference between multinomial and Gaussian Naive Bayes classifiers, it can be said that for data with continuous feature variables, the Gaussian probability calculation is required, while for discrete and thus categorical feature variables whose frequency of occurrence can be counted, the multinomial probability calculation is required. In this way agents have a statistical and evidence based method to decide their next step of action.

2.3.9. Markov Decision Process

This section lays the technical and formal foundations for understanding reinforcement learning (RL). The activities of agents are modelled and represented by Markov decision processes (MDPs) in reinforcement learning. Furthermore, strategies learned by means of value functions are expressed by policies. These terms therefore require clarification. First, MDPs are introduced, then a definition of policies and value functions is given. For the description of *finite MDPs, policies, and value functions* in this section, the definitions were adopted from [203] and [242].

According to [242], a *finite* MDP is a mathematical formalisation that allows RL agents to make sequential, goal-oriented decisions that do not only impact immediate rewards but also subsequent states and future (i.e. delayed) rewards. MDPs are defined through a tuple (S, A, R) of finite sets, where S is the finite set of process states, A is the finite set of possible actions that can be performed by an agent and R is a finite set of reward

[22] Multinomial in this context means that the frequency of multiple discrete events forms a probability distribution that can be represented by a histogram.

values. MDPs enable agents to learn policies through their interactions with their external environment. In particular, an agent has the objective to maximise through its performed actions and accumulated reward values its internally maintained *value* function. Each action $A_t \in A$ is selected and performed in discrete time steps $t = 0, 1, 2, 3...n$ based on the perceived state $S_t \in S$. After executing an action A_t, the agent obtains a numerical reward signal $R_{t+1} \in R \subset \mathbb{R}$ and a subsequent state S_{t+1}. Thus, a sequence or *trajectory* of a MDP is defined in [242] as follows: $S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3, ...$ The probability distribution p of states and rewards is determined by their preceding states and conducted actions. Eq. 2.26 formalises this fact:

$$\mathrm{p}(\mathrm{s}', \mathrm{r}|\mathrm{s}, \mathrm{a}) \doteq Pr\{S_t = s', R_t = r|S_{t-1} = s, A_{t-1} = a\} \tag{2.26}$$

Eq. 2.27 specifies the conditional probability p that considers the vocabulary respectively sets $s, s' \in S$, $r \in R$, $a \in A(s)$ and expresses the *dynamics* of MDPs:

$$p : S \times R \times S \times A \rightarrow [0, 1] \tag{2.27}$$

The introduced conditional probability also indicates the *Markov property* of a state, since only the previous state and the action performed determine the probability of the next state S_{t+1} and the corresponding reward R_{t+1}. Thus, it is not necessary to consider all previous states, but only the current state, to determine the transition probabilities of S and R.

Any state in an MDP can be a starting point state of an agent. To express how likely it is that a state will become an initial state, there is a probability $p0$ that expresses this fact.

Since the foreseen agents have to solve process oriented tasks, MDPs are predestined to serve as a process model. In particular, reinforcement learning approaches represent tasks by means of MDPs. However, these MDPs are in most of the cases unknown beforehand and have to be *explored* by the agent, especially if the solvable tasks are complex and have an infinite state space.

By means of a MDP, an agent can train policies π: S $\rightarrow$ A for every state-action pair specified within the MDP.

For instance, Fig. 2.7a illustrates a 2-D game[23] in which a robot has the task to collect red apples whereas it has to avoid green apples since they are poisoned. In addition, the robot needs to be careful not to bump into one of the 4 outer walls of the playing field and to eat red apples in time to recharge its energy or battery level. The robot can perform 7 actions: *Pick up apples*, *move* (right, left, up, down), *eat apples* and *stop* its activities when its battery level is depleted. While the nodes of the MPD represent the appropriate states (*Apple in front, Bumped into wall,...*), the edges represent the actions that lead from one state to another state. There are 3 final states (i.e. *No battery energy, Died by green apple, Collected all red apples*) that are represented through bold node edges and finish

[23] Accessible on Github at `https://github.com/nmerkle/SMAPPI-Smart-Apple-Picker-Agent`

the game whereas the state *Collected all red apples* is the goal state that the robot intends to achieve. Each state has a positive or negative reward value that indicates whether a state is desirable or not. Furthermore, for every transition (T) a probability value is given. For instance, if the robot has a wall in front of it the chances are 50 % that it will bump into the wall and the appropriate reward for that action that leads to this state is a value of -10. In the state *Bumped into wall* two different actions are possible (e.g. *Move* further against the wall, or *move* away from the wall). However, it is not clear in which order the actions have to be performed. Therefore, the agent has to explore which action sequences promise the highest expected cumulative reward.

As Figure 2.7b illustrates, the probability distribution of transitions approximately sums up for every single state to the value one. MDPs are simple models. However, tasks can be arbitrarily complex and then it is usually not possible to visualise a MDP. The agent has to discover itself by exploration and exploitation steps the state representations and relating actions with the appropriate transition probabilities. The visual representation of MDPs are only a means for developers to understand the processes of a task and gain knowledge about the problem domain.

2.3.10. Reinforcement Learning

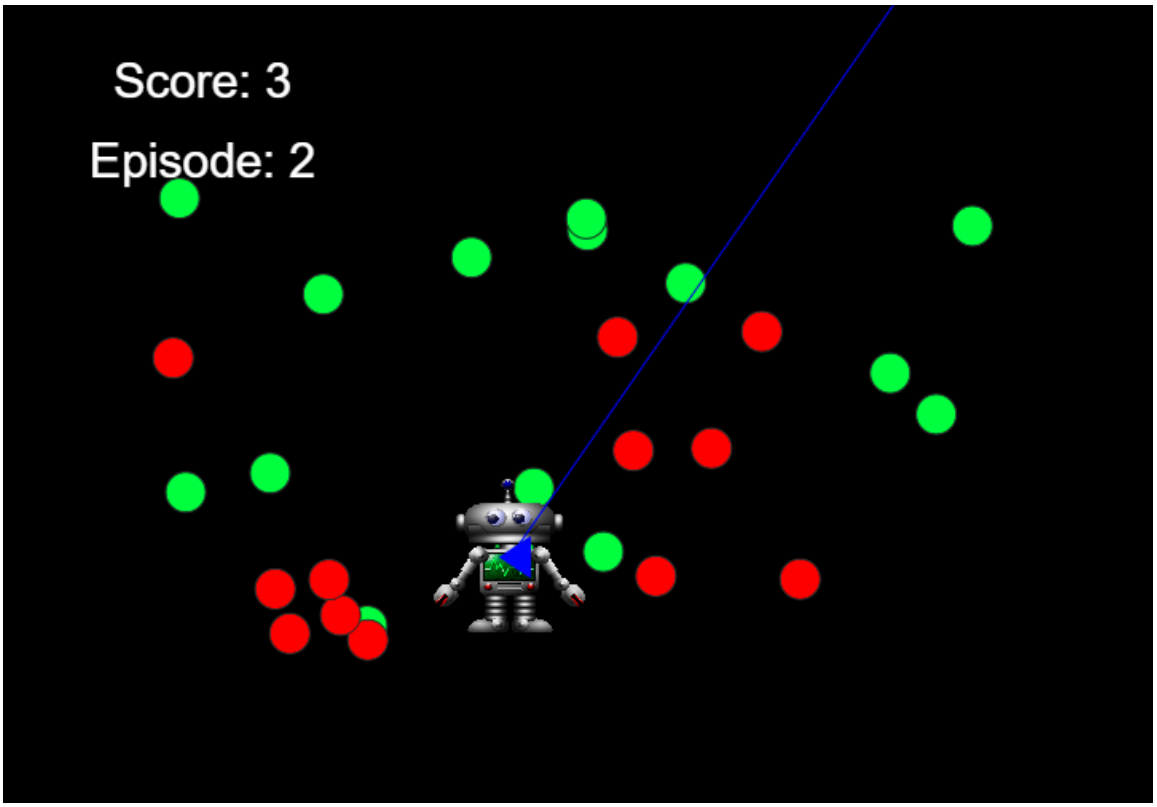
Reinforcement learning (RL) is another ML paradigm, along with supervised and unsupervised learning [242]. One assumption in reinforcement learning (RL) is that every behaviour has in the long- or short-term, a consequence and this consequence can be either positive or negative. An agent learns its strategies (policies) by interacting with its environment and is either rewarded or punished for its performed actions. The objective in RL is to imitate the way of how human beings learn from their childhood on. We humans get directly or indirectly rewarded or punished for our actions and RL exactly reproduces this way of human learning for software agents.

An agent that implements RL, performs an action and receives for the performed action by a real or simulated environment a *return* (e.g. reward or punishment) represented by a scalar number. RL processes can be considered as a cycle in that the agent perceives and interacts with its environment (see Figure 2.8). However, before an agent can compute a policy, it requires a model representation of its task environment with a terminating goal state. A RL task is usually represented by a *Markov Decision Process (MDP)* (see Sec. 2.3.9). Within a MDP model every transition probability from one state to a next state depends only on the current state of the environment and the performed action.

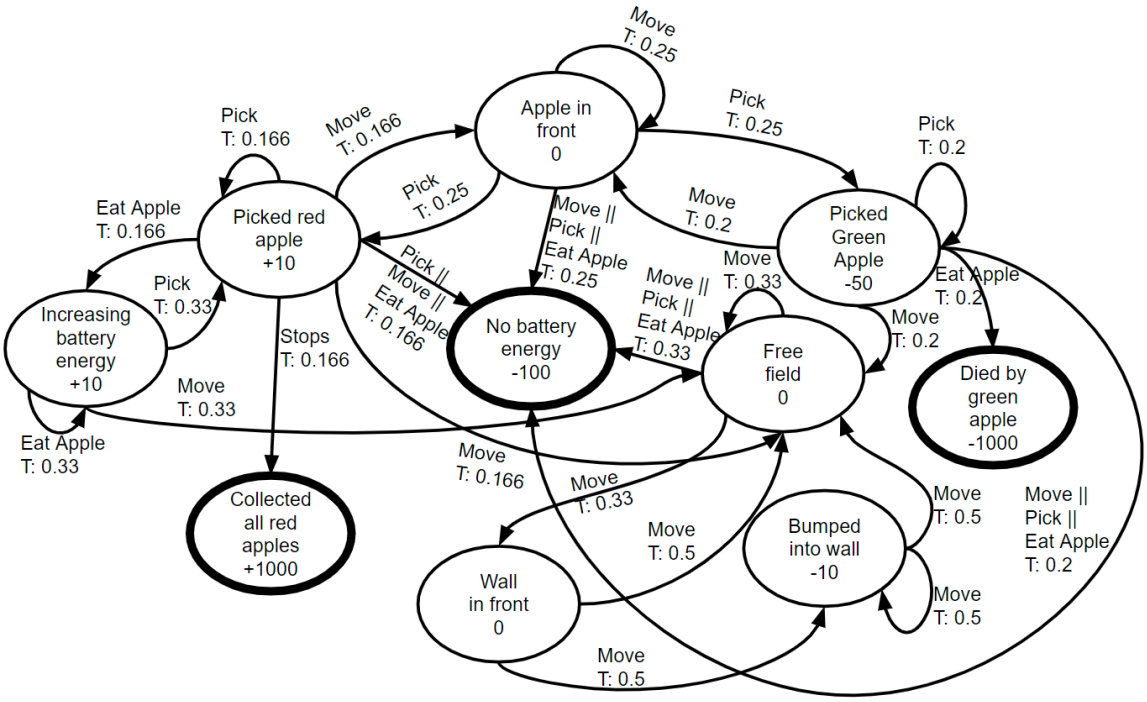
A policy π represents the probability of performing an action a in a certain state s. Equation 2.28 represents the general definition of a policy.

$$\pi(\mathrm{s}, \mathrm{a}) \tag{2.28}$$

Furthermore, a resulting optimal policy can be considered as a mapping from environmental *states* to optimal *actions*. Equation 2.29 shows an optimal policy representation.



(a) A screen-shot of the smart apple picker (SMAPPI) game.



(b) An example MDP modelling the SMAPPI game that is derived from the rules of the game.

Figure 2.7.: Screenshot of the SMAPPI game [232] and corresponding MDP created as part of this work to illustrate an example of an MDP.

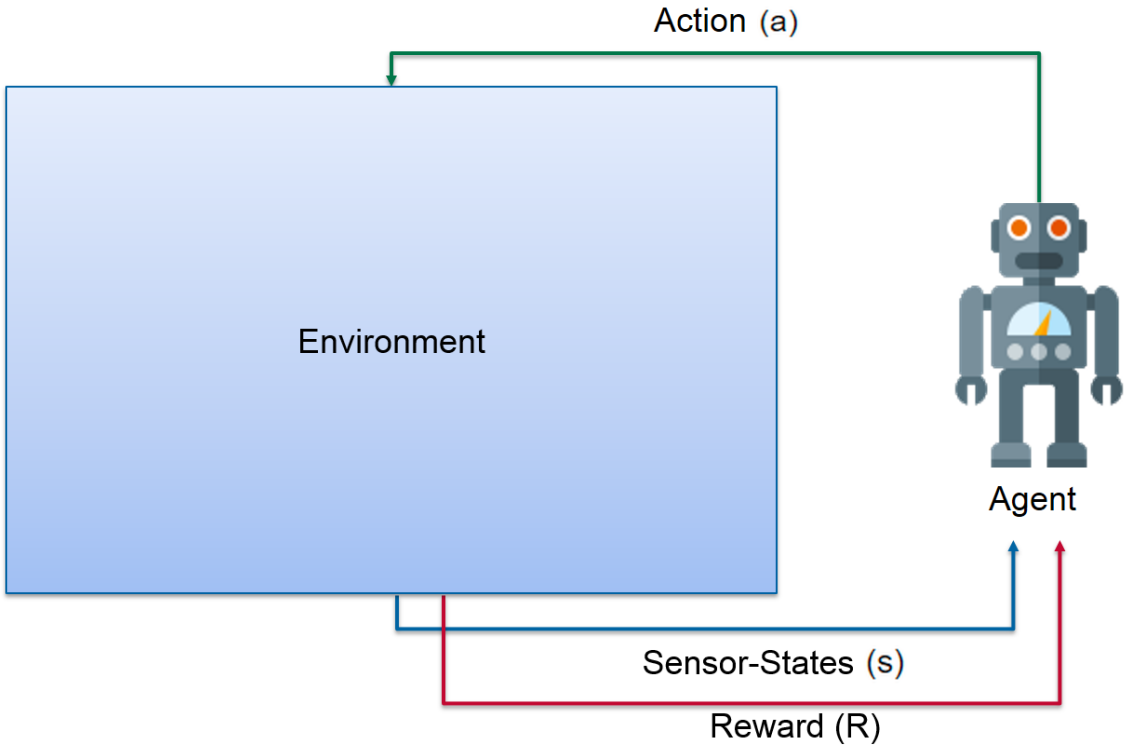


Figure 2.8.: Learning cycle in RL

For every given state s of a task, an optimal action a is computed by the agent, so that the agent knows in every possible state which action to perform. An optimal action is in this context an action that leads to the highest expected cumulative reward value. This implies that the agent needs to iteratively compute the optimal action for every state until a value function of each state-action pair is converging.

$$\pi^*(\mathrm{s}) = \mathrm{a} \tag{2.29}$$

There are two different value functions in RL, called *state value function* and *action value function*. A *value function* indicates how reward-maximising, a state or state-action pair is for the agent because it takes into account future rewards that can be expected by the agent. Equation 2.30 depicts the state value function V that computes the expected $\mathbb{E}_\pi$ reward value $\mathrm{R_t}$ of a state s for performing a certain policy π at time t.

$$\mathrm{V}^\pi(s) = \mathbb{E}_\pi[\mathrm{R_t} \mid \mathrm{s_t} = \mathrm{s}] \tag{2.30}$$

A value function requires always to be considered in relation to a policy where a policy is a probability function $\pi(a|s)$ indicating the probability $a \in A(s)$ that an action $a = A_t \in A$ is performed in state $s = S_t \in S$. According to Barto et al., a value function, computing the *expected reward* $\mathbb{E}[.]$, w.r.t. a policy π, is defined as follows, where γ is the *discount factor* that controls how strong an immediate reward is affecting the value of the value function. Thus, unlike Eq. 2.30 and Eq. 2.32, Eq. 2.31 and Eq. 2.33 also take into account the discount factor γ for future steps.

$$v_\pi(s) \doteq \mathbb{E}_\pi[G_t|S_t = s] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \middle| S_t = s\right], \text{for all } s \in S \tag{2.31}$$

Equation 2.32 depicts the action value function $Q^\pi(s, a)$ that computes the expected $\mathbb{E}$ reward value R_t of performing an action a in a certain state s according to a policy π.

$$Q^\pi(s, a) = \mathbb{E}_\pi[R_t \mid s_t = s, a_t = a] \tag{2.32}$$

Equation 2.33 depicts the action value function with discount factor γ:

$$q_\pi(s|a) \doteq \mathbb{E}_\pi[G_t|S_t = s, A_t = a] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \middle| S_t = s, A_t = a\right]. \tag{2.33}$$

Despite the previously mentioned value functions, usually in RL, the policies are computed by means of a *quality value function* [242] that is a Bellman equation. The Bellman equation can be derived from the previously discussed value functions. However, the mathematical derivations will not be presented within this work. Equation 2.34 depicts the Bellman equation (q-function).

$$Q^*(s_t, a_t) = \sum_{t=0}^{T} r_{t+1} + \gamma * max(Q^*(s_{t+1}, a_{t+1})) \qquad 0 \leq \gamma \leq 1 \tag{2.34}$$

The quality value Q^* of a state-action input at time t is computed by considering the reward r_{t+1} for choosing the action a_t plus a discount factor gamma multiplied with the maximum expected quality value of the next state s_{t+1} and the next related action a_{t+1}. The gamma discount factor helps to balance the importance of the quality values of future states. The value range of the discount factor is [0, 1].

By using the quality value function, the agent iteratively computes a strategy that promises the highest expected reward for each state-action pair of a task.

In RL, an agent has to face the *credit assignment problem (CAP)* that first was raised by Marvin Minsky in his work *Steps towards Artificial Intelligence* [166]. There exist three types of the CAP:

i. Temporal CAP – The problem to figure out which of the actions within an action sequence lead to the assigned reward or punishment.

ii. Structural CAP – The problem to identify which system-immanent, i.e. structural, influences (e.g. observed states or cooperating agents) have contributed to the sequence of actions and thus to the achieved result.

iii. Transfer CAP – The problem to generalise learned action sequences across different tasks (see *Transfer learning*).

The most considered CAP in RL is the temporal CAP. For instance, an agent receives delayed rewards, which means that a longer sequence of actions has to be performed by the agent until a feedback is given. In order to solve the mentioned problem, there are two kinds of agent activities during learning, called *exploration* and *exploitation* [242]. Exploration means that the agent performs randomly an action without considering the collected utility or quality values. In the exploitation step the agent performs actions that have the highest accumulated utility value. Both steps are required for discovering the state-action space and learning policies.

At the beginning, the agent initiates different actions without knowing if this leads to a high reward value [172]. The agent learns over several thousand episodes which actions in the observed states lead to the maximum expected cumulative reward. The obtained cumulative reward values for each state-action pair are stored either in a table where rows and columns represent states and actions, or in a neural network that trains the model parameters for each state-action pair. If some quality value functions are already computed, the agent is also able to exploit (decide) based on its computed Q-values which action to select in order to maximise its cumulative reward. According to [242], the agent should alternate between exploration and exploitation, since the exploration allows to find unknown and unapparent constellations, which can maximise the future expected cumulative reward of the agent. In particular at the beginning of the learning phase, the exploration of actions is necessary.

In order to balance exploration and exploitation, usually a *greedy strategy* is applied by defining a probability ϵ that specifies when a random action and when a trained action is performed (see Eq. 2.35).

$$a_t = \begin{cases} a_t^* \text{with probability 1-}\epsilon \\ \text{random action with probability}\epsilon \end{cases} \tag{2.35}$$

In this way the agent can explore the environment's feedback (reward) and exploit the already learned knowledge (experience).

2.3.11. Hyperparameter Optimisation

The most ML algorithms require besides of initial weights, fix hyper-parameters (e.g. learning rate, greedy value, discount factor, etc.) to train their models. Hyper-parameters control the training process and are usually defined before the training starts. Depending on the required hyper-parameters, the performance of a ML model can vary considerably. Therefore, finding and setting the hyper-parameters that contribute to the best prediction results is crucial for ML tasks.

In several papers [139, 21, 226, 82] strategies and approaches (e.g. grid search, random search, Bayesian, gradient-based and evolutionary optimisation) for finding hyper-parameters are discussed and proposed to optimise the performance of ML models with

respect to evaluation metrics. The outcome of a hyper-parameter search is a set of hyper-parameters that promises the best evaluation metric results.

The applied evaluation metrics (see Sec. 2.3.7) depend on the appropriate ML algorithm that requires the hyper-parameters. In supervised ML settings often metrics such as precision, recall, F1, ROC/AUC are applied, while in RL settings, the performance of trained policies is usually evaluated by the gained cumulative reward function. Since in this work, RL agents are in focus, the cumulative reward function and required number of steps or episodes until the agent has learned optimal, i.e. reward maximising policies, are adopted as performance metrics, also regarding hyper-parameter optimisation. In this work, hyper-parameter optimisation is part of the agent-lifespan and is applied before the initialisation and training of the agent starts. Therefore, the following two optimisation algorithms for hyper-parameter tuning are considered and proposed for the agent framework.

2.3.11.1. Grid Search

In *grid search* beforehand a subset of parameters has to be defined for evaluation. Usually, common adopted values are taken that base on experience and assumptions. The appropriate ML algorithm (agent function) is then executed several times with different combinations of the according parameters, while the prediction outcomes (cumulative reward) are recorded. To get the mean reward value of the prediction performance, the cumulative reward is averaged by the performed episodes. The evaluated hyper-parameter combination with the best evaluation results is then chosen for the according ML algorithm.

Algorithm 1 represents the corresponding procedure. After evaluating the different combinations of hyper-parameters with the agent function, they are sorted by the performance of the parametrised estimator in descending order.

2.3.11.2. Random Search

In contrast to grid search, in *random search* it is not required to set a fix subset of hyper-parameters manually. Only the parameter range is necessary for every considerable hyper-parameter, in order to randomly determine the appropriate parameter values within the given ranges. However, the number of how many random values have to be generated, is defined by the developer. The random parameter values are then tested by the algorithm in one or several iterations. The more random parameters are tested, the more the hyper-parameter subspace is covered. Also here the testable parameter size depends on how much computational resources are available. The best performing parameter configuration is then selected by the random search algorithm as well as in grid search.

Algorithm 2 demonstrates the procedure of random search.
Grid search and *random search* are intended to be used in the agent framework and the agent developer can decide which of the optimisers to select and apply.

Algorithm 1: Grid Search

Input: set of selected hyper-params, iterations, episodes, agent function
Output: best performing hyper-params

```
1  Function GridSearch(paramSet, iterations, episodes, funcName):
2      evaluatedParams = []
3      for i = 0; i < iterations; i++ do
4          result = 0
5          selectedParams = combineParameters(paramSet)
6          for j = 0; j < episodes; j++ do
7              result += funcName(selectedParams)
8          result /= episodes
9          evaluatedParams.push([selectedParams, result])
10     evaluatedParams.sortByResult(ascending)
11     return evaluatedParams[0]
12
13 Function Main:
14     iterations = 10
15     episodes = 30
16     selectedHyperparams = [...]
17     bestResult = GridSearch(selectedHyperparams, iterations, episodes,
         agentFunction())
18     store(bestResult)
19     return 0
```

Algorithm 2: Random Search

Input: search size, hyper-param range, hyper-param names, iterations, agent function

Output: best performing hyper-params

```
1  Function RandomSearch(randomSearchSize, iterations, paramNames, ranges,
     funcName):
2      evaluatedParams = []
3      for i = 0; i < randomSearchSize; i++ do
4          evalParamSet = []
5          forall param in paramNames do
6              value = getRandom(ranges[param].min, ranges[param].max)
7              evalParamSet.push([param, value])
8          result = 0
9          for i = 0; i < iterations; i++ do
10             result += funcName(evalParamSet)
11         result /= iterations
12         evaluatedParams.push([param, result])
13     evaluatedParams.sortByResult(ascending)
14     return evaluatedParams[0]
15
16 Function Main:
17     iterations = 10
18     selectedHyperparams = [...]
19     bestResult = RandomSearch(selectedHyperparams, iterations)
20     store(bestResult)
21     return 0
```

Hyper-parameter optimisation can be, if required, performed in parallel in order to save time. However, it depends on the given computational resources, such as computation power, available processors, RAM, etc.) whether the hyper-parameter optimisation is performed in parallel execution.

2.3.12. Word/Entity Embeddings

In 1957, J.R. Firth wrote in [74]: *"You shall know a word by the company it keeps."*. Word embeddings operate based on this principle of *distribution similarity*, because words in text corpora are distributed in the word embedding (vector) space depending on their contextually surrounding words.

Since the proposed agent framework adopts word embedding techniques in order to a) compute the similarity between different agent task representations, i.e. task entities, and b) represent contexts numerically, this section provides the foundational knowledge for word embeddings and their utilisation within this work. However, in the scope of this work, the term *Entity Embedding* will be utilised since not words but semantic entities of MDP tasks are encoded within embedding spaces.

Simply spoken, *Word/Entity Embeddings* are vectors of real numbers (e.g. trained weights), representing single words in order to allow the quantification, processing and examination of these words respectively entities within data mining and ML algorithms. This is necessary due to the fact that the mentioned algorithms require numerical vector representations in order to process them.

Word embeddings are in particular used in *Natural language processing (NLP)* to capture the context and semantics (meaning) of words within phrases of huge text corpora. Moreover, word embeddings have very interesting properties that provide useful insights about the semantic relationship of words to each other. For instance, Allen and Hospedales demonstrate in their paper [6] that word embeddings enable to find semantic relations such as *King relates to Man as Queen relates to Woman*, by performing simple mathematical vector operations, such as i.e. addition and subtraction. For instance, the following mathematical operations on the appropriate word vectors are possible: Subtract the word embedding vector of man from king and add the embedding vector of woman to the previous vector, which results in the embedding vector of queen [6].

$$\mathsf{WE}_{\mathsf{King}} - \mathsf{WE}_{\mathsf{Man}} + \mathsf{WE}_{\mathsf{Woman}} = \mathsf{WE}_{\mathsf{Queen}} \tag{2.36}$$

Said that, the (dis)similarity and semantic relatedness of word vectors, respectively words can be measured by computing the *Cosine similarity* (see Eq. 2.37) respectively *Cosine distance* (see Eq. 2.38) or the *Euclidean distance* (see Eq. 2.39). For instance, the cosine similarity of nearly one indicates a high similarity between two word embeddings, while a similarity value between zero and minus one indicates a high dissimilarity between those words.

However, the situation is different with the Euclidean and Cosine distance. An Euclidean distance or a Cosine distance of nearly zero indicates a very low distance and thus a high similarity. Computing the similarity or distance of entity embeddings is required to enable agents to find reusable policies for similar tasks. In the approach chapter (see Sec. 4.5.1 and 4.5.2), it is outlined in detail, how entity embeddings were adopted and trained for agent tasks and their properties in order to derive matching policies.

$$\cos_{similarity}(\mathrm{p}, \mathrm{q}) = \frac{\mathrm{pq}}{\|\mathrm{p}\|\|\mathrm{q}\|} = \frac{\sum_{i=1}^{n} \mathrm{p}_i \mathrm{q}_i}{\sqrt{\sum_{i=1}^{n} (\mathrm{p}_i)^2}\sqrt{\sum_{i=1}^{n} (\mathrm{q}_i)^2}} \tag{2.37}$$

$$\cos_{distance}(\mathrm{p}, \mathrm{q}) = 1 - \frac{\mathrm{pq}}{\|\mathrm{p}\|\|\mathrm{q}\|} = 1 - \frac{\sum_{i=1}^{n} \mathrm{p}_i \mathrm{q}_i}{\sqrt{\sum_{i=1}^{n} (\mathrm{p}_i)^2}\sqrt{\sum_{i=1}^{n} (\mathrm{q}_i)^2}} \tag{2.38}$$

$$\mathrm{d}(\mathrm{p}, \mathrm{q}) = \|\mathrm{p} - \mathrm{q}\|_2 = \sqrt{(p_1 - q_1)^2 + \ldots + (p_n - q_n)^2} = \sqrt{\sum_{i=1}^{n} (p_i - q_i)^2} \tag{2.39}$$

Single entities in entity embeddings are usually represented in a high-dimensional vector space. To visualise entity embeddings, however, dimensionality reduction techniques such as T-Distributed Stochastic Neighbour Embedding (T-SNE) [145], Principal Component Analysis (PCA) or Uniform Manifold Approximation and Projection (UMAP) [149] are required.

For instance, Google (Tensorflow) applies the mentioned dimension reduction techniques in the *Embedding Projector*[24], a web application that allows the upload and visualisation of word respectively entity embeddings in a two or three dimensional space.

Word or entity embeddings can be constructed in different ways (e.g. one-hot encoded vectors, word frequencies, co-occurrence matrix, etc.), however the most common method is to train embedding layers in an arbitrary Neural Network (NN), because in contrast to the earlier mentioned approaches, embeddings trained by NNs maintain the contextual meaning of words and entities. The corresponding NN architecture and supervised learning task depends on the problem domain for which the embedding layers have to be trained for. It is common between the different NN architectures to train the weights of the first hidden network layer by back-propagation. The embeddings layer is usually initialised with random weights. Figure 2.9 illustrates exemplary a deep neural network (DNN) with an embedding layer.
There are two popular types of embedding architectures, *Skip-gram* and *Continuous Bag of Words (CBOW)*. While in the CBOW architecture, the network predicts the embedded target word by the contextual surrounding words, in the Skip-gram architecture, the network predicts the contextually surrounding words by the appropriate target word. In

[24] `http://projector.tensorflow.org/`

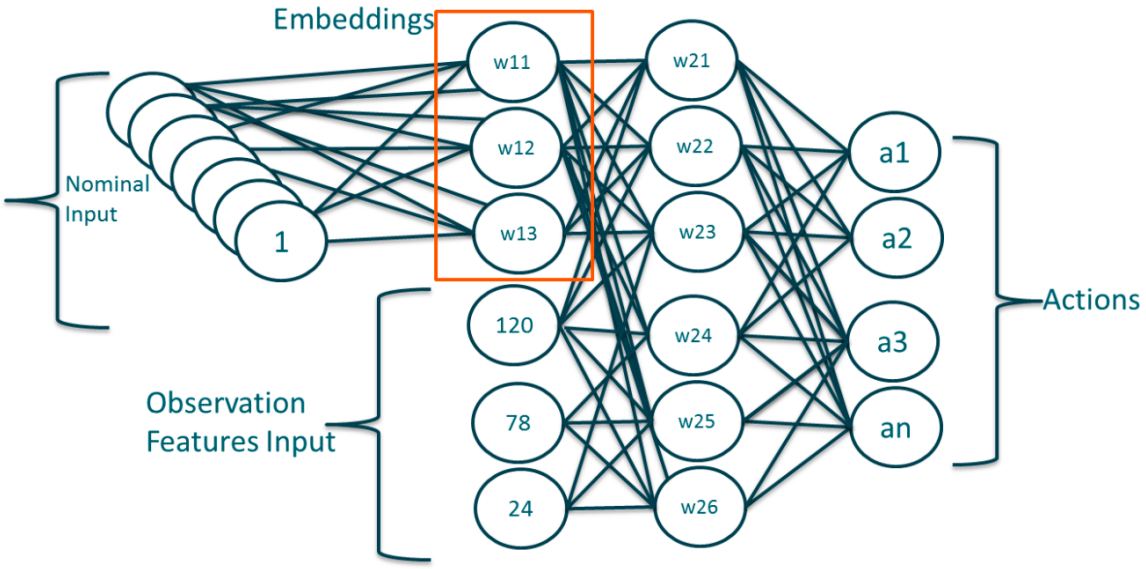


Figure 2.9.: An exemplary deep neural network with a word embedding layer.

both approaches the context of words plays an important role.

The number of neighbouring words that are considered by the NN, is defined by an arbitrary window size. For instance, if a window size of 2 is selected for the following sentence: *London is the **capital** of England*, then the appropriate word tuples, related to the target word *capital* will be as follows:

(capital, is), (capital, the), (capital, of), (capital, England).

In most of the cases, it is not required to train embedding layers from scratch because there are already pre-trained models (Word2Vec[25], GloVe[26], etc.) freely available that can be used for different NLP tasks (e.g. sentiment analysis). However, depending on the purpose of the specific embeddings, it can be required to train an own model that addresses the own task requirements. In this work, different embedding layers are trained for different task properties, i.e. entities, because the similarity of tasks is to be determined.

2.3.13. Genetic Algorithms

One objective of the proposed framework is to support different types of ML and optimisation algorithms to find task-specific strategies that can be applied by software agents. The implementation of a genetic algorithm (GA) is supported by the proposed agent framework. GAs can be used for different purposes in the context of the framework. One purpose is to create successful agent strategies (i.e. action sequences) based on executable agent actions. Another purpose is to combine actions of multiple agents and create agent populations

[25] https://code.google.com/archive/p/word2vec/
[26] https://nlp.stanford.edu/projects/glove/

that collectively contribute to solving tasks in a multi-agent system (MAS). In future works, it is also considerable that agent developers utilise GAs together with semantic model representations for finding certain agent skills that enable an agent to solve a variety of tasks, depending on given user and task requirements.

The foundational knowledge about GAs is adopted from the following literature: [86, 162, 95].

GAs are a subset of evolutionary algorithms (EAs) and are inspired by Charles Darwin's theory of natural evolution and selection [55]. They are primarily meta-heuristics that search for near-optimal solutions in mostly huge solution spaces. The advantage of GAs is that they are able to find sufficient solutions to NP-hard problems without requiring model knowledge or prior knowledge about a problem space. However, a transformation between real world (i.e. Phenotype) and computation space (i.e. Genotype) is required in order to achieve a mapping between both spaces.

GAs follow the principle of the *survival of the fittest*, which means that only successful solutions survive and prevail. The *fitness* of solutions is measured by a problem specific fitness function that allows to assess the success of a solution.

GAs apply various *genetic operators* (e.g. parent selection, recombination, mutation, survivor selection) that mimic the evolutionary process in nature. While *mutation* contributes to a high diversity of solutions within the populations, recombination ensures that successful solutions survive in subsequent generations by reproducing themselves. Mutation and recombination each occur with a certain probability, which is set in advance as a hyper-parameter.

A problem in GAs is the *premature convergence to local optima.* This means in simple terms that the algorithm stagnates before finding a global optimum what yields in a suboptimal population and solution of the problem. To overcome this problem different strategies (e.g. randomisation, crowding selection [37]) are applied that facilitate a high diversity of solutions and the reduction of *crowding*. Crowding means in this context that the fittest solutions survive and displace less successful solutions which in turn reduces the diversity of solutions and leads to homogeneous solutions. Since there is a trade-off between exploration and exploitation, a balance between the two should be maintained, with sufficient diversity of solutions and reproduction of good solutions. The corresponding strategies are very problem specific and so for each problem different strategies have to be tried.

The terminology of GAs is based on Darwin's theory of evolution. Thus, the following terms are commonly used in the discourse of GAs:

Phenotype A Phenotype represents problem specific solutions in real world environments. For instance the controlling and execution of certain peripherals and devices by specific commands in the physical environment of the agent, are phenotype representations.

Genotype A genotype represents in a simplified and numerical way solutions in the computation space so that genetic operators can easily manipulate the given solution representations by numerical and permutation operations.

Decoding and Encoding The representation and transformation of problem solutions from a computational representation (i.e. genotype) to a real world application (i.e. phenotype) and vice versa is called *decoding* and *encoding*. Depending on the problem and its complexity, genotype and phenotype representations can be more or less congruent. Decoding and encoding operations require model knowledge about the problem space and its allowed solutions, their representation and their mapping to both spaces.

Population A population represents the current subset of all possible solutions of the entire solution space. The population size can vary according to the given problem. Depending on how fast solutions have to be found, the population size has to be adjusted. A population is manipulated by according genetic operators that stochastically alter the individual solutions of a population.

Chromosome Chromosomes represent individual solutions within a population. Each chromosome can be altered by the given operators. They can have further properties such as an age that indicates the existence time span of the chromosome, and a fitness value that allows the GA the assessment of the chromosome's success.

Gene A gene is an atomic element within a chromosome. If a chromosome is considered as an array of values, one field of this array represents a gene. A gene can be considered as a property of the chromosome.

Allele An allele is a possible value that a gene can take. For instance in binary problems, the allele would consists of two possible values e.g. 0 or 1. However, depending on the phenotype and genotype representation also more values of different types (e.g. integers, real values, strings) are possible.

Genetic Operators In order to change and reproduce chromosomes different operators such as *mutation, recombination* and *selection* are iteratively applied.

Fitness Function Fitness functions are problem-specific and allow the assessment of solutions. A fitness function evaluates the given solutions and provides a fitness value that represents the performance of the solution for the given problem. Fitness functions should be easy and quick to calculate because a GA is an iterative algorithm that multiple times computes via fitness functions the success of provided solutions.

The general process flow of GAs looks as in Algorithm 3. Firstly, a random population with a certain size is generated. Depending on the problem the population size can vary. Usually the population size comprises between hundreds or thousands chromosomes. The higher the size is, the longer takes the computation time for a finite number of iterations.

However, a small population size lacks after some time a high diversity of solutions. It is important to find a balance between sufficient diversity, computation time and resources.

For each chromosome of the initial population a fitness value is computed and assigned. Subsequently, a *while-loop* starts that first terminates if one of the conditions are met. In this case, the conditions are a maximum number of iterations and an optimal fitness value. From the population an amount of chromosomes are selected for mating with each other in order to produce offsprings. The selection can be made by different selection strategies, e.g. *tournament selection, rank selection, fitness proportionate selection, stochastic universal sampling, etc.*

Recombinations and mutations are performed with a pre-defined probability. Hence, a random number between 0.0 and 1.0 is generated and if this number is lower than the set probabilities then the corresponding operator is applied. Firstly, offsprings are generated via the recombination of the selected parents. Then, the mutation operator is applied on the population. For the mutation, different operations can be performed. The basic idea is to make small changes on the chromosomes in order to generate and explore new chromosomes that provide new solutions.

For each new generated or reproduced chromosome the fitness function is applied in order to compute its fitness.

The *survivor selection* is done on the end of the while-loop. There are as well different strategies for selecting the survivors of the population. One strategy is age-based selection. In every iteration (i.e. generation) the age of a chromosome is increased and the oldest chromosomes are kicked out of the population without considering the fitness of chromosomes. Thus, each chromosome has a finite life time within the population during that it can reproduce itself. A second strategy is the fitness-based selection. As in the parent selection, different fitness-based strategies can be applied, e.g. ranking, tournament selection, fitness proportionate selection, etc. In this way, only the fittest chromosomes survive in the population while less chromosomes are replaced by their children.

The termination of the GA algorithm depends on the objectives of the GA designer. The purpose is to achieve near optimal solutions. There exist different termination criteria that allow the determination of near optimal solutions. In order to determine the convergence of the algorithm the following termination conditions can be utilised:

- Convergence of the best fitness value for a predefined number of generations.
- Achieving a predefined limit of generations.
- Achieving the targeted fitness value.

As soon as the termination condition(s) is/are met, the while loop is left and a sufficiently optimal population is returned (see Algorithm 3).

Algorithm 3: Genetic Algorithm

```
   Input: populationSize, mutationProbability, crossoverProbability,
          optimalFitnessValue, numMaxIterations
   Output: population
 1 Function geneticAlgorithm(populationSize, mutationProbability,
     crossoverProbability, optimalFitnessValue, numMaxIterations):
 2     population = []
 3     bestFit = 0
 4     iteration = 0
 5     mutatedPopulation = NULL
 6     recombinedPopulation = NULL
 7     population = randomInitialPopulation(populationSize)
 8     fitnessFunction(population)
 9     while (bestFit < optimalFitnessValue) && (iteration < numMaxIterations) &&
         !isConverged(population) do
10         crossoverProb = getRandomDouble(0.0, 1.0)
11         if crossoverProb <= crossoverProbability then
12             selectedParents = selectParents(population)
13             recombinedPopulation = recombine(selectedParents)
14             fitnessFunction(recombinedPopulation)
15
16         mutationProb = getRandomDouble(0.0, 1.0)
17         if mutationProb <= mutationProbability then
18             mutatedPopulation = mutate(population)
19             fitnessFunction(mutatedPopulation)
20
21         concatPopulation = concat(population, mutatedPopulation,
             recombinedPopulation)
22         population = selectSurvivors(concatPopulation)
23         bestFit = getBestFitnessOf(population)
24         iteration++
25
26     return population
27
```

2.4. Decision Rule Mining

Since processes, environments and people can generate a huge amount of data through the effects of their behaviour in real-world environments, a methodology for processing and using knowledge from corresponding history data is required. Decision rule mining approaches are useful for this very purpose. In order to discover policy rules, various rule mining approaches have been considered and adopted for this work. The following sections outline the corresponding decision rule mining approaches that enable the extraction of policy rules from history data. The presented knowledge about Frequent Item sets, A-priori algorithm, FP-Growth algorithm, Sequential Covering, OneR and Association Rules were taken from the books, *Mining of Massive Datasets (MMDS)* [134] and *Interpretable Machine Learning (IML)* [173].

One objective in this work, is the discovery of decision rules, respectively policy rules, following the structure in Eq. 2.40. A rule consists of a list of observations that the agent makes in the environment and a derived action that is one of the actions the agent can take.

$$\{\text{observation}_1, \text{observation}_2, ...\text{observation}_n\} \rightarrow \{\text{action}_n\} \tag{2.40}$$

2.4.1. Frequent Item sets

Many different algorithms (e.g. A-priori algorithm, FP-Growth, etc.) have been developed in rule mining to find *frequent item sets* in *Big Data*. Frequent item sets in *transactions* (i.e. samples) are considered to determine frequent item co-occurrence. Frequent item sets are useful, for instance, to infer customer buying behaviour in recommender systems. A popular sample Rule 2.41, which is also mentioned in the MMDS book, is:

$$\{\text{diaper}, \text{milk}\} \rightarrow \{\text{beer}\} \tag{2.41}$$

The example rule in Eq. 2.41 indicates that customers who buy diaper and milk are likely to buy also a six-pack of beer. This information can foster different sale strategies in order to maximise the profit.

Extracting frequent item sets can be considered as a data pre-processing step that reduces the amount of data considerably, before decision rules, respectively patterns, can be discovered by applying subsequent statistical metrics, such as *confidence, interest* and *lift*.

However, preceding to the discovery of frequent item sets, an additional metric is required that determines the *support* or coverage of an item set. Equation 2.42 demonstrates how the support value of item sets is computed. The number of transactions (e.g. sets of purchased items, here x and y) that appear together in a transaction, is counted and divided by the number N of all recorded transactions. If the specified support threshold is surpassed by the computed support value, an item set counts as frequent item set. For this reason, every item set combination, respectively transaction, in the dataset has to be considered by the

appropriate algorithm that determines frequent item sets.

In a first step, the support value of single items is computed while the algorithm iteratively combines different items and determines their support value. The support threshold is a hyper-parameter that beforehand has to be specified, depending on the use case and dataset. According to the *Mining the Massive Datasets (MMDS)* book [134], a rule of thumb is that 1% of the data amount is taken as support threshold value. For instance, if 10000 transactions are given, a threshold factor of 1% will set the support threshold to the number 1000. However, it depends on the dataset and the use case which support threshold is ultimately useful.

$$\text{support}(\{x\} \rightarrow \{y\}) = \frac{(x \sqcup y)}{N} \tag{2.42}$$

Extracting frequent item sets requires a huge amount of computational resources (i.e. RAM and CPU). In most cases, the volume of data is so large that customised algorithms have to be used to process the huge amounts of data in main memory. The widely used A-priori algorithm turns out to be very slow compared to the FP-Growth algorithm. For this reason the FP-Growth algorithm that is building and traversing recursively an item tree, is preferred for extracting frequent item sets. The concrete implementation of the mentioned algorithms are out of the scope of this work and can be found e.g. in the referenced book [134].

2.4.2. Association Rules

Association rules are *if-then* rules consisting of an antecedent (i.e. conjunction of conditions) and a consequent (conclusion), that represent the relationship (co-occurrence) of items. Association rules are identified based on previously discovered frequent item sets. For this reason, different metrics such as *confidence (accuracy), lift* and *interest* are applied in order to determine association rules.

The confidence of an association rule represents the *probability of a target item y given one or several items* x_n. The computation of the confidence value by means of the appropriate support values, is shown in equation 2.43.

$$\text{confidence}(\{x\} \rightarrow \{y\}) = \frac{\text{support}(x \sqcup y)}{\text{support}(x)} \tag{2.43}$$

However, the confidence alone is not a sufficient indicator for the significance of an association rule, since considering the confidence value alone, can lead to misinterpretations, because a high frequency of the target item influences the confidence value positively. Is an item set frequently present, the confidence value will be highly independent of the co-occurrence of items, which means that the discovered association rule may be not necessarily significant. For this reason also the *interest* (see Eq. 2.44) of an item set has to

be considered. The interest of an association rule determines the difference between the confidence of the association rule and the probability of the consequent rule item. Is the value highly positive or negative (e.g. > 0.5 or < -0.5), then the corresponding association rule seems to be significant.

$$\text{interest}(\{x\} \rightarrow \{y\}) = \text{confidence}(\{x\} \rightarrow \{y\}) - P(\{y\}) \tag{2.44}$$

The lift value indicates the likeliness of the target item y with respect to its frequency and the occurrence of the independent items (here x). For the sake of completeness, Eq. 2.45 depicts how the lift value of an association rule is determined. A lift value greater than one indicates that the target item y is likely to occur if the appropriate independent items occur. A lower value than one indicates the opposite, which means that the target item y will most likely not occur if the independent item x occurs.

$$\text{lift}(\{x\} \rightarrow \{y\}) = \frac{\text{support}\{x, y\}}{\text{support}\{x\} \times \text{support}\{y\}} \tag{2.45}$$

Discovering association rules by certain metrics is not the only method to gain decision rules from huge datasets. The next two possibilities that are presented are the *OneR* (stands for one rule) algorithm and the *sequential covering* algorithm. Furthermore, *Bayesian rule lists* provide also a possibility to derive decision rules. However, they will be not discussed here, because they were not considered or applied for this work.

2.4.3. OneR

OneR is an algorithm that has been proposed by Holte [106]. The algorithm allows the derivation of decision rules from one single feature. The advantage of OneR is its simplicity in terms of its implementation and representation. Since the rules are represented by only one feature and its corresponding possible values, the rules are concise and interpretable. OneR discovers the most informative feature that allows the best distinction of predictable classes. The implementation of OneR is outlined in Algorithm 4.
Before applying OneR, numerical features have to be transformed into discrete features, for instance by binning. Then, for each feature-value pair and prediction class, the algorithm counts the co-occurrence. The underlying assumption is that the class that has the highest co-occurrence with the corresponding feature-value pair is considered the correct prediction class. Based on this assumption, the number of misclassifications is counted in a cross-table for each feature-value pair to determine the prediction error that results from the sum of the incorrectly predicted classes. Finally, the feature rules that give the lowest error rate are selected as decision rules.

2.4.4. Sequential Covering

Sequential Covering is a widely used algorithm that follows the principle of *divide and conquer*. The approach is to remove all data points from the dataset step by step, taking

Algorithm 4: OneR

```
   Input: dataset, classes, features
   Output: rules of one feature
1  Function OneR(dataset, classes, features):
2      crosstable = []
3      forall sample IN dataset do
4          forall featValuePair IN features do
5              if featValuePair NOT IN crosstable then
6                  crosstable[featValuePair] = 0
7              forall class IN classes do
8                  if featValuePair IN sample AND class IN sample then
9                      crosstable[featValuePair] += 1
10     featRules = selectFeatRulesWithLowestError(crosstable)
11     return featRules
12
```

into account the currently considered and learned rule that covers the corresponding data points in the sample. In doing so, a rule list is kept in which all considered and learned rules are collected. To determine *significant* rules from the data, an algorithm (e.g. CART tree, bar search) is required to find the purest path to the corresponding class node. This is implemented in Algorithm 5 by the function *learnOneRule*.

The implementation of the sequential covering algorithm differs depending on whether it is a binary or a multi-class classification problem. Since a multi-class classification problem is given in this work, only the implementation for multi-class classification is discussed (see Algorithm 5).

According to [173], classes have to be ordered by the least prevalent class in ascending order. The algorithm starts with the least common class and its prediction rules and moves to the next class rules after a rule is learned. Iteratively, data points covered by the considered rules are removed from the data samples. The rules of the most prevalent classes are set as default rules.
It is important to note that the most common class rules are marked as default rule. A default rule is a rule that applies if no other rule applies, thus it has no if-path. Through default rules, always a prediction is possible also in cases when no other rule meets the conditions.

A general problem with decision rules is the overlapping of rules. It is possible for rule conditions to apply to conflicting class predictions, resulting in an uncertain prediction. Various strategies are used to avoid rule overlap. For example, decision lists bring order to the rules. For class prediction, the first rule in the list whose conditions apply is selected

Algorithm 5: Sequential Covering

```
   Input: dataset, classes
   Output: list of decision rules
1  Function SequentialCovering(dataset, classes):
2      classdistribution = []
3      rulelist = []
4      forall sample IN dataset do
5          forall class IN classes do
6              if class NOT IN classdistribution then
7                  classdistribution[class] = []
8              if class IN sample then
9                  classdistribution[class].push(sample)
                                  // Consider least common classes' rules first.
10     classdistribution.sortBy(sampleSizePerClass, ascending)
11     while samples.length > 0 do
12         forall class IN classdistribution do
13             bestRule = learnOneRule(class, classdistribution)
14             rulelist.add(bestRule)
15             samples.removeAll(bestRule)
16     return rulelist
17
```

as the prediction rule.

In decision sets, rules either are weighted by their accuracy or by majority voting methods that allow rules the same voting right. In this way, the overlap of rules can be addressed and handled.

2.5. Summary of this chapter

In this chapter, the basics on which this thesis is built were discussed. The terms *agent* and *task environment* as well as their properties and requirements were clarified. Furthermore, the most important building blocks of semantic technologies (e.g. serialisation formats, rule-based languages, description logics, etc.) were introduced. In this thesis, semantic technologies are needed to formalise and represent domain knowledge about agents and tasks, and to share and reuse this knowledge so that agents are able to understand and interact with their environment and each other.

In the second half of this chapter, ML as foundational building block was outlined. As we have seen, in ML different paradigms and approaches for optimising the learning strategies of agents are applied. By ML, agents are enabled to learn their strategies depending on task inherent process characteristics. We have seen that learning is required in cases where the dynamics of the environment and certain strategies cannot be anticipated at design time and where agents require to adapt their behaviour to these changing dynamics. In particular, RL is important in this work, since a subset of agents that use the proposed framework implement RL strategies in order to learn policies for solving varying MDP tasks.

However, in order to successfully apply ML, certain conditions have to be considered. For instance, hyper-parameter optimisation is a crucial part of ML that contributes to the fail or success of ML algorithms. Different optimisation algorithms (genetic algorithm, grid search, random search) as well as probabilistic classifiers such as the *Multi-nomial and Gaussian Naive Bayes classifiers* were introduced that are adopted by the agents of the proposed framework.

Another aspect of the proposed agent framework is the derivation/extraction of decision rules representing policies. This chapter has discussed various techniques for extracting decision rules that have been tested and applied to derive data-inherent policy rules. These rules are needed, on the one hand, to enable rule-based agents to perform tasks without having to train new rules from scratch and, on the other hand, to make the agents' behaviour explainable. In this way, the approach facilitates the integration of new agents, making expert knowledge reusable and comprehensible.

By adopting and combining different technologies, the proposed approach is intended to enable agents to gather domain knowledge, learn to solve problems autonomously and adapt themselves to new upcoming situations.

3. Related Work

In this chapter, related work is considered with different problem areas that the agent framework also addresses with its contributions. The goal is to distinguish the approaches presented in this thesis from related work and to demonstrate the relevance, novelty, and utility of the approaches implemented in the agent framework presented. The structure of this chapter is based on the identified themes and contributions of this thesis and is organised as follows.

Section 3.1 considers related work that addresses the *cold-start problem* outlined earlier, which arises particularly in recommender systems and user-centred application systems where software agents need to interact with users. However, it also occurs in systems where agents perform tasks and provide services in different environments and contexts. State of the Art (SOTA) approaches use either *simulation frameworks* [180, 276, 15], *learning by demonstration* [219, 244, 30, 239, 78] or by *imitation* [122, 201, 103, 130] to overcome the cold-start problem or for other purposes. Other approaches use *crowd sourcing, active learning* [66, 67, 144, 220, 212, 222] and *human-in-the-loop approaches* [177] to obtain representative data and domain knowledge that can enable agents to act in an intended way at start-up time. The contributions *C1.1 - C1.2* of this work are compared with the contributions of the considered related works on the basis of various criteria listed in Table 3.1.

Behavioural data from users and processes can be used to derive decision rules for agents and thus extract task strategies from the corresponding data. In this context, personalisation or contextual adaptation of policies plays a crucial role when default strategies, i.e. policies, have to be adapted to individual requirements. Section 3.2 considers various works that apply mining techniques such as association rule mining, sequential rule mining and process mining to derive decision rules from behavioural data. Moreover, approaches such as reinforcement, transfer and incremental learning approaches are considered, which deal with the adaptation of policies, models and services. The objective is to show what advantage the contributions *C2.1.1 - C2.2.2* offer in comparison with the related works. Therefore, the criteria in Table 3.2 and Table 3.3 are used.

Context-aware behaviour is a crucial requirement that software agents are expected to fulfil. The proposed agent framework implements an approach that enables the context-aware combination of policies. To evaluate this approach, three different baseline approaches were considered (see Section 3.2.2) and compared in the evaluation (see Section 6.2.4.2) that use language models and natural language descriptions for the execution of activities in the *Virtual Home* environment.

Furthermore, policy mining, sharing, retrieval and reuse is an important part of the agent framework as its philosophy is to create, provide and openly share knowledge as well as support agent collaboration in an automated manner. Therefore, in Section 3.3, related works that deal with policy discovery and reuse are compared with the proposed approaches of the agent framework. For this reason, the contributions *C3.1 - 3.2* will be considered in order to show the novelty of the proposed agent framework.

Finally, Section 3.4 summarises the findings and discusses the differences between the proposed agent framework from the considered related works.

To the knowledge of the author of this doctoral thesis, there are no agent frameworks in the presented form that have the same purpose as the proposed agent framework. For this reason, only parts of the proposed agent framework and single approaches can be compared to existing related works.

3.1. Cold-Start Problem

The deployment of new tasks and agent instances requires knowledge about the given task, its properties and dynamics, so that software agents can immediately act in a well-defined and intended manner with respect to the task they require to solve. However, if this knowledge is not available, an affected system or agent cannot act in the intended manner. The problem described is called the *cold-start problem*. The proposed agent framework tries to overcome this problem with a methodology that allows the formalisation and simulation of MDP tasks, so that agents can train the strategies required to solve the task before they are deployed in a real operational environment. For this reason, the following contributions are intended to help overcome the cold-start problem:

C1.1 A lightweight task ontology that captures the different characteristics of MDP tasks.

C1.2 A task simulation framework that allows agents to train policies based on process datasets and domain knowledge.

The criteria for evaluating and comparing related work with the proposed contributions are whether the approaches considered are a) *domain-specific* or *cross-domain*, whether b) *datasets*, c) *guidelines* or *domain models* are required for training an intended behaviour and whether d) *user interactions* or e) *domain expert interventions* are required in advance. The above criteria are summarised in Table 3.1, and the corresponding works are evaluated with respect to these specific criteria.

Numerous scientific papers deal with the outlined cold-start problem, which indeed occurs in situations where there is no prior knowledge about environments, users and their preferences due to lack of representative data and adaptation time. The areas where cold-start is a common problem are divided into three different categories in this thesis (see Sec. 3.1.1 - Sec. 3.1.3): *recommendation systems, robotic systems* and *dialogue systems* that perform human-computer interaction (HCI).

3.1.1. Recommender Systems

Recommender systems are often confronted with the cold-start problem, since recommender systems have to take into account the specific characteristics and behaviour of new target users who apply or use the corresponding recommender system. Integrating new users means that their characteristics and behaviour have to be analysed and learned in order to make appropriate recommendations. This may require numerous interactions with the user and thus take some time until sufficient data and knowledge about the corresponding user are available.

For instance, Theocharous et al. propose a process that precedes the transformation of *passive data* into *active data* to overcome the cold-start problem that often occurs in user recommendation systems [249]. Here, the authors consider *passive data* to be data that does not consist of historically evolved sequences, i.e. recommendations that are not past. A cold-start problem is present in these recommender systems because historically evolved data on user-related recommendations is not immediately available and it takes some time to collect data consisting of past user interactions and corresponding recommendations. In particular, the methods of Theocharous et al. rely on datasets of user interactions with web applications to cope with the cold-start and thus rely on data collection. In addition, the considered approach is domain-specific. In contrast, the proposed agent framework uses semantic task descriptions with rules that can be defined by domain experts and used to simulate corresponding tasks. This means that no user interactions are required and all types of processes and data can be simulated based on simple task and rule descriptions. The lightweight framework ontology makes it possible to specify diverse and domain-independent tasks, allowing agents to train well-defined policies before deployment without requiring to wait for data.

In their work [204], Quiao et al. combine clustering and reinforcement learning to avoid the cold-start problem in software crowd-sourcing recommendation systems. To do this, they create developer and software project models based on user interactions that can be used to describe and cluster developers and projects. Similar to their approach, this work does apply reinforcement learning, but not for learning developer skills and preferences, but for learning task strategies for agents. Unlike the related work of Quiao et al., the proposed agent framework applies reinforcement learning in an off-line simulation of tasks and therefore does not require prolonged interactions with users to obtain feedback. The feedback provided in the agent framework comes from the simulation component, which uses task descriptions provided by domain experts. In this way, the exploration phase is shifted to off-line learning and can be largely omitted or reduced when executed in the real world. Moreover, the model they propose for developers and software projects is precisely limited to the domain at hand and therefore cannot be generalised to other application domains, as it is very specific to project recommendations for software developers, while the task ontology of the proposed agent framework offers good generalisability due to its semantic representation of Markov decision processes and abstract concepts such as states, actions, effects, etc.

The paper [196] by Poirier et al. proposes to annotate user web content based on the opinions expressed in the texts. Based on these opinion annotations, they create a "user-item rating matrix" to apply collaborative filtering for user-specific item recommendations. Their approach is based on the intersection of opinion mining and opinion classification, and correlation with item recommendations. However, this approach is only applicable to agents making user recommendations on online platforms which means that this approach is domain-specific. While one could use text to describe tasks and use NLP techniques to create MDP representations of tasks, this would be error-prone and difficult to implement, and depending on what important information is needed, there is no guarantee that all the needed information about a task, from human-readable text, can be extracted to create MDP knowledge graphs. For this reason, the considered approach seems to be not seamlessly applicable for cross-domain applications. For this reason, the agent framework requests the explicit description of tasks by domain experts who provide the necessary specifications and conditions under which the tasks can be performed. Furthermore, the considered approach requires datasets that are produced through user interactions with the platform.

Alaa et al. propose the application of an ontology to capture and represent target users and their item preferences and to draw conclusions for the creation of item recommendations [5]. In this way, the authors hope to overcome the cold-start problem prevalent in recommender systems for new users and items. The domain ontologies used are constantly updated according to the dynamic changes in user and item properties. Furthermore, the architecture consists of components that perform content-based filtering and collaborative filtering to make recommendation predictions and evolve the relevant ontology by adapting the corresponding ontology to the structure of the dataset used. Therefore, a reasoner infers similarities between user profiles and recommendations for users based on the ontology. Thus, user profiles are also needed to represent and infer user preferences. The problem with the approach considered is that it is not applicable if no data and no domain-specific knowledge (i.e. domain ontology) are available and no direct user interactions are possible. For this reason, in addition to the use of datasets, the proposed agent framework also allows for a purely formal and abstract description of task entities that can be defined and provided by domain experts in advance, so that the task entities can be used for task simulation purposes. In contrast, the proposed agent framework does not require ontology evolution as it is generalisable enough to capture and represent tasks and activities in the mentioned way. Only feedback from the environment (simulated or real) determines the adaptation of task strategies, i.e. policies. At the beginning of the life cycle, given the cold-start problem, well-defined default strategies are trained, which can later be adapted to user preferences and the environmental context, if necessary.

In the approach of Soleymannejad et al., the results of different recommendation methods, such as collaborative filtering, matrix factorisation, demographic filtering and content-based filtering, are combined by an operator called *ordered weighted averaging operator (OWA)* [283] to mitigate the user cold-start problem [234]. The considered approach can be considered as an *ensemble learning* approach where the results of each method are combined to obtain accurate recommendations for users about whom little or no data

is available in the recommender system. However, the considered related work requires still user interactions, some data and guidelines in order to work. In the proposed agent framework, similarly an ensemble learning approach is applied when predicting and combining strategies in environments with multiple contexts, e.g. home environments, outdoor environments, etc. The difference, however, is that not only different recommendation methods are evaluated by the proposed agent framework, but also different strategies are executed by different agent instances. In this way, policies are comparable in terms of their performance and utility. The advantage here is that policies that have already been successful over time are represented in a knowledge graph or dataset, which then serves as input for embedding vectors that represent different contexts. In this way, alternative action sequences can be tested in advance and those that lead a task to a successful end can be combined.

Di Tommaso et al. propose "an unsupervised semantic recommender" called *SeRenA* that relies on Wikipedia documents and social media data of users [62]. The Wikipedia documents and their categories serve as reference for user interests and preferences that are derived from social media data such as posts and friendship connections. The approach shows that semantically related knowledge can help to draw conclusions and find similarities between semantically represented entities. The proposed agent framework uses also the potential of semantic knowledge representation to represent MDP tasks and their properties so that the framework can generate prior knowledge in the form of task simulations. Moreover, strategies can be interlinked so that agents are enabled to autonomously create and find the knowledge or information they require. The difference to the approach considered in the agent framework is that semantic categories are not used as a reference for user interests, but that MDPs are represented as linked graphs, which then serve as the basis for the simulation framework. In contrast to the proposed agent framework, the considered related work is domain-specific, i.e. not applicable across domains, requires data from user interactions and a domain model of the recommender system.

The authors Jeevamol et al. also propose a semantic recommendation system, but for e-learning, that makes recommendations for learning content based on an ontology describing user profiles and learning units [114]. The approach applies machine learning techniques such as K-means clustering and collaborative filtering to find similarities between users based on their semantically described learning skills, learning level and interests. In addition, the used domain ontology enables the dynamic mapping between user profiles and learning content. In doing so, the approach takes into account changes in learner interests and abilities as well as changes in learning units by adjusting the corresponding profiles based on user assessments. However, the considered approach is domain-specific and even task-specific, requires data and user interactions until it can address the cold-start problem. Similar to the approach under consideration, the agent framework uses an ontology and creates a mapping between tasks, agents and policies. Since policies can change depending on the context, the agent framework also addresses the adaptation of policies based on environmental feedback. It might be obvious that the cold-start problem is closely linked to personalisation and dynamic context. However, rec-

ommendation systems rarely consider heterogeneous environments that can also change dynamically. The agent framework captures this by allowing domain experts to define and simulate different, varying environments. In addition, reinforcement learning and the immediate feedback from the (simulated) environment allow the agent to adapt policies to the environment requirements.

Park et al. propose an approach [191] that extends and combines the *Elo rating system (ERS)* [69] approach with the *explanatory item response theory (IRT)* [58] to solve the cold-start problem regarding new students within e-learning platforms. In doing so, the approach takes into account students' profile information and their success in previous learning sessions to predict ability levels and thus provide adapted learning session recommendations. The aim is to provide learning lessons that correspond to the learners' abilities. However, unlike the proposed agent framework, the approach considered is only tailored to e-learning environments and requires pre-existing data that other students have already generated through their learning sessions, as similarities between students are also identified in order to make recommendation predictions for new students. The proposed agent framework is not limited to recommendation activities but can simulate different kind of tasks.

In their work [23], Bharadhwaj et al. propose a "model-agnostic meta-learning" approach in which "a recommendation model is pre-trained off-line for a wide range of users". The pre-trained model is then adapted for new users using gradient descent. The authors claim that their approach would be applicable to all types of recommendation tasks. However, since they use deep neural networks to train the user recommendation model, they need data on user ratings, i.e. user feedback, and items. However, this issue points out that the user cold-start problem is not really eliminated, as no generalisable model can be trained without existing data and user interactions. The agent framework circumvents this problem with semantic task definitions that adhere to task-specific guidelines provided by domain experts. The required states and feedback is then generated by the simulation component based on the information contained in the task descriptions.

In the paper of Hawashin et al. various multi-label classifiers such as Bayes classifier, deep neural networks and J48 are used to predict recommendation items based on user interests [96]. A prerequisite for this approach to work is that the user and his or her interests are known in advance through user interactions, datasets, and a domain model, whereby the approach can only be applied domain-specifically. In the agent framework however, no information about the user is required at the beginning, as the agent learns a general default strategy to subsequently perform personalisation based on user-specific and environmental feedback.

3.1.2. Robot Systems

Robot systems are confronted with heterogeneous as well as dynamically changing environments and tasks of varying complexity, depending on the field of application. The

deployment of robots in new unknown environments consequently leads to the cold-start problem as the following related works show.

For instance, Yang et al. face a different kind of cold-start problem in the field of nano/colloidal robotics. The authors apply "model-free deep reinforcement learning" [284] to teach colloidal robots to avoid obstacles and navigate in unknown environments. The presented approach is based on the idea of considering only immediate state representations, i.e. binary images, indicating local, i.e. "proxy" obstacles relative to the robot's position. In doing so, the robot has to decide between two actions, i.e. ON and OFF, which control the robot's "self-propulsion". The authors claim that MDPs cannot adequately solve the problem of navigation in stochastic or unknown environments that cannot be modelled in advance. However, in the approach of this dissertation, it is assumed that the immediate presence of obstacles can be modelled as a binary state in an MDP. Furthermore, a simulation framework for agents could generate generative MDPs through generative ML approaches (e.g. GANs, variational auto-encoders) to simulate even rare environments with random obstacles, as proposed in Sec. 7.2 of this thesis. This has the advantage of creating different and even unlikely environments that the agents can train on the basis of limited information, which increases the generalisability of the trained models. Unlike the proposed agent framework, the considered related work is domain-specific and requires datasets in order to work.

In the paper [182] of Niroui et al., mobile rescue robots explore "unknown, cluttered environments" by combining Deep Reinforcement Learning (DLR) with *occupancy grid maps* that model the relevant environment to find "boundary locations." In their proposed architecture, a model called *world model* enables the creation of a 2-D occupancy grid map that uses sensor data, i.e. odometry data and 3-D depth measurements, to represent the boundaries of the relevant environment. Similar to the proposed agent framework, the architecture consists of different components or modules responsible for different process steps. The state representation of the *world model* is kept very simple by indicating whether a grid cell is "open, occupied or unknown". The occupancy grids serve as the input state for the deep neural network which is a *Asynchronous Advantage Actor Critic (A3C)* network used in the exploration module. To train the network, the approach uses a robot simulation framework, i.e. the *stage simulator* of ROS[1] that represents the simulated environment via definitions within a *world file*. The presented approach is very specific to robot navigation and thus has its focus on exploring unknown environments. In contrast to this approach, the agent framework proposed here considers the model representation of heterogeneous domains, tasks and environments in a higher-level, i.e. human- and machine-understandable and generalisable, way by providing a semantic representation of MDP tasks that serves as a guideline for simulating possible states and responses.

In the approach of Hu et al., "dynamic Voronoi partitioning" is used together with "model-free, actor-critic deep reinforcement learning" for "collision avoidance" to enable multi-robot agents to collaboratively explore unknown environments [109]. The specific goal is

[1] `http://wiki.ros.org/turtlebot_stage`

to coordinate robots in different regions of the environment so that the exploration task is efficiently correlated and assigned to the appropriate mobile robots. The training of collision avoidance and thus of the deep neural network is done through demonstration learning. Demonstration learning requires an infrastructure that allows the agents to be taught their tasks. This can be a costly, lengthy and error-prone process, as every possible state has to be played out to provide relevant and sufficient information to the learning algorithms. Although demonstration and imitation learning is useful in robotics, there are still domains and tasks that do not have the capabilities to provide resources and infrastructure for demonstrations. To close this gap, the proposed agent framework allows the definition of semantically represented MDPs and rules both by log datasets, e.g. obtained from demonstration learning, and by domain experts specifying relevant information such as states that can occur and be simulated by the simulation component. Moreover, the automated collaboration process, i.e. sharing and reusing of trained policies, via the proposed agent framework, enables overcoming the cold-start problem.

3.1.3. Human Computer Interaction Systems

Human computer interaction (HCI) systems are similar to recommender systems as they also have to incorporate continuously new users with different characteristics and behaviour. Thus, it can be stated that recommender systems are a subset of HCI systems. The following related works discuss corresponding related works that have other purposes than recommender systems.

Denaux et al. attempt to solve the cold-start problem for web learning systems with OWL-OLM, a framework that builds user models based on domain ontologies [59]. Therefore, a software agent interacts with the corresponding user to conceptualise user models in a dialogue-driven way. The dialogue with the user is aligned with the underlying ontology, which depends on the corresponding domain and serves as a default model for the dialogue flow. In this way, users of web platforms can be guided by a dialogue agent based on their user profile and capabilities. Similar to the proposed agent framework, Denaux et al. use ontologies for the purpose of personalisation and to avoid the cold-start problem, whereby domain ontologies specify how the process flow is performed by an agent. However, in the related paper, the warm start is limited to dialogue control in web-based learning platforms, while the focus of this thesis is on avoiding the cold-start problem in heterogeneous domains and agent-driven tasks.

The approach proposed by Banovic et al. aims to allow a warm start for "general-purpose smart agents" that conduct personalised services by, on the one hand, predicting informative questions that the agent should ask the user and, on the other hand, automatically inferring information about the user from the answered questions and their underlying variables [11]. In doing so, their goal is to consider only a subset of user variables that allow inferences about the corresponding users of agent services. Although the considered work is applicable domain-independently, its proposed approach to user variable selection requires the use of existing and labelled datasets containing different variables about the

target users. A drawback of the considered approach is that it is not always possible to find suitable datasets about user groups across different domains and tasks, as they are not always available for various reasons. Even if the proposed approach allows for inferences about users, the problem remains how an agent can solve tasks individually and context-dependently from the beginning of its execution. Inferences about the user are not always sufficient in this case, and an agent needs to know how to act in a goal-oriented way at the beginning of its runtime in order to adapt to user preferences and contexts in the course of its execution.

Zhang et al. propose in their paper [289] to use "cognitive models to pre-train RL agents" for avoiding the cold-start problem in human-computer-interaction (HCI). By cognitive models the authors aim at simulating "average" user behaviour in "complex task environments". In contrast the proposed agent framework of this thesis is not limited to human-centred models and simulations but also considers all kinds of states in general. The abstract and generic character of the proposed agent framework ontology allows to simulate different kind of environments and tasks that are not limited to human responses and behaviour. The approach of Zhang et al. covers only a subset of tasks, i.e. only HCI-related tasks. In contrast, the agent framework also supports tasks beyond human-machine interaction. Moreover, the HCI models used only represent average user behaviour without considering RL agent interactions on user interaction. The authors state that only validated cognitive models that include further assumptions and parameters about the HCI task should be used. The agent framework, on the other hand, takes into account the effects that an agent has on its environment and interaction partners.

3.1.4. Summary

Considering the related work, it turns out that the related approaches are essentially found in recommender systems (Sec. 3.1.1), mobile robotics (Sec. 3.1.2) and dialogue systems (Sec. 3.1.3), where new, unknown users, environments or processes may occur that the approach should be able to deal with. With the exception of two approaches, almost all are limited to a specific domain or even a specific task and cannot be applied across domains.

Most of the approaches considered require datasets in advance, which means that interactions with users or user interventions are necessary to generate data. From this perspective, the cold-start problem is not really solved by the related work, as a cold-start ultimately means that no data is available and prior knowledge has to be provided to the system or agent in some other way. Some of the approaches, including the proposed agent framework, use a domain model and/or knowledge [204, 5, 234, 62, 114, 191, 96, 182, 59, 11, 289] provided by domain experts [182, 109, 289]. However, the corresponding related approaches are not generic, but relate only to a specific task or domain. Agents that solve different tasks across domains are not envisaged in these approaches. Among the related works considered, there are no approaches or agent frameworks that deal with the cross-domain cold-start of software agents. This gap is filled by contribution C1.1 of the agent framework presented in this thesis. While the lightweight ontology provides a generic

representation, i.e. concepts and properties, of MDPs that can be used for simulation, process datasets can also be processed to derive task entities in an automated way. The prior knowledge required to overcome the cold-start problem can be extended over time by both domain experts and agents in a semi-automated way, learning and exchanging their policies in a well-defined procedure.

The novelty of this approach lies in the fact that agents are provided with a framework that maps their life cycle and supports the agents in the individual phases (integration, training, application and provision of strategies). When an agent follows the predefined process, it can autonomously learn strategies and then adapt them depending on the context and feedback from the environment, and the simulation component prepares the agent for the corresponding task.

Table 3.1 summarises the studied properties of the considered related approaches. The table lists whether a) the related work is domain-specific or applicable across domains, b) requires datasets, c) requires a domain model, e.g. an ontology and/or policies, d) requires user interactions with the proposed approaches, e) requires domain experts to provide knowledge about the domain or task in advance, and f) what the approach specifically does and what techniques it uses.

Paper	Domain-specific vs. Cross-domain	Datasets required	Guidelines/ Domain-Model required	User Interaction/ Intervention required	Domain Expert required	Approach
Proposed agent framework	Cross-domain agent tasks	No	Yes	No	Yes	Semantic framework ontology simulation component reinforcement learning agent(s)
[249]	Domain-specific recommendations	Yes	No	Yes	No	Data Acquisition Data Transformation
[204]	Domain-specific Software Crowd-sourcing	Yes	Yes	Yes	No	Reinforcement Learning + Clustering
[196]	Domain-specific User-specific item recommendations	Yes	No	Yes	No	Opinion mining/classification from human readable text Collaborative filtering
[5]	Domain-specific Item recommendations	Yes	Yes	Yes	No	Domain Ontologies content-based and collaborative filtering
[234]	Domain-specific User-specific item recommendations	Yes	Yes	Yes	No	Ordered weighted averaging operator Matrix factorisation, collaborative, content-based, demographic filtering
[62]	Domain-specific Unsupervised semantic recommender	Yes	Yes	Yes indirectly (social media)	No	Wikipedia documents and social media data for finding user interests
[114]	Domain-specific Semantic recommendation system for e-learning	Yes	Yes	Yes	No	Ontology + K-Means clustering, collaborative filtering,
[191]	Domain-specific e-learning platform recommendations	Yes	Yes	Yes	No	Combination of Elo rating system + explanatory item response theory
[23]	Cross-domain recommender system	Yes	No	Yes	No	Model-agnostic meta learning gradient descent
[96]	Domain-specific Prediction of item recommendations	Yes	Yes	Yes	No	Bayes classifier, Deep neural network J48 classifier
[284]	Domain-specific Colloidal robots in unknown environments	Yes	No	No	No	Model-free deep reinforcement learning
[182]	Domain-specific Mobile rescue robots in cluttered environments	No	Yes	No	Yes	Depp reinforcement learning Occupancy grid maps World model Robot simulation framework
[109]	Domain-specific Exploration of unknown environments by multi-robot agents Collision avoidance	No	No	Yes Demonstration Learning	Yes	Dynamic Voronoi partitioning Model-free actor critic deep reinforcement learning
[59]	Domain-specific Dialog control in web-based learning platforms	No	Yes	Yes	No	Domain ontology + Interacting agent for building user models
[11]	Cross-domain user interaction	Yes	Yes	Yes	No	Prediction of informative questions Derivation of informative user variables
[289]	Domain-specific Human computer interaction	No	Yes	Yes	Yes	Cognitive models for pretraining RL agents Simulating user behaviour

Table 3.1.: Comparison of related works and agent framework w.r.t. the cold-start problem.

3.2. Context-Aware Policies Adaptation and Combination

Policies adaptation and combination is necessary for the personalisation of tasks and provided agent services since default strategies trained are sometimes not sufficient for providing adequate services to target users. For this reason, the proposed agent framework addresses personalisation respectively customisation by approaches (e.g. reinforcement learning, decision rule mining) that contribute to the adaptation of default policies towards specific policies. Default policies may be sufficient at the beginning of an agent's life-cycle. However, as soon as the environment changes or the agent obtains user feedback the agent requires to take user preferences and context changes as well as varying environments into account. In the following three sections, i.e. Sec. 3.2.1-3.2.3, related work on reinforcement learning, language models as well as rule mining approaches are considered and compared with the proposed agent framework of this thesis, since the agent framework supports the adaptation and combination of policies based on context information and feedback. In this context, the contributions of this work are the following ones:

C2.1.1 A methodology that combines reinforcement learning and rule mining techniques in order to adapt policies and derive adapted policy rules.

C2.1.2 A methodology for the selection of suitable rule mining algorithms that is based on the corresponding task description.

C2.1.3 An algorithm for data preprocessing so that the applied approaches can process and mine the given datasets.

C2.1.4 An approach, i.e. *Best Fit Sequence*, for deriving rules for sequential tasks that are divided in stages.

C2.2.1 A methodology for representing and transforming task entity descriptions into a processable representation that can be used to calculate similarities between task, state and action entities and to represent and delimit task contexts.

C2.2.2 A methodology for the context-dependent composition of policies that address tasks and activities.

The columns of the Tables 3.2 and 3.3 represent the criteria, such as *a) intended use, b) domain independence, c) adaptation of policies, d) used algorithms, e) input- and f) output data* that were considered in the related works and proposed agent framework.

3.2.1. Reinforcement Learning Approaches

Reinforcement Learning (RL), especially deep-q-learning [269, 94, 98], is applied in many different application fields (see e.g. [280, 293, 171, 170, 230, 61, 168, 79, 63, 143]) concerned with computational agents [198], MDPs [40, 190, 269] and decision making. For instance, den Hengst et al. note that reinforcement learning is increasingly being used in "human interaction systems" for personalisation purposes. They remark in their paper [97]

that reinforcement learning is a "robust" paradigm "to apply in contexts that involve humans" and provide an overview of numerous works that use reinforcement learning for personalisation in human-machine interaction. The review of den Hengst shows the relevance of personalisation in heterogeneous domains such as healthcare, web services and recommender systems. In their survey, the authors find that general applications of reinforcement learning are used, distinguishing between general "task-related" adaptation and "user-specific" adaptation. The proposed agent framework addresses both types of adaptation by representing, i.e. generalising, and processing both user and task characteristics in state entities. However, a drawback of reinforcement learning might be that training reward-maximizing strategies could require lengthy training procedures, depending on the complexity of the tasks. In particular, the state and action space of a task determines the duration of learning appropriate strategies that enable successful task execution.

Da Silva et al. investigate *inter-agent teaching* respectively *transfer learning* that is applied for increasing the learning speed of RL agents [229]. The paper points out that considered teaching approaches between agents are restricted regarding their ability to generalise to different domains and application scenarios. Their approach is domain-specific and allows adaptation only through demonstrations and transfer learning. However, the complexity of today's world, as well as the advancing digitalisation, makes it necessary that software agents can be used more and more universally. For this reason, a generalisation methodology is needed that allows agents to operate in different environments, for different domains and tasks, especially when their services are needed across domains, which might be the case in IoT environments and service robotics, for example. For this reason, the agent framework is intended to support cross-domain applications that can be adapted on demand to new environments, contexts, tasks and users.

The following related works show various applications in which reinforcement learning is used. For instance, Liu et al. propose in their work [142] a data-driven deep-q-reinforcement learning framework for learning policies of sequential treatment plans for patients. The approach combines supervised learning and RL for composing sequences of dynamic treatment plans. Similar to the approach of the agent framework, their objective is to reduce the state and action space and derive depending on the patient's context (i.e. patient data, vital parameters) suitable treatment plans. However, their approach is domain-specific and requires medical records of patients in order to adapt treatment paths. The considered approach thus also requires supervised learning through processable datasets. In contrast, the proposed agent framework does not necessarily require annotated data, since domain experts have the option to specify in the framework guidelines, in cases where annotated data is sparse or not available.

Kanervisto et al. discuss how manipulating (e.g. remove or discretise actions) the action space in RL problems, can influence positively the performance of RL applications (e.g. in video games) [116]. The proposed agent framework proposes an alternative approach that allows to constrain the state and action space and relate states and actions semantically through entity embeddings and to predict the most reward maximising actions depending on the agent's state and context. This also allows to predict which are the most likely

states and actions that are relevant within a context. Furthermore, it seems that Karnevisto et al. only tested their approach with video games and it is questionable whether their approach would also apply for real-world use cases and contexts. One aim of the proposed agent framework is not to be limited to video games, but to support real-world applications outside of video games.

The approach [92] proposed by Hanawal et al. uses state-action samples representing intended policies in order to learn generalisable policies by means of L_1 *regularised logistic regression*. However, if annotated data are not available or environments and conditions change, the approach may not be applicable, unlike the proposed agent framework.

Table 3.2 summarises the different properties of the considered related works and the agent framework. The table shows that many approaches apply reinforcement learning for adaptation purposes. In particular, simulation and demonstration or imitation learning play a crucial role in training reinforcement learning agents. Furthermore, the generalisability of strategies is important in most approaches considered, as the dynamics of different environments and tasks as well as user preferences have to be taken into account. In contrast to the considered related works, the agent framework provides a simulation component that uses expert knowledge in the form of task descriptions as well as datasets to reproduce and simulate MDP tasks and activities for RL agents. Due to the formal representation of tasks, the simulation component of the agent framework is able to simulate heterogeneous tasks, not being limited to specific domains and applications, while the considered related works are mostly bound to one specific task and domain.

Independent of the considered comparison criteria in Table 3.2, one challenge in RL is the complexity of tasks that impact how fast RL agents explore and learn useful policies. In particular, the state space and action space determine how quickly and successfully strategies are trained by reinforcement learning. Therefore, many approaches concerned with RL look for ways to reduce the state and action space so that agents arrive at solutions faster. The agent framework also implements an approach that aims to reduce an agent's state and action space to immediately assemble, on demand, context-dependent policies that allow agents to solve context-specific tasks on demand, without having to learn policies from scratch, unlike in reinforcement learning. While the presented agent framework applies reinforcement learning for training policies, it is not limited to RL but also supports other approaches (e.g. rule mining) to quickly find policies by using knowledge gained through experience, i.e. log records, and abstracted context knowledge (e.g. entity embeddings).

3.2.2. Language Models for Policies Combination

Language Models are increasingly used for the transformation of natural language descriptions into program instructions for software agents. For instance, the authors Puig et al. propose an approach that *"translates natural language instructions of activities into programs"* [202]. In order to do so, their approach uses two *recurrent neural networks (RNNs)*, i.e. *Long Short-Term Memory (LSTM)* [104] networks, as encoder and decoder of

Paper	Intended Use	Domain Independence	Adaptation	Used Algorithms	Input Data	Output Data
Proposed Agent Framework	Adaptation of default policies	Yes	Yes	Deep-q-Reinforcement Learning	Simulation of environmental feedback (rewards, continuous and categorical data)	Adapted q-learning network weights
[229]	Inter-agent teaching Acceleration of policy learning	No	No	Transfer learning	Agent demonstrations	Policies
[142]	Learning of dynamic treatment plans	No	Yes	Deep-q-reinforcement learning supervised learning	Datasets e.g. patient data, sensed vital parameters	Sequential treatment plans
[116]	Performance Increase of RL applications	Yes	Yes	Reinforcement learning Manipulation of Action space	Action space	Policies
[92]	Learning generalisable policies	Yes	Yes	Sampling of state-action space L1 regularised logistic regression	Sampled State-action space	Intended policies

Table 3.2.: Comparison of reinforcement learning approaches

the input instructions. The words of the natural language instructions are represented through *Word2Vec* [164, 163] embedding vectors. Furthermore, they apply reinforcement learning, i.e. *policy gradient optimisation* [207], to train policies for the execution of the encoded activities. The policies are trained based on rewards that are computed through the *longest common subsequence* and the feasibility of the executed activities within the *Virtual Home*[2] simulator.

In their paper Liao et al. propose *ResActGraph*, a model that *"extracts programs directly from natural language descriptions"* [138] in order to support virtual agents to conduct activities within *Virtual Home* environments. The related approach allows to generate activity programs with action sequences based on predicted activity sketches. Their objective is to extract programs that are generalisable to new environments since environments may vary and lead to tasks or activities to be adapted. For this reason a context-aware adaptation of activities is desired. According to Liao et al. *"a common sense knowledge about the world and how to deal with typical situations"* [138] is required. For their approach they use *Graph Neural Networks (GNNs)* that generate activity programs based on changes in the environment. For predicting activity programs, activity and environment sketches serve as input [138] for the *ResActGraph* model. However, previously sketches have to be generated from natural language descriptions or demonstration videos of domestic activities.

Li et al. propose a framework called *LID "that uses pre-trained language models for sequential decision making"* [136]. The authors use pre-trained language models to predict action sequences based on encoded observation and goal embeddings that enable the completion of heterogeneous tasks in different environments. The approach implements a *"policy network with pre-trained language models"* that predicts the most likely next action

[2] `http://virtual-home.org`

during task execution [136]. In addition, they use *"expert demonstrations"* to adapt their pre-trained language models. Their approach also includes a method called *"Active Data Gathering (ADG)"* [136], i.e. agents interact with their environment to optimise themselves in terms of combining strategies and improving their task completion rate.

The considered baseline approaches are compared with the proposed approach of this work in terms of their completion rate on seen and unseen tasks respectively activities. In contrast to the related baseline approaches, the implementation of the proposed policy combination approach is simple and only requires task entities that can be encoded as entity embeddings. This makes the implementation simpler by eliminating one layer of complexity, namely natural language preprocessing, which complicates the task of combining policies according to an agent's context. Language models nevertheless make sense if natural language descriptions by domain experts are intended to serve as instructions for action by software agents. The policy combination approach is different in this sense. It uses knowledge graphs with semantic task entities created by domain experts or derived from time series data as input to combine policies. The advantage is that irrelevant information can be omitted from the tasks in their representation. Furthermore, the ambiguity of language in the considered works may lead to worse task completion rates and generalisation than in the proposed policy combination approach.

3.2.3. Rule Mining and Process Mining

Various methods and approaches are proposed to derive meaningful decision rules from large log datasets that reflect (user) behaviour and processes. Since the agent framework also applies rule mining approaches to derive policies from log datasets that emulate MDPs, the following related work provides insight into how different rule mining approaches are used to derive decision rules for providing context- and process-oriented policies and to reproduce goal-directed behaviour in heterogeneous domains.

In the paper of Norambuena, the author outlines the need to integrate process mining with simulation tools and discusses the challenges that arise when process mining has to "complement" the simulation of processes [183]. The particular challenges that may arise are the lack of process knowledge in process-agnostic systems, unstructured, noisy data and the "level of abstraction" that should be balanced in terms of *complexity, precision* and *representation* of the data. In addition, the paper identifies the evaluation, optimisation and usability of processing-mining results as challenges that need to be addressed by appropriate methods and frameworks. The proposed agent framework addresses some of the mentioned challenges by providing a predefined structure and the provision of datasets and linked knowledge through ontology concepts. Furthermore, the complexity is reduced and abstracted by the MDP model representation used in the agent framework, while the optimisation of standard knowledge is automated by reinforcement learning agents that adapt existing policies to current requirements. The evaluation is done by taking into account environmental feedback, i.e. reward signals that help to capture the quality of actions and activities.

It turns out that many decision rule mining algorithms are hybrid approaches that combine common data mining techniques and aim to greatly reduce the amount of data in order to derive significant rules that provide valuable insights into the underlying data. Most algorithms work on the basis of association rule mining and use frequent item sets, employing metrics (e.g. support, confidence, lift) and statistics (e.g. Bayes). For instance, Letham et al. present *Bayesian rule lists* [135], a statistical approach that does Bayesian analysis on frequent item sets. While the approach is applicable domain-independently, it does not allow for policy adaptation as it is required in the proposed agent framework.

Bemthuis et al. present an architecture based on event logs, process mining and agent-based modelling for mining inherent processes that reproduce supply chains [18]. The considered approach has the limitation that it is not generalisable and, according to the authors, requires further modelling efforts to enable its application in heterogeneous environments. They also mention that the autonomous behaviour of the corresponding agents could be enhanced and improved by "self-learning" strategies and "automated feedback loops". In summary, the considered related approach is neither domain independent nor does it support adaptation of policies. The proposed agent framework addresses the mentioned requirements through the simulation component, the framework ontology, rule mining and reinforcement learning. The framework ontology provides generalisation of task processes so that it can be used across domains and processes. Furthermore, the proposed agent framework enables agents to contribute new task-specific process data that can be used to obtain different task strategies while agents are enabled to learn and adapt autonomously by RL and rule mining their policies.

Tour et al. propose an "*agent system mining*" framework that applies business process mining techniques together with *agent-based modelling and simulation* on the "*micro level*" of business processes [256]. According to the authors, in this way complex, related business processes can be mined and modelled, both at macro and micro level. However, the authors mention that "meta models" and "formalisms" are required for representing multi agent systems. Furthermore, the considered approach is neither domain-independent applicable nor does it allow the adaptation of rules. The proposed agent framework tries to bridge this gap by the framework ontology and corresponding decision rules that capture both the static and dynamic properties of task processes. In addition, the agent framework defines a methodology for the appropriate selection of rule mining algorithms based on task characteristics to derive policies for the underlying MDPs from log data.

Roldan et al. propose an approach for process mining that allows the building of process models for analysing multi-robot tasks [209]. The proposed process steps of the approach consist of the preprocessing of event logs, the mining of process models through *Petri Nets* respectively *Inductive Miner* [132] and the enhancement and adaptation of the mined models. The utilised Petri nets are a means for the visualisation of process models, that is intended for human recipients. Thus, the approach is tailored towards human experts who would like to analyse log data in order to comprehend processes and optimise interactions and utilised resources. In particular, model evaluation and model enhancement require the intervention of human experts and the authors claim that in future works they will address

these topics. The agent framework, on the other hand, aims to automate the majority of agent tasks so that agents can independently learn, extend and share task policies without user intervention.

The paper of Polyvyanyy et al. suggests a "probabilistic goal recognition" approach that allows the prediction of the most relevant goal among different probable (contradicting) goals [197]. The novelty of this approach is that it does not assume any prior knowledge or model about the process under consideration. The most relevant goal is derived from behaviour log data representing the real-world event or activity traces of different agents. The authors implicitly assume that agents have to choose between heterogeneous goals while agents in the proposed agent framework try to achieve each sub-goal (i.e. each stage) required to complete a task. Therefore, agents in the proposed agent framework do not have to choose between different goals, but only have to choose the right sequence of actions leading to the sub-goals, since in the conceptual idea of the agent framework, each sub-goal and stage of the MDP contributes to the solution of the performed task. Furthermore, the related approach considered only supports stationary environments in which activity traces do not change. This means that, in contrast to the agent framework, the adaptation of goals and activities to changing environments is not addressed.

Mesabbah et al. propose an "auto-simulation model builder framework" that utilises clinical event logs in order to generate simulation models of discrete events in clinical pathway processes[161]. The framework consists of two components, one for applying process mining and one for data analysis. The output of the framework are simulation models of different scenarios providing different KPI[3] estimates. Similar to the agent framework, the considered auto-simulation builder framework uses datasets, i.e. event logs, in order to derive a process model that can be utilised for simulation. However, while the work under consideration aims at providing simulation scenarios to stakeholders, simulation in the agent framework is intended for software agents to learn to perform tasks autonomously. In this context, the development of policies that promise a good or even optimal performance are one of the main objectives of the agent framework while in the considered related work, providing business stakeholders insights about their processes is of interest.

In addition, Mesabbah et al. provide an extension [160] of their former approach [161] that combines data mining, process mining and machine learning approaches for "complex decision making in resource handling" and performance prediction. In the considered paper, log data as well as patient records are processed in order to generate simulation models that help to capture underlying processes and predict future outcomes and the need of resources in the health care domain. The considered process mining framework is, unlike the agent framework, in particular intended for and restricted to the medical domain. Agents within the process mining framework are considered as individual patients that have to follow a clinical pathway. Identifying clinical pathways can be easily replicated in the proposed agent framework, while the considered related approach of Mesabbah et al.

[3] KPI is an abbreviation for Key Performance Indicator and is mostly used to quantify performance of economic activities.

is not applicable in domains other than the medical domain.

Ghazanfari et al. present a "*Sequential Association Rule Mining*" approach that maintains the "*Hierarchical Structure of Tasks in Reinforcement Learning*" [83]. The outcome of the approach considered, is "hierarchical policies" that capture Markov decision processes. The decision to derive rules, i.e. policies, instead of directly applying reinforcement learning (RL) is justified by the authors through the fact that RL can require time-consuming learning procedures until task suitable polices are trained. The authors assume that sub-goals are given in tasks that show a certain correlation and allow their hierarchical extraction by sequential association rule mining. In the proposed agent framework, sub-goals can also be defined to narrow down the state search space and contribute to the completion of a task. However, in the agent framework, the state space does not necessarily need to be constrained by frequent item sets and association rule mining, but is trained and represented by entity embeddings that already constrain the state and action space by their spatial distribution and distance. Based on the trained entity embeddings and provided state information, a novel approach is provided in the agent framework that enables the immediate composition of policies. However, within the agent framework, some rule mining approaches are also used that apply frequent item sets to constrain the state space. In these cases, the agent framework distinguishes which type of rule mining approach is applied depending on the task characteristics. The proposed agent framework therefore provides means (e.g. framework ontology) for the semantic specification of task characteristics that enable the automated selection of the appropriate rule mining approach(es).

The approach of Porhet et al. extracts behavioural rules of virtual patients by means of *sequence mining*, i.e. *ERMiner*, from training data of physicians' trainings so that virtual patients can provide appropriate feedback to the physician who is training the delivery of bad news to a patient [199]. Virtual patients can also be viewed as software agents that have to apply rules to reproduce contextual, real-world actions, such as giving suitable feedback. In the approach considered, it is assumed that rule sequences are unordered. In the context of the considered use case this may make sense, but in other contexts, rules or policies may be required that presuppose and represent an ordered sequence of states, i.e. premises, and actions. In contrast to the considered related work, the agent framework therefore, also addresses the composition of ordered policies (see Sec. 4.5.2), that take into account the effects of actions on states and their changes.

The approach of Leotta et al. constructs a visual representation, i.e. a model, of human behaviour by evaluating sensor events from sensor log data and mapping them to actions that represent human behaviour [133]. Therefore, the approach first filters and transforms sensor logs into event logs and then applies an "action recognition algorithm". The authors claim that "smart services" or "ambient intelligence" can be enabled in this way. However, it is not clear from the paper how the derived behaviour patterns can be reused and improved if, for example, deviant behaviour is identified. Furthermore, the authors point out that their approach can only consider discrete data for mining human activities. In contrast, the proposed agent framework avoids considering raw sensor data, but assumes abstracted information provided by IoT wrapper components. The IoT wrapper components therefore

need to transform raw sensor data into semantic observation features that conform to the ontology concepts of the proposed framework. This procedure allows to integrate any kind of sensor data, as long as an appropriate Wrapper component is provided that transforms the data into the required format. In addition, the agent framework, unlike the considered related work, also supports numerical data as input to the rule mining approaches used.

By Mannhardt et al., an *Guided Process Discovery (GDP)* approach is proposed that enables the "uplifting" of low-level events to explainable high-level activities [146]. GDP uses activity patterns and clusters obtained from low-level events based on the given activity patterns. The result of this approach are process models that reflect behavioural activities and enable insights about them. Similar to the proposed agent framework, the approach under consideration uses domain knowledge to describe low-level events and high-level activities. The agent framework introduces in its ontology the concepts *Task*, *State* and *Action*, which can be understood analogously to activity (i.e. task) and event (i.e. state and action). However, the agent framework does not only allow to describe sequential processes, but also empowers agents to find alternative strategies to fulfil an activity or task. While the considered related approach is rather meant to make processes comprehensible to domain experts, the agent framework is rather meant to support agents in finding reward-maximising strategies. The authors mention that their approach has some limitations for complex activities consisting of more than 250 low-level events. Furthermore, their approach is only as good as the manual activity patterns described by domain experts. In the agent framework this is addressed by the application of reinforcement learning, since it allows the adaptation, i.e. improvement of policies w.r.t. the environmental feedback.

In Mannhardt et al.'s approach, low-level events that occur in more than one activity cannot be unambiguously assigned to an activity if the cost or fitness function does not provide a distinction. Lastly, the authors mention that their approach requires high computational resources for abstracting complex activities. In the proposed agent framework there are no such restrictions given.

By Kovalchuk et al., a "hybrid approach" and "conceptual framework" combining "data-, process- and text mining" as well as "rule-based processing" is proposed for the development of simulation models that enable the computational simulation of health-care processes, patient-specific clinical pathways and treatment scenarios [126]. For this purpose the approach mines "electronic health records (EHR)" that reproduce individual processes that can be simulated. The paper outlines the necessity of model adaptation for *precision medicine* since according to Kovalchuk et al. individual patient characteristics have to be considered for comprehensive simulation purposes [126]. The considered paper addresses similar requirements and objectives as the proposed agent framework. However, the considered paper does not claim to be applicable across domains, although the authors state that their approach is generalisable to processes in the medical domain. This is due to the fact that EHRs serve as main input for the different mining algorithms. Thus, the approach is very specific to medical guidelines, policies and data sources and does not yet

support "diverse datasets".

Viktoratos et al. propose a "hybrid approach" that combines mining of association rules and contextual information with probabilistic approaches [267]. The approach shows the necessity of integrating different sources of information (e.g. context data) to derive decision or recommendation rules. Similar to the agent framework, the considered approach tries to overcome the cold-start problem and uses Semantic Web technologies for domain-specific knowledge representation. Here, the user context determines the best ranked recommendation rules for the most likely points of interest (POIs). In the proposed agent framework, a similar use case is given where policies need to be composed based on the agent's context. However, while the considered paper allows the derivation of individual recommendation rules, the agent framework allows the complete combination of policy recommendations that have to follow a specific order. Furthermore, the agent framework is not limited to the task of making recommendations about POIs. The representation of the context in the agent framework, unlike in the work under consideration, is kept generic by the MDP state concept and thus applicable to different tasks.

Sarker et al. extract association rules about user behaviour from smart-phone log data [218]. The focus of this work is on eliminating redundant association rules that reflect user behaviour. The goal is to enable intelligent and user-specific services that take into account and reproduce the user's needs. However, the considered approach only is applied and tested for smart-phone log data, i.e. is domain-dependent, and considers only categorical features for context representation. In order to apply this approach also for continuous features a data preprocessing step would be required that transforms continuous features into categorical features. In contrast, the proposed agent framework allows both the processing of categorical and continuous features.

Sarker et al. aim at addressing the derivation of behavioural rules from smart-phone data and enable the adaptation of applications and services [217]. The authors propose a five-stage "context-aware rule learning framework" that applies machine learning techniques to smart-phone data. According to the authors, the goal is to "build context-aware, rule-based intelligent systems" in an "automated" way. For the purpose of rule adaptation, they propose incremental learning, which enables contextual updating of rules by considering current contextual smart-phone data. The common features of the considered approach and the proposed agent framework are that both frameworks enable the derivation and adaptation of rules considering the environmental context. However, while the considered approach is only intended for smart-phone devices and uses corresponding data, the proposed agent framework allows the incorporation of diverse data from different domains and environments, which however have to adhere to the underlying ontology and data structure of the proposed framework. Moreover, the related paper presents a very high level view of the proposed architecture and it keeps unclear how their context model as well as integration of heterogeneous data, applications and services could look like.

Chen et al. use clinician activity logs from electronic medical records to learn clinical tasks and gain insights into their complexity [41]. For this purpose, tasks are derived

from session events, taking into account the duration of the activities. Although the approach derives tasks and their sequential process steps, the main focus is on revealing the complexity of clinical tasks in terms of execution time and repetition of events. The approach considered is only applicable to electronic health system protocols and thus in the medical domain. Generalisation to other domains is not envisaged, while the agent framework proposed in this work generalises the tasks so that the rules for each MDP task can be derived by corresponding datasets.

Chang et al. propose a methodology that uses "Predictive Association Rule Mining (CPAR)" to automatically instantiate entities of an ontology for an intelligent tutoring system and enrich it with mined association rules that define which tutoring actions, i.e. feedback, need to be performed or given by the tutoring agent based on the student's learning progress [38]. The proposed ontology schema for the tutoring system is a simple one with a few classes and properties that has not much expressiveness and can only be applied in tutoring systems that provide the corresponding real-world data with matching attributes. A further drawback of the approach is that the applied association rule mining algorithm requires threshold definitions for the metrics, such as *support* and *confidence*, used for the rule mining. Depending on the set thresholds the results of the algorithm may strongly vary regarding the accuracy and significance of mined decision rules. Moreover, only datasets consisting of categorical features can be mined by the algorithm for association rules. Continuous features have to be binned into categories and this has to be done by defining the bin size in advance.

3.2.4. Summary

Table 3.3 indicates that event log data in particular is used to derive either behavioural patterns or processes that need to be reproduced and analysed. The drawbacks of the related work considered include several aspects. For example, some of the approaches [18, 256, 209, 161, 160, 199, 133, 146, 126, 267, 218, 217, 41, 38] lack generalisability and extensibility to other domains. They are tailored to specific application scenarios and limited to a specific target. In addition, some of the approaches, i.e. in [135, 18, 256, 83, 41], do not provide "self-learning" strategies for agents and thus do not adapt to dynamic changes in the environment. Other works, i.e. in [18, 256, 209, 161, 160, 199, 133, 146, 126, 267, 218, 217, 41, 38], use log data as input to the rule mining algorithms but do not offer a formalisation of the input data or only a very sparse one, as they lack meta-models about the application domains in question. For this reason, the earlier mentioned approaches are not domain independent. Some of the approaches, i.e. in [18, 256, 161, 160, 126, 41], and their produced output data are not directed at agents at all, but serve domain experts to analyse processes.

In some approaches, i.e. in [133, 161, 217, 38], only discrete features are supported as input data, while continuous features are left out. The automated selection of rule mining approaches depending on the given features of a dataset is also not supported in the related work considered. The agent framework addresses the aforementioned shortcomings with

the contributions *C2.1.1 - C2.2.2* mentioned earlier. For instance, the agent framework allows to narrow down task contexts by entity embeddings, so that context-specific policies can be assembled on demand faster than e.g. in reinforcement learning. Furthermore, the semantic task descriptions enable to automatically select adequate rule mining algorithms.

In addition to the task properties, the proposed method also takes into account whether tasks are divided into phases or subtasks whose sequence is to be observed. In contrast to the proposed agent framework, the considered related works on decision rules and process mining do not include feedback, i.e. rewards, and do not support task stages and sub-goals and are mostly used in other contexts or applications than intended in the agent framework.

In summary, the problem that is not or only partially addressed in the considered related work is that there are agents (see e.g. service robotics, web agents, swarm agents, etc.) that operate in different contexts and need to respond to immediate context, environment and task changes. Therefore, it is important to avoid domain dependency and enable agents to deal with rapid context and environment changes and to react appropriately. For this purpose, different sensor values, i.e. discrete and continuous, need to be supported so that agents can make observations, derive rules and make decisions in all possible environments. Decision rules derived from collected time series data should enable agents to act and adapt their policies in different contexts and environments. For this reason, a formal representation, i.e. an ontology and entity embeddings, is needed that models and represents tasks and their relevant properties so that rules can be derived from acquired log datasets and policies can be combined based on context information.

Paper	Intended Use	Domain Independence	Adaptation	Used Algorithms	Input Data	Output Data
Proposed Agent Framework	Mining of decision rules that follow MDPs	Yes	Yes	FP-Growth OneR Classification Tree Sequential Covering Best Fit Sequence	Log Data i.e. environmental feedback continuous and categorical data	Decision rules as policies
[135]	Item rules	Yes	No	Bayesian rule lists	(Frequent) item sets	Rule lists
[18]	Deriving inherent processes of supply chains	No	No	Process mining + Agent-based modelling	Event log data	Processes of supply chains
[256]	Modelling of business processes at macro and micro level	No	No	Business process mining techniques agent-based modelling simulation	Real world event data	Business process models
[209]	Analysing multi-robot tasks by means of process models	No	Yes	Inductive Miner Petri Nets	Event log data	Process models
[197]	Agent goal reasoning	Yes	Yes	Process mining techniques	Behaviour log data agent behaviour traces	Skill models relevant goals
[161]	Simulation models of discrete events	No	Yes	Process mining data analysis	Clinical event log data	Varying simulation models with different scenarios and KPIs
[160]	Decision-making in resource handling	No	Yes	Process mining data analysis machine learning	Clinical event log data patient records	Simulation models for predictions of required resources
[83]	Deriving hierarchical structured policy rules for MDP tasks	Yes	No	Frequent item sets sequential association rule mining	State transactions of MDPs	Hierarchical policy rules
[199]	Behavioural rules of virtual patients	No	Yes	Sequence mining e.g. ERMiner	Data of physician trainings	Behavioural patient rules for simulation
[133]	Mining human behaviour model	No	Yes	Action recognition algorithm	Sensor log data Event log data	Human behaviour patterns
[146]	Mining high level activities from low-level data	No	Yes	Guided process discovery Clustering, Uplifting of events	Low-level events	Explainable behavioural activity patterns and process models
[126]	Computational simulation of health care processes	No	Yes	Data-, process-, text mining rule-processing	Electronic health records	Simulation models for health care processes
[267]	User-specific recommendations for Points of Interest	No	Yes	Association rule mining probabilistic approaches Semantic Web technologies	User ratings user context	User-specific recommendation rules
[218]	Detecting user specific needs for services	No	Yes	Association rule mining	Smartphone log data	Association rules of user behaviour user-specific services
[217]	Adaptation of user applications and services Context-aware rule learning	No	Yes	Classification and association rule mining incremental learning	Smartphone log data	Behavioural rules
[41]	Learning of clinical tasks and complexity estimation Optimising clinical workflows	No	No	Unsupervised machine learning Mann-Whitney U tests process mining	Audit logs from clinical medical records	Process steps of tasks complexity metrics
[38]	Recommendations of tutoring actions depending on the learning progress of the student	No	Yes	Predictive association rule mining	Ontology real-world data of tutoring systems	Association rules for tutoring actions

Table 3.3.: Comparison of rule mining approaches.

3.3. Retrieval and Reuse of Policies

The agent framework allows agents to share their knowledge, i.e. their policies with other agents who have similar tasks to perform, on the web. For this reason, finding matching task descriptions and corresponding solution strategies, i.e. policies, plays an important role in the agent framework. The assumption made is that tasks with similar properties also require similar solution strategies. Approaches for task retrieval and reuse of policies is therefore one of the contributions that are addressed and fulfilled by the proposed agent framework. The goal of the agent framework is to identify similarities between tasks via their context and at the same time to narrow down the state and action space so that agents can more quickly find a solution, i.e. composed policies, or similar tasks and adapt their related policies to their requirements. Below the novel contributions of this work are listed:

C3.1 A methodology that allows the mining of representative decision rules from behavioural, operational and task-specific data.

C3.2 A methodology that allows the formal specification and provision of provenance data and knowledge that can be utilised by agents to assess the suitability of policies.

Related work that is concerned with the mentioned contributions of this thesis, can be found especially in the field of *Case Based Reasoning (CBR)*, as this field is mainly concerned with finding similar cases that can be adapted and reused. Analogously, tasks can also be considered as a kind of cases. For this reason, many related works in this chapter deal with the retrieval of cases in CBR.

For instance, case representation, indexing of features and similarity metrics are crucial for the retrieval of similar cases and policies in CBR. Some CBR approaches utilise e.g. a *object oriented, hierarchical, contextual knowledge or frame-based representation* [20, 175, 216], *attribute-value pairs*, *fuzzy attributes or sets* [137] and *set of activities represented through temporal constraint networks* [46] as case representation to allow measuring the similarity between cases by different metrics such as e.g. Jaccard index [265], cosine similarity [141], Mahalanobis distance [4], nearest neighbours [81, 247]. All of these approaches have in common that they represent cases in an abstracted and formal representation in order to uncover similarities between cases. The implicit assumption is that similar cases may have similar solution strategies. Since CBR is a broad scientific field and the literature on it is abundant, the survey [51] by Cunningham et al. provides a good overview of common approaches to CBR.

The survey states that *structural aspects*, e.g. *hierarchical structuring, semantic networks, graphs, flow structures, bag of words, taxonomies etc.*, among feature values play a crucial role for case representations and thus in case retrieval [51]. The survey categorises different "similarity mechanisms", such as *direct, transformation based, information theoretic and emergent metrics*. The mentioned *direct metrics* compute distance or similarity measures in feature vector spaces or determine feature overlaps by "set-theoretic" approaches (e.g. *Jaccard index, Dice similarity*). According to Cunningham et al., other metrics compare

the probability distribution of features in order to determine case similarity, see *Kullback-Leibler divergence* or x^2 *statistics*. However, the survey points out that direct metrics, besides their efficiency and advantages, also have the problem of high dimensionality [51].

The survey argues that feature-value vector representations and direct metrics are the dominant representations and metrics in CBR. The survey also points out that the considered alternative approaches have their advantages in terms of certain aspects, such as higher accuracy or simplified case design. However, the alternative approaches can be "computationally expensive" and thus require specific retrieval strategies [51].

Cunningham's survey identifies different classes of similarity metrics and approaches used in case retrieval. Based on the classification of these metrics and approaches, related work was selected and considered.

Bergmann et al. represent cases in an object-oriented way by class hierarchies and attribute relations in order to derive similarities between case objects [20]. One assumption of the approach is that cases belonging to common classes are similar to each other. In the considered paper, different metrics are presented that vary by the semantics of different queries and rely on object-oriented relationships such as inheritance, aggregation, etc. An object-oriented view is to some extent equivalent to semantic entity representations within a knowledge graph. However, knowledge graphs are more expressive and powerful when it comes to reasoning about entity relations, as they inherently consider and model subsumptions, relations (e.g. transitive, (a)symmetric, reflexive) between object properties and constraints [105] in a natural way that is "*close to the human cognitive thinking*" [254]. Due to the above advantages, the proposed agent framework represents agents and tasks by semantically related entities that form a knowledge graph. In this way, relationships between task entities can be inferred and embeddings of entities can be trained for ML algorithms, while a knowledge graph can be easily extended with new entities. The advantage is that arbitrary complex tasks can be represented in an abstract and machine-interpretable way in order to enable agents to find semantically related tasks and policies. The fact that agents can autonomously find other similar tasks and policies through semantic task entities is novel compared to the approaches considered.

Shi et al. suggest a "model" to assess the similarity of cases [227]. The approach considers different feature types, i.e. numeric, symbolic and ordinal features, and applies different similarity measures depending on the type, such as Hamming distance. In addition, weighting coefficients that determine the importance of each feature are calculated for each feature by an adopted procedure called *G1*. Based on the determined feature similarities and the calculated overall similarity, the authors apply the *Nearest Neighbour* algorithm to a case base to compare and find for a problem similar cases. Depending on the cases' feature size, the complexity of computation operations increases and many conditions, that require domain knowledge from domain experts, have to be evaluated in order to apply the right metrics to the right features and retrieve similar cases.

Liao et al. propose a hybrid approach that applies fuzzy sets and crisp sets in combination to search for similar cases [137]. Therefore, domain experts need to assign linguistic variables and weights to the features under consideration to describe fuzzy sets of different types of features. The problem here is that human intervention/knowledge is required, while in the proposed agent framework, process data can be used to create and store semantic task entities that agents can access over the Internet to compute task similarities and find available strategies for reuse.

Xiong et al. propose a "fuzzy rule reasoning approach" to evaluate the similarity and utility of cases, where the similarity determines the utility [282]. Moreover, the considered approach learns the corresponding fuzzy rules from the cases in the case database using a genetic algorithm. Here, the cases are represented by their feature values and the referenced solution, while the problem requirements are also represented by feature values. Depending on the problem, either classification or regression tasks are performed. However, the considered approach may not be suitable for agent-based MDP tasks, where an environment provides stochastic feedback and is not necessarily based on datasets. Moreover, the authors' assumption that "similarity" is "equivalent" to "usefulness" might be problematic, as similarity of cases does not necessarily imply that the solutions of already existing cases are useful for solving a problem, as other cases with different characteristics might provide better strategies. For this reason, a feedback mechanism and a more meaningful formalisation of cases or tasks is needed to evaluate more similarity indicators than just general features. In addition, it requires to be evaluated whether the utility of a solution is actually given as assumed, before the solutions found, can be adopted and adapted. In the proposed agent framework, a simulation is therefore carried out afterwards to show whether and how the solution found fits and needs to be adapted. To achieve this, tasks are formalised as semantic graph entities and their property relationships to relevant entities are spatially mapped by embedding vectors. The mapping into an embedding vector space shows the semantic proximity of tasks and allows the use of the embeddings in ML algorithms due to the resulting numerical representation.

By Nunez et al. cases are considered and classified as "operational situations" that take place in an arbitrary environment [184]. From this point of view, the retrieval of similar cases is considered as a classification task, where cases in a case database are annotated with labels characterising "operational situations". Similarity between cases is determined by the proposed *L'Eixample* metric, which evaluates preprocessed feature vectors representing cases. In the preprocessing step, continuous features are assigned to discrete terms to obtain discrete categories. The relevance and weighting of case features is determined by the proposed *class-value distribution* approach, which takes into account the probabilistic distribution of features and their correlation. Also here, depending on the feature types, distinctions (i.e. heuristics) have to be made with regard to the calculation of similarities. The approach considered requires several elaborate intermediate steps involving *preprocessing of data, determination of relevant features, discretisation of continuous features* and *weight assignment*. The accuracy of the results obtained depends on the quality of the data that represent cases. The drawback here may be that complex case relationships cannot be modelled by a pure feature-based approach. For this reason, the agent frame-

work, considers no features for finding tasks, but semantic task entities that interlink e.g. actions and states into comparable entities. The spatial distribution of these entity encodings in the embeddings space allows the determination of similar tasks, states or actions.

Leake et al.'s approach estimates the adaptability of existing cases based on case similarity [131]. Therefore, the authors propose a "similarity assessment approach" that relies on adaptation experience. The approach is a *transformation based approach* where the costs are estimated and compared for adapting existing cases to a new given case. The required knowledge for adapting a case is also considered for their estimates. Therefore, the considered approach first detects unresolved problems in the new case and then looks up cases that solve the according problem class. One drawback of this approach might be that a case library about adaptation costs has to be maintained while the relevance of cases has to be estimated. Hence, it is not clear how *relevance* is clearly defined. The authors mention that they select the most relevant cases by "coarse-grained semantic similarity criteria" what sounds quite vague and needs to be further specified, since *relevance* depends also on objectives and can mean different things in different contexts.

3.3.1. Summary

From the related work considered, it can be observed that most retrieval approaches view cases as feature vectors whose similarity within a vector space is measured. The problem with this view, however, is that it does not take into account the relationships between cases. A knowledge graph representation is a more appropriate representation because knowledge graphs persist, reveal links and describe semantically relevant entities. Furthermore, knowledge graphs can be extended with additional knowledge and queried to obtain knowledge about task properties and application policies. The implementation of semantic knowledge graphs enables compliance with *FAIR*[4] Data principles that make data *findable, accessible, interoperable* and *reusable*. In terms of the proposed agent framework, software agents can benefit from these principles to solve complex tasks.

With the help of semantic graph entities, cases or tasks can be encapsulated, related, stored and retrieved more efficiently. If the entities are transferred, i.e. encoded, to an embedding space, the advantage is that these entities and their proximity to each other can be expressed numerically, visualised and thus better processed and understood. The mapping between numerical and semantic representation thus enables an explainability of determined similarities between cases because embeddings can be visualised by dimensionality reduction techniques. Moreover, embedding vectors can be efficiently referenced or retrieved by their semantic task entities and reused in machine learning algorithms as they preserve the properties of the semantic entities in their numerical representation. A positive side-effect is that similar task contexts can be represented as well by entity embeddings through their spatial distribution in the embedding space. In addition, the

[4] FAIR stands for Findable, Accessible, Interoperable, Reusable. See also `https://www.go-fair.org/fair-principles/`

contextual proximity of the embeddings enables an implicit delimitation of the state and action space, so that suitable strategies can be found and assembled more quickly than with other approaches (see Sec. 4.5.2).

It has to be mentioned that a significant difference is given between task entities and cases in CBR. While task entities in the context of this work represent problems with objectives that have to be solved by software agents, cases are already solutions for given problems. In the agent framework ultimately policies are considered as strategies or solutions to a certain task. In order to find suitable policies, tasks are evaluated in terms of similarity, since an assumption in this thesis is that similar problems require similar solutions that can be easily adapted by reinforcement learning and the simulation of the new task. The task knowledge graph representation is an alternative means that allows the retrieval of similar tasks through plain SPARQL queries that look up property and object intersections. In cases where no numerical representation is required for ML algorithms, SPARQL queries for task retrieval may be sufficient.

The review of related work on CBR has shown that some of the CBR retrieval approaches considered are only suitable for classification or prediction tasks. This is because they define the similarity of cases only on the basis of common feature values and do not take into account the extent to which semantic similarities or relationships exist. In MDP tasks that map a sequential process with goals, such CBR approaches might be not useful, since tasks, actions and environments of agents may vary and tasks may be more complex than just making predictions about class memberships of cases. It should also have become apparent that domain- or problem-specific knowledge is a prerequisite for most approaches to work at all. Therefore, the agent framework combines both knowledge graphs and entity embeddings to represent tasks and contexts in a way that is understandable to humans, agents and (ML) algorithms.

Further, the contributions *C3.1* and *C3.2* enable, on the one hand, the derivation of rules from behavioural data and, on the other hand, a way for agents to specify and assess meta-information about tasks. The novelty of these contributions is that agents are enabled to create and exchange policies independently, so that they can benefit from the knowledge and experience of other agents from other domains on the World Wide Web. This also means that agents are no longer bound to one application domain or task, but can share their strategies to solve heterogeneous tasks in different contexts and environments.

3.4. Summary of this Chapter

In this chapter, related works that address similar problems and apply comparable approaches and algorithms to the proposed agent framework were examined. In doing so, the contributions of this thesis were compared with the approaches of related works.

In Sec. 3.1 it was pointed out that many of the considered approaches that deal with the cold-start problem are fundamentally intended for recommender systems and are either limited to specific domains or tasks. In other words, some approaches [249, 204, 196, 5, 234, 62, 114, 191, 96, 284, 182, 109, 59, 289] lack generalisability as they are tailored to specific applications. Moreover, a significant number of approaches still rely on datasets that need to be collected before the approaches can address the cold-start problem. This implies that the cold-start problem has not been fully solved in its intended sense. The search for related work dealing with agent-based systems has shown that there are not many approaches that address the cold-start problem in agent-based systems. This gap is addressed by the proposed agent framework, which provides a formal and generic representation of MDP tasks, as well as a simulation component that uses this formal task representation for simulation purposes. In this way, software agents can train standard strategies and evaluate them before they are used in a real environment. In addition, the agent framework also supports the mining of datasets to derive semantic simulation task entities. This means that both the knowledge of domain experts and log datasets can be used to create formal task descriptions.

In Sec. 3.2, it was shown that several approaches also lack generalisability and extensibility, as they only address a specific objective and application domain. Enabling agents to autonomously adapt their policies is also not supported in many of the works considered. Depending on the complexity and structure of the tasks, some approaches [229, 142, 135, 18, 256, 83, 41] do not provide a formal representation or meta-models that enable the representation, adaptation and extension of domain knowledge. However, the most serious drawback may be that some approaches [133, 161, 217, 38] only support discrete features and require pre-processing or discretisation of data for continuous features. Depending on the feature type, it may also be necessary to use different rule mining algorithms. In contrast to the approaches considered, the agent framework selects the appropriate algorithms depending on the given feature types. In addition, the related works considered do not support a division of datasets into stages and sub-goals. However, process data can be structured to include and represent stages. In the approach presented, sequential stage goals and feedback from the environment are also considered and addressed.

It was outlined that some baseline approaches [202, 138, 136] use language models to represent contexts and predict the next actions to be performed based on these contexts. However, the baseline approaches considered require natural language processing and exhibit some degree of complexity and ambiguity, while the proposed approach on the agent framework refrains from transforming natural language text into agent instructions. In contrast to the related approaches, the agent framework uses task descriptions from process datasets or task knowledge graphs and in this way avoids the costly pre-processing

of natural language text and inherent ambiguities.

In Sec. 3.3, mainly CBR approaches have been studied because an important part in CBR systems is the search and reuse of similar cases and the agent framework also needs to perform the search and reuse of similar task strategies. The findings obtained from the related works show that many CBR approaches have a simplified case representation, which basically consists of feature-value vectors that are not meaningful when it comes to case relationships. Moreover, the CBR systems considered mostly deal with classification or prediction tasks where feature-value pairs may be sufficient. However, the agent framework deals with complex, environment-specific tasks involving entity relations that go beyond classification and prediction applications. This observation suggests the use of more meaningful representations such as knowledge graphs that encode property relationships between entities. These relationships contain important knowledge, e.g. about task similarity. The added advantage is that entities can be learned as entity embeddings that provide a numerical representation of the corresponding entities. This numerical representation can in turn be processed by ML algorithms and is used for similarity detection on the one hand and for context representation and constraining the state and action space on the other hand, which is useful for speeding up policy retrieval and composition. In addition, the agent framework also supports task retrieval through simple SPARQL queries that filter tasks with overlapping features and values. The SPARQL queries have the advantage that they are humanly understandable and indicate why certain tasks are considered similar. Depending on which features are important and most relevant, corresponding SPARQL queries can be defined that search for similar, i.e. overlapping, task features, which helps to explain task similarities.

4. Approach

In this chapter on the approach, the concepts and methods of the proposed agent framework are presented. The intended goal is to disclose how the elaborated concepts and framework components are brought together to build and apply the proposed agent framework. Furthermore, this chapter shows how the framework's provided functionality addresses the individual RQs in order to tackle the previously discussed challenges and problems.

Section 4.1 gives a general overview of the framework's architecture and its inherent software components. The individual phases of the framework are highlighted and related to the life cycles of the intended software agents. In addition, different user groups with different goals can use the agent framework. For this reason, this section identifies and describes intended user roles of the agent framework.

Section 4.2 discusses the underlying knowledge representation, i.e. the ontology, of the proposed agent framework that is required for knowledge exchange and the simulation of tasks. Furthermore, the representation and use of configurable mathematical models (e.g. differential equations) to simulate real-world phenomena in order to estimate the effects of certain courses of action is essential for simulation tasks, as it is often necessary to gain insight into the short- and long-term effects of actions and the corresponding evolution of phenomena (e.g. a pandemic). Therefore, domain experts require means to describe natural phenomena and (task) processes, which are provided by the framework and enable the simulator to simulate the phenomena and processes for software agents. For this reason, the proposed task simulation framework also addresses the requirement to apply rules and mathematical models that indicate how phenomena can evolve over time (see Section 4.2.1).

Furthermore, the different components as well as the agents need the knowledge for processing information in order to make intentional and contextual decisions that lead to subsequent goal-oriented actions and policies. The objective is to create a common and shared understanding within the framework that enables agents to learn and solve tasks by using the framework. In particular, the share and reuse of newly generated knowledge (i.e. policies, rules) plays an important role because different implementations of agents should be enabled to share their strategies with each other for a beneficial cooperation via the framework and shared knowledge graph, see Section 4.2.2.

Section 4.3 addresses the *cold-start* problem **(see RQ1)** by means of the simulation component, which allows agents, e.g. reinforcement learning agents, to train strategies (i.e. policies) from scratch that can be adapted and shared with other agents. This pre-training step is necessary because, at the time of deployment, the agents in question would have

no knowledge of how to solve an assigned task. They have to train previously appropriate strategies to successfully solve the task by perceiving state changes and feedback (i.e. rewards) from the simulation component. The simulation component, in turn, requires the generic task representation to update states based on the actions performed by the agent. In addition, the simulation component needs to know what feedback (i.e. rewards) to send to the agent(s) involved. Furthermore, the agent framework aims at simulating long- and short-term effects that depend on executed actions in order to provide beforehand information about the course of tasks. The corresponding section describes in detail how the simulation component works and communicates with the concerned agents.

Target users who interact or operate in real-world environments produce valuable data that can be used to adapt and improve strategies (i.e. policies), as the underlying data reproduce the task processes and characteristics as well as the behaviour and strategy performed by the user. This assumption makes it necessary to derive representative and meaningful rules from the produced log data **(see RQ3)**. Section 4.4 illustrates the devised concepts that allow the agent framework to derive representative rules, which in turn can be reused by rule-based agents without having to learn new strategies from scratch.

The proposed agent framework aims at providing a platform for software agents to share and reuse their knowledge. For this reason, agents who are deployed to participate in the proposed framework have to adhere to certain rules or guidelines when contributing new solution strategies. Section 4.5 discusses an approach of the framework that enables agents the retrieval and combination of suitable policies for their own tasks.

Furthermore, the agent framework aims at supporting different agent implementations that are able to autonomously adapt their behaviour depending on their current context **(see RQ2)**. Therefore, a number of algorithm implementations (e.g. reinforcement learning, genetic algorithm, Naive Bayes Optimisation) are provided that can be used by an agent depending on the given problem (i.e. task) and its properties. Section 4.6 introduces the concepts underlying the agents of the framework. It also outlines the algorithms used and their problem-specific extension and application by the framework agents.

Prior to the conceptualisation of the proposed agent framework, several design decisions and assumptions had to be made that set the boundaries of the agent framework. Section 4.7 outlines these assumptions and shows the limitations of the agent framework.

Finally, Section 4.8 discloses and concludes important aspects and findings of the proposed agent framework.

The concepts, methods and approaches discussed here are based on the following published papers: [152], [155],[158], [159], [156] and [157].

4.1. Overview of Roles and Framework Phases

Before the framework architecture is considered, it may make sense to look at the intended business process, i.e. usage, of the framework architecture from different perspectives and requirements. The perspectives considered include different user roles (i.e. domain expert, agent developer and software agent) that will interact with the framework. In addition, there may be users who use or are supported by the agents' services in their daily lives. Therefore, the framework needs to address different goals and requirements of the respective target users. The following section briefly highlights the different roles and their requirements to the framework.

Domain Expert The domain expert (e.g. physician, technician) has a specific domain knowledge and contributes this knowledge, which can be based on guidelines as well as experience. The domain expert does not need to have programming skills or knowledge of formal languages. He/she may be an expert in his/her field, but may not have experience or knowledge in handling computer technologies. His/her role in the context of the framework is to provide and deploy generally valid task descriptions that reflect relevant task characteristics (i.e. possible states, observable features, executable actions) and enable a software agent to solve the task at hand.

Agent Developer The agent developer has the task of implementing problem-specific optimisation or machine learning algorithms for agents. This also includes determining hyper-parameters needed for the corresponding algorithm. Since task execution sequences can be unpredictable and stochastic during coding time, an agent developer needs to rely on algorithms that allow the agent to independently learn dynamic and flexible strategies that are context-dependent and sometimes unknown. Moreover, task and user requirements may change, making it necessary to enable agents the adaptation of their strategies during run-time to new upcoming situations. Within the proposed framework, the agent developer therefore contributes implementations of algorithms to extend the range of agents and problem-specific solutions. If algorithms are already available in the framework, an agent developer can reuse them by defining semantic agent entities and parametrising them. The extension and programming of algorithms for the framework is only necessary if the existing algorithms are not suitable for a new task and its characteristics. Usually, the agent developer has to follow provided interfaces of the framework that allow the integration of new agent implementations.

Software Agent Software agents and their different properties have already been discussed in the foundations Section 2.1. The framework distinguishes and supports two types of software agents, a) *worker agents* that learn, develop and share their strategies and b) *consumer agents* that retrieve and reuse provided strategies. The latter agents are independent from the framework and communicate via the World Wide Web (WWW) with the framework, while the first ones are integrated in and controlled by the framework. However, there are also *hybrid agents* that are both worker and consumer agents, which means that they combine both agent types and can switch their role from one mode to the

other depending on the purpose and requirements.

Worker agents are basically agents that implement machine learning (ML) and optimisation algorithms, while *consumer agents* can include all kinds of software agents that only have the requirement to adhere to the APIs and protocols of the framework. In the context of the presented framework, worker agents are considered as individual software program entities that have a corresponding semantic representation and follow determined, common procedures, but differ in their applied algorithms and hyper-parameters. In particular, worker agents are an integral part of the framework and are used where knowledge is to be generated and shared. At the time of deployment, a worker agent has neither prior knowledge nor does it know how to solve the task at hand, since the task strategies are unknown in advance. Therefore, it is necessary to first learn a strategy, i.e. an intended behaviour, that enables the agent to achieve the task's goal(s). For this reason, the agent has to observe and evaluate the environmental feedback it receives for its performed actions, also to determine the termination of its learning process. The termination of training procedures depends on the convergence of the trained ML model and the achieved performance, i.e. utility, of the agent.

After the training, the worker agent generates and shares provenance data about the training it has completed (e.g. training iterations, task learned, performance measured by rewards). This provenance data is intended for *consumer agents* who would like to assess the suitability of the trained strategies provided. Furthermore, provenance data about the origin of strategies ensures accountability so that consumer agents are enabled to decide whether the provided strategies meet their quality criteria.

End-Users End-users are the last category of user roles that are part of the framework life cycle. They usually interact with the agent or the framework by given IoT devices (i.e. wearable devices, stationary sensors, actors, applications). In this context, end-users provide direct and active or indirect and passive feedback that is utilised by the corresponding agents in order to adapt their strategies to user-specific needs, preferences and requirements. It is important to note that end-user preferences and domain expert objectives may diverge. A made assumption and design decision of the agent framework is that in these cases the objectives of the domain experts (e.g. physician) have a higher priority than the end-user (e.g. patient) preferences. This assumption makes sense because domain experts are considered as stakeholders who specify the intended task objectives. If end-users want to utilise the provided services they have to agree and adhere to the given objectives and corresponding strategies.

The Framework Phases The framework is modularly designed because its process flow covers different aspects that address different requirements. The subsequent paragraphs deal with the mentioned process flow and its inherent framework components. In the scope of this work, five phases have been identified that are considered representative and useful for the process flow of the agent framework. The appropriate phases are discussed

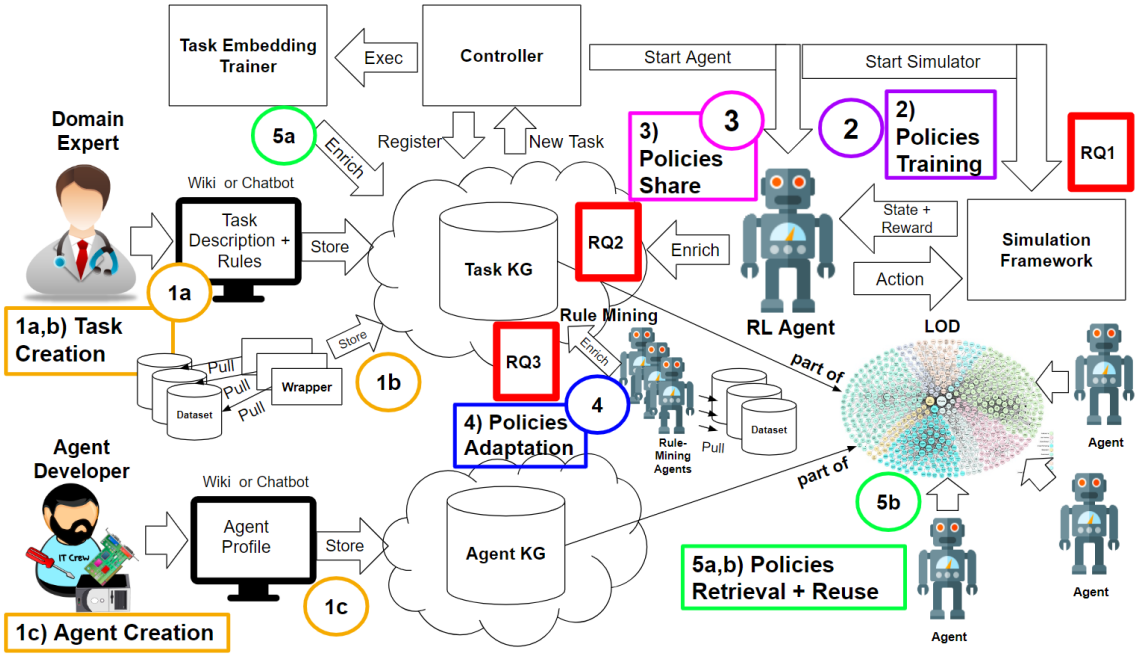


Figure 4.1.: Big picture of the presented approach.

and highlighted by coloured numerical labels in Fig. 4.1. Some phases are subdivided and represented by lower-case letters. The list below identifies the different process phases:

1. Semantic task and agent instance creation.
2. Policies training.
3. Policies share.
4. Policies adaptation.
5. Policies retrieval and reuse.

(1) Semantic task and agent instance creation **a)** The developed agent framework requires a user interface (UI) (e.g. a chatbot application, see Fig. 4.2) that guides domain experts to create an assertional (A-box) representation of task instances stored in a knowledge graph (KG), which in turn should be part of the *linked open data (LOD)* cloud. The representation of assertional task instances is based on the underlying terminological knowledge (T-box) defined by the corresponding light-weight framework ontology (see Sec. 4.2). Therefore, UI applications (e.g. web applications, chat bots, content management systems) that are intended to access the agent framework need to implement the semantic concepts of the developed T-box. In this first phase of the framework, domain experts specify task instances that contain relevant information about task properties and enable the simulation component to create, update and evaluate states of the task in order to provide feedback (i.e. reward values) to worker agents (e.g. reinforcement learning agents). The

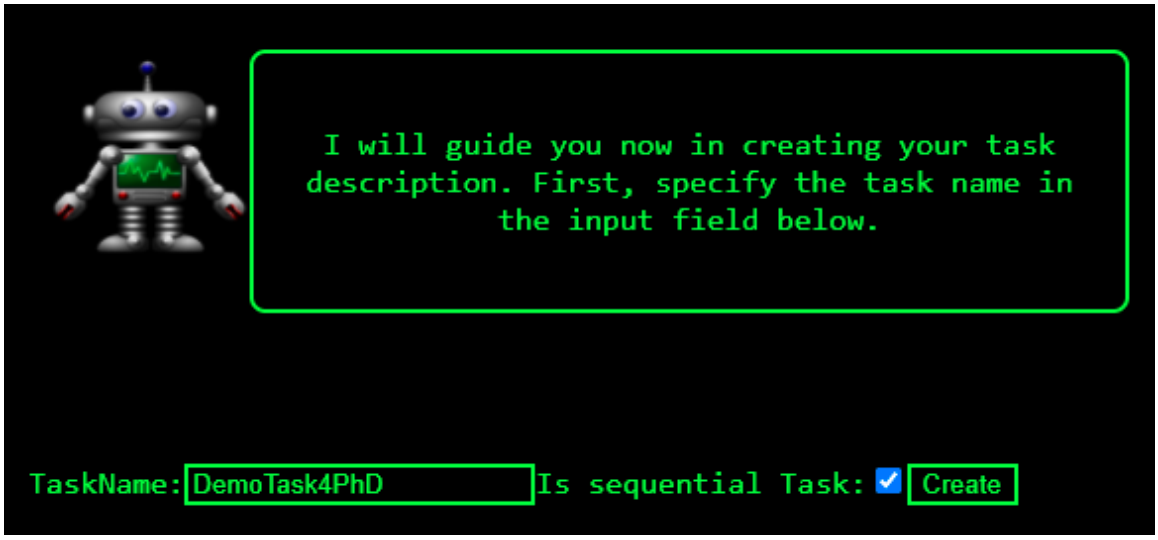


Figure 4.2.: The definition of a task instance with the guidance of a chat bot web application.

task description provides a domain-specific but abstract and generalised perspective and representation of the task that can be computationally processed by the components of the agent framework. The task entities are serialised in a semantic representation format and stored in the corresponding task knowledge graph.

The task description enables the simulation component to recognise states and update them depending on the last state and action performed. In addition, the simulation component determines what feedback (i.e. reward value) should be returned in a certain state for a performed action. It is important to note that in contrast to the simulation component, the agent does not have a strategy, i.e. policies, to solve the given task. The agent only has the ability to learn a strategy based on the given feedback by the simulation component and its inherent utility function. The reason why the agent has to learn strategies first, instead of immediately using the task description as a procedure specification, is that the task description only provides a general representation of the task properties, but not a strategy for solving the task. Some tasks are stochastic and thus their progress cannot be predicted in advance, so that the strategies required depend to a large extent on the dynamics of the environment, the preferences and requirements of the users. Thus, the targeted tasks usually have no one-fits-all strategies. For this reason, the agent needs to be able to make flexible adjustments depending on the current context and end user, so that reprogramming of hard-coded procedures, i.e. strategies, is no longer necessary, but the adaptation of the strategy is made by the agent based on received contextual feedback.

b) Independent of domain expert knowledge, there may also be data records available that contain important information about task processes and agent activities (e.g. Virtual Home Dataset (VH)[1]). However, these records sometimes cannot be used directly by the framework because they may be unstructured or have their own structure and thus do not

[1] `http://virtual-home.org/`

conform to the proposed framework ontology. In these cases, wrapper components have to provide a remedy by transforming these datasets into processable task representations. The framework requires the ability to be expandable with dataset specific wrappers so that different datasets can be transformed into semantic task descriptions. Thus, the framework also allows the automated generation of semantic task instances that are subsequently stored within the provided knowledge graph of the framework.

Wrapper components can be provided by developers or domain experts who want to utilise unstructured datasets from data sources about tasks within the agent framework. Since the syntax and semantics of such unstructured data can vary, specific wrapper implementations need to know the T-box of the framework on the one hand and the structure of the underlying dataset on the other. For the evaluation, for instance, the activities of the VH dataset were transformed into framework-specific semantic task entities. Section 6.3.3.2 demonstrates how the transformation from activity descriptions (i.e. agent programs) of the VH dataset to semantic task entities has been conducted in the scope of the evaluation. It is important to note that the proposed agent framework is extendible by various wrapper components since it is modularly built. Wrappers can register to the framework and are executed by the framework regularly (i.e. in defined time intervals) in order to adopt new datasets and updates.

c) In the same manner as domain experts, agent developers can create semantic entities of agents that subsequently are utilised by deployed worker agent programs as configuration. In this context, it has to be distinguished between *semantic agent entities* and *agent programs*. An agent entity is the semantic representation that supplies the properties respectively configuration of the worker agent while the running software instance of the worker agent is executed program code that follows a predefined program procedure and implements the execution of the agent. The semantic agent entity is required because a worker agent needs to know which task to solve and which actions it is allowed to conduct. Depending on the implemented optimisation or machine learning algorithm also hyper-parameters (e.g. learning rate, number of episodes, greedy parameter, etc.) are required that have to be provided by the agent developer before the agent program is deployed and running on the framework.

(2) Policies Training As soon as the assertional task entity and agent entity are created and stored within the knowledge graph, a *Controller* component is informed that controls the process of policy training. Thus, in addition to a server for bidirectional communication, the controller also launches a program instance of the simulation component and an instance of a working agent. Currently, the framework supports three different types of worker agents: a) reinforcement learning, genetic algorithm and naive Bayesian agent (see Sec. 4.6).

The simulation component as well as the worker agent subscribe to the communication server by sending the event, i.e. topic, names they are interested in. Each topic name is

referenced within the semantic task and agent entity.

A *topic* is an abstract representation that encapsulates IoT device events and messages. Via the communication server the agent as well as the simulation component instance are enabled to publish their own messages and to subscribe to other publishers of messages.

In the next step, the simulation component requests the semantic entity of the previously assigned task name from the knowledge graph while the worker agent requests its configuration.

With the provided knowledge, the agent starts its training. First, it randomly selects an action from the possible set of actions and publishes a message containing the selected action URI to the simulation component via the communication server. The simulation component in turn updates the current maintained state depending on the interlinked effects of the published action and simultaneously determines the reward value of the new state. Both information, i.e. new state and reward value, are then sent back by the simulation component to the agent. Based on the rewards received and the updated states, the agent iteratively trains the policies to perform actions depending on the observed states, usually, in many thousands of episodes, either until a convergence of the agent's utility function or performance with respect to the rewards is achieved or until the defined number of episodes has been performed. Each training episode ends as soon as the goal and final state of a task is accomplished. Each task can have either one initial state or a set of initial states, that begin the task execution. The simulator component selects one of these initial states either randomly or in a round-robin fashion to start a new task episode.

(3) Policies Share During the training, the worker agent maintains meta data about its machine learning model or optimisation algorithm and statistics (e.g. achieved mean positive/negative reward, number of training episodes, etc.) about the training process. After the training has finished, the agent stores these meta data as provenance data together with the trained model that represents the learned strategy to solve the corresponding task. The shared policies are generic policies that represent a general valid task strategy. As outlined below, these general task strategies can be adjusted by user and environmental feedback for personalisation purposes.

(4) Policies Adaptation The policies adaptation phase addresses the personalisation respectively customisation of task policies. Agents that execute shared policies during their runtime, collect data about their conducted activities, perceived observations (e.g. sensor values) and feedback (i.e. rewards). These datasets are valuable in that sense that they can be utilised in order to adapt and personalise agent services to contextual and personal requirements.

In this context, adaptation of strategies is achieved in two ways: 1) by adapting the models (e.g. weights) of the trained strategies, and 2) by rule mining techniques that use the collected datasets to derive representative and significant rules that reproduce the adapted

(i.e. personalised) behaviour of individual agents or users. This implies that a task can produce numerous versions and variations of task-solving strategies that can be reused by agents who need to solve similar tasks to the existing ones.

(5) Policies Retrieval and Reuse A central assumption in this work is that similar tasks, in terms of their semantic characteristics, require similar solution strategies, i.e. policies. This assumption leads to the idea that similarity measures reflecting semantic task properties need to be applied to help consumer agents assessing the suitability of task strategies in relation to their own tasks to be solved.

The framework facilitates the retrieval and reuse of task strategies by worker and consumer agents through entity embeddings (i.e. task embeddings) that represent semantic task entities by numerical vectors and enable the retrieval of strategies by determining the similarity (i.e. spatial distance within a high-dimensional vector space) of task embeddings. The more similar two tasks are to each other, the closer they are spatially distributed in a high-dimensional vector space.

a) The *TaskEmbeddingTrainer* component of the framework encodes each newly created semantic task entity and related other entities (e.g. actions, states, features, etc.) of the knowledge graph by means of a deep neural network as an embedding vector that contains the trained weights of the neural network. This numerical embedding vector is then stored as web resource to interlink it in the semantic task entity.

b) In the case that a consumer agent searches for suitable task policies for its own task, it has to provide a T-Box conform semantic task entity as input to the framework so that the *TaskEmbeddingTrainer* encodes also a task embedding for the searching consumer agent. Subsequently, the framework computes a predefined number of the most similar task entities and provides the requesting customer agent with corresponding task URIs as a result. The consumer agent can then retrieve the most similar tasks as well as the policies associated with them using the task URIs from the corresponding knowledge graph and try out the provided policies using the available simulation component. Based on the evaluated performance, the consumer agent can decide on one of the retrieved policies. The advantage is that the consumer agent does not have to learn the policies from scratch, but can reuse existing policies that are suitable for its own tasks.

This section gave an overview about the proposed framework architecture, user roles and life-cycle phases. In the next sections single framework components are discussed in detail.

4.2. Framework Ontology: Formal Representation of Domain-Independent Knowledge

In this section, the knowledge representation (i.e. ontology) developed for the proposed agent framework is presented and discussed. The concepts presented are based on the following published works: [159, 158, 155, 152], where parts of the ontology have already been presented.

The framework as well as the agents require an ontology in order to have a shared understanding of relevant concepts and their relationships among each other. Ontologies allow agents to infer knowledge that is implicitly given in semantic facts and their relationships to each other (see also Sec. 2.2).

Many ontologies and vocabularies [47, 34, 270, 124, 243, 43, 221, 296, 50] are available for different domains and application fields in the World Wide Web (WWW). However, nevertheless a new use case and application specific ontology was conceptualised for the agent framework because it required less effort to conceptualise an own ontology than extend already existing ontologies that do not provide all required concepts and properties reflecting certain task- and agent characteristics.

One objective regarding the framework's knowledge representation was to have a low complexity of the ontology, so that a simplified usage and adoption of the ontology is facilitated. For this reason, a light-weight ontology was defined that is not too overloaded but sufficiently generic and expressive to share and process common task- and agent-specific knowledge across domains and applications.

In the analysis and design of the concepts and their relationships, a common denominator was sought that characterises the commonalities between tasks and the commonalities between agents. The common denominator of tasks was found by the Markov decision process (MDP) character of each considered task. Hence, the MDP and its vocabulary served as role model for the conceptualisation of the ontology. Elements (e.g. states, actions, rewards) representing the formal vocabulary of MDPs were adopted into the ontology and additional concepts (e.g. effects, observables, equations, variables, etc.) and properties were derived that go beyond the MDP vocabulary, as tasks can also have different characteristics that make them distinguishable from each other and imply a differentiated handling by platforms (e.g. simulation component) and agents. Therefore, these characteristics need to be reflected within the proposed ontology. The following bullet points list the various considered characteristics and differences that a task may have and that affect the handling of the task within the proposed simulation framework and within agent programmes.

- Varying number of participants, i.e. single or multi agent tasks.
- Consists of sub-goals and stages that first have to be fulfilled before the final state is reached.

- (A-)synchronous handling of state updates in multi-agent environments, i.e. for the update of states, the simulation framework has either to wait or not to wait until each agent has performed its action.
- The task requires for its solving, fixed ordered action sequences versus no fixed ordered actions.
- The task describes effects and corresponding state changes by rules and/or differential equations, see Sec. 4.2.1.
- The task consists of states that are represented by continuous and/or discrete observation features.

The framework ontology that addresses these task characteristics, consists of 15 concepts, 23 object properties and 34 data properties. It represents semantic concepts that can be adopted and extended, e.g. by *subsumptions* or property relationships, to achieve interoperability between different domains. Sec. 4.2.1 outlines the requirements to support domain experts in simulating natural phenomena using mathematical models. In Sec. 4.2.2 all concepts, their meaning and purpose are described.

4.2.1. The Simulation of Phenomena by Mathematical Models

In real-world scenarios, natural phenomena and events play a crucial role, since they follow natural laws and patterns that can be represented and simulated by mathematical models [246, 71, 26]. For instance, upcoming pandemics (e.g. COVID-19) [119] or social and political phenomena (e.g. president elections, terrorism, etc.) evolve according to certain laws, patterns, events and causes [248]. These natural or artificial laws, patterns, events and causes can be recorded in data [294] and modelled by mathematical equations (e.g. differential equations, fuzzy logics) [148], consisting of parameters and variables that reproduce, i.e. emulate, the dynamics (e.g. rate of change) of the appropriate phenomena.

This thesis proposes a very generic model representation that allows domain experts to create their own task and environment specifications for simulation purposes without requiring programming knowledge and without being limited to game environments.

Since the treatment of real-world phenomena, driven by decision making, requires appropriate mathematical models in the virtual world, a simulation framework for (RL) agents has to be able to utilise these mathematical models in order to simulate states that follow the emerging changes. This is particularly relevant for complex phenomena whose non-linear and non-parametric models cannot be derived or computed directly by the agent.

Pre-condition for the simulation of phenomena and dynamics is that the appropriate mathematical models are formally represented and serve for processing and emitting state changes. An important aspect thereby is that model parameters, reflecting the impact of measures and actions, can help to control according state changes in the appropriate environment. Usually, prior knowledge through data [198] and assumptions of domain

experts guide the parametrisation and mathematical modelling process [19, 147], whereby their implications need to be formally represented and transformed by feature engineering [198] into processable state representations (e.g. numeric and/or categorical) so that RL agents can learn to deal with pending states to achieve predefined and intended goals.

An interesting aspect is that implications (effects) of actions can be simulated in order to evaluate the more or less beneficial impact and utility of action sequences to the according process and environment. Moreover, agents can learn to handle trade-offs with respect to contradicting goals [168]. The challenge thereby is to find action sequences that serve for heterogeneous objectives of a task. If some reconciled actions sequences are found, they can be formalised as concise decision rules and shared with rule-based agents.

An additional aspect of describing and simulating phenomena and tasks can be that not just one but multiple actors (agents) can influence mutually the process and environment by their performed actions and behaviour (e.g. cooperation, collaboration, competition) [14, 72, 200]. This observation shows the necessity of having an agent-based simulation framework that can synchronise actions [277] and deal with multiple actors (agents) influencing by their actions separately the course of a task process.

Furthermore, it may be useful for a simulation framework to be able to represent and simulate mathematical models in addition to rules to show the long and short term effects of actions taken by one or more agents. Therefore, professionals and researchers who do not know programming languages may need means to specify and configure the behaviour of simulation environments through mathematical equations, such as differential equations. This may be important because a simulation framework should also take into account trade-offs between competing goals of agents through mathematically modelled effects.

The presented work aims at providing an approach to define and integrate different environments into the simulation framework that contain rules and mathematical models for state generation and state updates depending on executed actions and their effects. The framework ontology presented in Sec. 4.2.2 enables a configurable agent framework that allows also not technically inclined domain experts to specify and integrate simulation environments with rules and differential equations for simulating phenomena and short- and long-term effects of actions and measures.

4.2.2. Concepts and Properties

State

A state is a concept that represents and encapsulates possible and measurable observation features within a task. Thus, a state aggregates all possible features that can be perceived and evaluated by an agent. Four types of states are supported: *initial, interim, final and goal state*. The initial state indicates the starting point of the given task. Interim states are states that occur between initial and final state. The final state terminates the task. It is important to note that a distinction between final and goal state is made, because a task

can comprise several goal states that do not necessarily terminate the task but indicate that parts of the task were successfully solved. The state concept can be considered the equivalent of states in Markov decision processes (see Sec.2.3.9).

The *hasExpression* property references string literals and allows the agent to recognise the state. An expression can be considered as rule with logic operators (*AND*, *OR*, *XOR*, *NOT*) and range expressions (<=, >=, <, >) that prescribe conditions and limits of the state. For instance the conditions in Example 4.1 might apply to the state *NormalBodyTemperature*. Thus, the mentioned state would reference the illustrated expression. An agent that monitors the body temperature of a person, might decide in this way whether the person has a low, normal or high body temperature. Based on the recognised state, it then could decide its next action to perform. It is important that all observation features that are stated in the expression require to be referenced by the state's property called *hasObservationFeature*.

ObservationFeatures are measurement variables that are indicators of a state and are observed by the agent in order to recognise states. The corresponding rules can be represented in different rule languages, e.g. SWRL, Notation3, SPARQL (see Sec. 2.2.3, 2.2.4, 2.2.5). In Sec. 4.3 examples will be provided that demonstrate how rules are created and utilised for reasoning purposes.

$$(\text{BodyTemperature} \geq 36.0) \wedge (\text{BodyTemperature} \leq 37.4) \rightarrow \text{NormalBodyTemperature} \tag{4.1}$$

Concept: State			
Roles/Properties	**Range**	**Mandatory**	**Cardinality**
:hasObservationFeature	:ObservationFeature	Yes	≥1
:isGoal	xsd:boolean	Yes	1
:isInitialState	xsd:boolean	Yes	1
:isFinalState	xsd:boolean	Yes	1
:hasExpression	xsd:string	Yes	1

Table 4.1.: State concept and its properties.

Observation Feature

Observation features are observable quantities that define entirely the given state by their properties. For instance, different observation features that are relevant for the task execution process, allow the recognition of states and contexts through their evaluation in combination.

An observation feature can have, depending on the measurable variables, different value ranges. For instance, a temperature sensor can only measure a certain range of values (e.g. from 0 to 100). Depending on available sensors and their sensitivity and capability to

measure, the value ranges of the features have to be defined. Therefore, the properties *hasRangeStart, hasRangeEnd* reference the corresponding range limits.

Observation features can have various statistical properties, e.g. mean, median, mode, variance and probability distributions, i.e. probability density functions (PDFs) or probability mass functions (PMFs), which need to be parametrised to extract the most likely values from the distribution for simulation purposes. For simplicity, standard distributions are specified in the observation feature concept, e.g. *Gaussian normal, Poisson, binomial, uniform and exponential distribution*, which are commonly used in descriptive statistics. It is optional whether the domain expert specifies the statistical properties and a data distribution for the corresponding observation characteristic. If no feature distribution is specified, *NONE* has to be selected for the property *hasProbabilityDistribution*. The statistical properties are useful and show their strength when domain experts know which statistical properties and data distributions they want to simulate or when tasks and thus observation features are derived and created from datasets. The corresponding distributions and parameters, such as mean and variance for a normal distribution or e.g. λ, success/failure rate for the binomial and exponential distribution, can be estimated from the given data and specified in the observation feature entity.

Each observable feature variable is defined by its feature type and its current value. A feature can be either a categorical, i.e. nominal, feature or a numerical, continuous feature. Thus, the property *hasFeatureType* indicates the corresponding feature type while *hasValue* indicates the observed value at a certain time.

Since quantities are measured in units, there is also a *hasUnit* property that specifies the corresponding unit string literal (e.g. Fahrenheit or Celsius degree).

Topic

A topic is a concept that encapsulates one or multiple observation features. As soon as an observation feature changes in the environment, the subscribed agent is informed. In this event-driven way, new upcoming states can be observed by agents. The term *topic* is adopted from the MQTT application protocol where topics deal as entity for messages that can be subscribed by clients that would like to receive appropriate messages.

The observable features that are aggregated in a topic are referenced by the *hasObservationFeature* property.

Each topic can have a human readable name that is stated by the property *hasName*.

Action

Actions that are interlinked to the task and the agent, represent actions that can be conducted by the agent and may be related to objects (e.g. operating elements such as devices,

Concept: ObservationFeature			
Roles/Properties	**Range**	**Mandatory**	**Cardinality**
:hasRangeStart	xsd:double	Yes	1
:hasRangeEnd	xsd:double	Yes	1
:hasMeanValue	xsd:double	No	≤ 1
:hasVariance	xsd:double	No	≤ 1
:hasStandardDeviation	xsd:double	No	≤ 1
:hasMedian	xsd:double	No	≤ 1
:hasModeValue	xsd:double	No	≤ 1
:hasLambda	xsd:double	No	≤ 1
:hasNumberSuccesses	xsd:integer	No	≤ 1
:hasSuccessRate	xsd:double	No	≤ 1
:hasFailureRate	xsd:double	No	≤ 1
:hasNumberExperiments	xsd:integer	No	≤ 1
:hasProbabilityDistribution	{NONE, GAUSSIAN, BINOMIAL, UNIFORM, EXPONENTIAL, POISSON}	Yes	1
:hasFeatureType	{NOMINAL, NUMERICAL}	Yes	1
:hasUnit	xsd:string	No	1
:hasValue	xsd:double	No	≤ 1

Table 4.2.: ObservationFeature concept and its properties.

Concept: Topic			
Roles/Properties	**Range**	**Mandatory**	**Cardinality**
:hasName	xsd:string	Yes	1
:hasObservationFeature	:ObservationFeature	Yes	1

Table 4.3.: Topic concept and its properties.

actors) that are invoked or triggered by the actions.

An action can have a description in natural language that is intended for human experts or users who would like to comprehend the purpose of an action what might be interesting especially regarding the explainability of agent behaviour.

Since some actions may have counteractions that cause complementary effects, a property called *hasCounterAction* is defined to indicate whether an action is a complementary action with opposite effects. Opposing effects may sometimes be required, as an agent may get stuck in a state and need to perform an action that undoes a previous action and lets the agent move back out of the state. Complementary actions are particularly needed when new policies are trained. Once the simulator detects that the agent is stuck, it calls itself a complementary action that returns to a previous state where the agent can continue its training.

The property *hasTransition* is optional and only useful if corresponding knowledge graphs can be created from MDP datasets and state transitions that have occurred can be extracted from the data. An action can have any number of state transitions, depending on the number present in the corresponding dataset.

It is conceivable that an action may also have a cost associated with it. For instance, it could be that it costs an agent energy or battery life to perform actions. For this reason, the *Action* concept also includes the *hasCost* property, whereas the use of this property is optional. However, if costs are to be modelled, these costs are added to the reward values of subsequent states of an action by the simulation component during the simulation.

The most important property of an action might be *hasEffect* that references instances of the effect concept. Defining effects is especially required for simulation purposes since the simulation component updates states based on performed actions and their effects that directly impact observable features in the agent's environment. Hence, an effect describes how an observable feature will most likely change.

Concept: Action			
Roles/Properties	**Range**	**Mandatory**	**Cardinality**
:hasDescription	xsd:string	No	≥0
:hasEffect	:Effect	Yes	≥0
:hasCounterAction	:Action	No	≥0
:hasTransition	:Transition	No	≥0
:hasCost	xsd:double	No	≥0

Table 4.4.: Action concept and its properties.

Effect

The ontology represents impacts of actions through the concept *Effect*. An effect references (differential) equations or constant values to update the values of observable features based on different defined operators linked through the property *hasImpactType*. The impact types define the possible operations to update the corresponding feature variables. It is possible to *compute* a feature value by an equation linked through the property *hasEquation* or to change feature values by the following operators *increase/decrease*, *convert*, *on/off* and *constant*. In case the impact type is set to *constant*, the feature is assigned a constant value that is immutable.

Concept: Effect			
Roles/Properties	**Range**	**Mandatory**	**Cardinality**
:hasObservationFeature	:ObservationFeature	Yes	≥ 1
:hasImpactType	{INCREASE, DECREASE, CONVERT, ON, OFF, CONSTANT, COMPUTE}	Yes	1
:hasEquation	:Equation	No	≤ 1

Table 4.5.: Effect concept and its properties.

Transition

The *Transition* concept is only relevant when semantic task entities are automatically created from datasets representing MDPs. In these cases, the *Transition* concept can be used to represent state transitions in an MDP caused by actions performed by an agent. This implies that each transition consists of an initial state, a subsequent state, an action and a transition probability, as this is also defined in MDPs. The transition probability takes into account the degree of uncertainty of an environment, and in deterministic environments this probability is usually 1 or 0, while in stochastic environments this value can vary. Transition probabilities are derived from the data.

Concept: Transition			
Roles/Properties	**Range**	**Mandatory**	**Cardinality**
:hasPreviousState	:State	Yes	1
:hasNextState	:State	Yes	1
:hasAction	:Action	Yes	1
:hasTransitionProbability	xsd:double	Yes	1

Table 4.6.: The *Transition* concept and its properties.

Virtual Influencer

The *Virtual Influencer* is a concept that specifies random and exceptional events that can occur independently of the agent acting in the environment and thereby training policies. A virtual influencer can be particularly important when the agent being trained has also to respond to unpredictable events that may occur randomly and outside of the agent's control. Thus, the virtual influencer has the task of introducing uncertainty into the environment in order to prepare the agent also for unforeseen or random events.

A *Virtual Influencer* entity is of type *Agent* and emits a stochastic event that can be controlled by three different conditions: 1) a probability of occurrence, 2) a time frequency of occurrence or 3) a pre-conditional state, see Table 4.7.

Concept: VirtualInfluencer			
Roles/Properties	**Range**	**Mandatory**	**Cardinality**
rdfs:subClassOf	:Agent	Yes	1
:hasEffect	:Effect	Yes	≥1
:hasProbability	xsd:float	No	1
:hasExecutionIntervall	xsd:float	No	1
:hasPrecondition	:State	No	1

Table 4.7.: VirtualInfluencer concept and its properties.

Strategy

Strategy is a concept that describes strategies of agents to solve respectively conduct a task. Thus, a strategy instance aggregates and references by the property *hasPolicy*, policies that define the action instructions for each probable state.

Since reinforcement learning agents train policies first before they apply them to a task, each policy and thus strategy is derived by an agent. Therefore, the *hasAuthor* property references agents as authors of the corresponding strategies.

Concept: Strategy			
Roles/Properties	**Range**	**Mandatory**	**Cardinality**
:hasPolicy	:Policy	Yes	≥1
:hasAuthor	:Agent	Yes	1

Table 4.8.: Strategy concept with its properties.

Policy

Policies were already introduced in Sec. 2.3.10. In analogy to policies in reinforcement learning, a semantic *Policy* concept maps a state to an action. The objective, especially

for RL agents is to train policies that promise a reward maximising behaviour. The policy concept is implemented in two alternative ways, e.g. as *reification* by subsuming a *rdf:Statement* or SWRL rules. Reification allows to make statements about statements what is useful in this case because a policy is considered as a statement that expresses which action instance is required by a given state instance (see Sec. 2.2). In this context, *rdf:subject* references instances of the state concept, *rdf:predicate* references a property named *requiresAction* and *rdf:object* references instances of the concept action. Reification can be used without requiring a specific rule engine. Listing 4.1 illustrates an example for a COVID policy that relates the state named *COVIDSymptoms* to the action named *StayAtHome*.

Listing 4.1: Reification used for a COVID-19 policy.

```
1 :COVIDPolicy rdf:type rdf:Statement.
2 :COVIDPolicy rdf:subject :COVIDSymptoms.
3 :COVIDPolicy rdf:predicate :requiresAction.
4 :COVIDPolicy rdf:object :StayAtHome.
```

As an alternative to reification, a policy can also be expressed by predicate logic (e.g. SWRL rule, see Sec. 2.2.4), which is represented by a string literal in the agent framework. The *hasRule* property refers to the corresponding literal. This literal can be retrieved and processed by rule-based agent programs to derive the action to be performed when a state defined by the rule is true. Example rule 4.2 is analogous to the previous reification example and describes the conditions that allow to infer the state called *COVIDSymptoms*. The rule specifies that the variable ?p is an instance of the class *Person*. The person requires to satisfy 3 different symptoms (e.g. DryCough, Fever, Nausea) concatenated by conjunctions. If all 3 expressions are true, the rule infers that the state *COVIDSymptoms* is given.

$$\begin{aligned} &\mathsf{Person}(?\mathsf{p}) \land \mathsf{hasSymptom}(?\mathsf{p}, \mathsf{DryCough}) \\ \land\ &\mathsf{hasSymptom}(?\mathsf{p}, \mathsf{Fever}) \land \mathsf{hasSymptom}(?\mathsf{p}, \mathsf{Nausea}) \\ &\rightarrow \mathsf{State}(\mathsf{COVIDSymptoms}) \end{aligned} \tag{4.2}$$

Analogously to the previous examples, if the agent infers from the observed symptoms that the state *COVIDSymptoms* is present, it can deduce from the example SWRL rule shown in Eq. 4.3 that the action named *StayAtHome* needs to be performed. In this way, policies can be described as rules.

$$\begin{aligned} &\mathsf{Agent}(?\mathsf{a}) \land \mathsf{Action}(\mathsf{StayAtHome}) \land \mathsf{State}(\mathsf{COVIDSymptoms}) \\ &\land\ \mathsf{recognisesState}(?\mathsf{a}, \mathsf{COVIDSymptoms}) \\ &\rightarrow \mathsf{requiresAction}(\mathsf{COVIDSymptoms}, \mathsf{StayAtHome}) \end{aligned} \tag{4.3}$$

Parameter

Since equations are required for expressing some effects, the ontology provides the *Param* concept. *Param* represents parameters that are immutable, fixed values and are used within

Concept: Policy			
Roles/Properties	**Range**	**Mandatory**	**Cardinality**
rdfs:subClassOf	rdf:Statement	Yes	1
rdf:subject	:State	Yes	1
rdf:predicate	:requiresAction	Yes	1
rdf:object	:Action	Yes	1
:hasRule	xsd:string	Yes	1

Table 4.9.: Policy concept and its properties.

an equation in order to influence dependent variables, i.e. observation features. A *Param* concept consists of a *hasName* property that defines the name of the parameter, a *hasValue* property that references the parameter value and a *hasDescription* property that describes the purpose of the parameter.

Concept: Param			
Roles/Properties	**Range**	**Mandatory**	**Cardinality**
:hasName	xsd:string	Yes	1
:hasValue	xsd:double	Yes	1
:hasDescription	xsd:string	No	1

Table 4.10.: Param concept and its properties.

Equation

An equation concept is important for mathematically describing the effects (e.g. rate of change) of actions that allow updating states. An equation concept consists of a mathematical expression that is represented through a string literal and referenced by the property *hasExpression*. The parameters and variables of the equation are referenced via the *hasParam* property. An equation can have an arbitrary number of parameters that have to rely to the parameter names that are stated in the corresponding expression. A property called *hasDescription* allows to refer to text that describes the purpose of the equation.

Concept: Equation			
Roles/Properties	**Range**	**Mandatory**	**Cardinality**
:hasExpression	xsd:string	Yes	1
:hasParam	:Param	Yes	≥1
:hasDescription	xsd:string	No	1

Table 4.11.: Equation concept and its properties.

Algorithm

The *Algorithm* concept represents the algorithms that agents implement for solving their tasks. The algorithm concept is especially required when an agent entity is defined that specifies for the agent the algorithm to select. The properties are: *hasName*, that references the name of the algorithm, *hasParam* that references hyper-parameter instances that are required for initialising and configuring the algorithm and *hasDescription* that describes in natural language text the purpose of the algorithm.

Concept: Algorithm			
Roles/Properties	**Range**	**Mandatory**	**Cardinality**
:hasName	xsd:string	Yes	1
:hasParam	:Param	No	≥1
:hasDescription	xsd:string	No	1

Table 4.12.: Algorithm concept and its properties.

Agent

The *Agent* concept describes the semantic representation of agents that conduct tasks. The objective is to provide relevant information about agent instances that allow the initialisation and configuration of agents.

For instance, agents can use the *hasTopic* property to determine which *Topic* entities they need to subscribe to in order to be informed as soon as associated observation features change.

The property named *hasAlgorithm* references the corresponding algorithm that is implemented by the agent.

Since agents, especially RL agents, train strategies in numerous episodes, the property *hasEpisodes* indicates how many episodes the agent should train to learn appropriate task strategies.

Depending on the agent's abilities or competencies, the agent has a set of actions that allow the agent to act within a task. The appropriate actions are referenced via the *hasAction* property.

Depending on the observable features and the state representation, it may be necessary to encode states using hash functions. The property *isHashing* indicates whether or not the agent should apply hashing to observable features.

Usually agents are responsible for certain tasks and therefore, the property named *hasTask* allows the agent developer to assign the appropriate tasks to the newly created and deployed agent.

Concept: Agent			
Roles/Properties	**Range**	**Mandatory**	**Cardinality**
:hasName	xsd:string	Yes	1
:hasAlgorithm	:Algorithm	Yes	1
:hasEpisodes	xsd:integer	Yes	1
:hasAction	:Action	Yes	≥1
:isHashing	xsd:boolean	Yes	1
:subscribesFor	:Topic	Yes	≥1
:hasTask	:Task	Yes	1

Table 4.13.: Agent concept and its properties.

Use Case

A *UseCase* is a concept that represents real-world use cases, (e.g. coaching a patient, making the household, etc.). It consists of an arbitrary number of tasks that have to be conducted in order to implement the use case.

Concept: UseCase			
Roles/Properties	**Range**	**Mandatory**	**Cardinality**
:hasName	xsd:string	Yes	1
:hasTask	:Task	Yes	≥1

Table 4.14.: UseCase concept and its properties.

Task

The task concept represents tasks that are intended to be solved by software agents. It encapsulates all relevant information (i.e. properties) of a task and interlinks all MDP concepts such as e.g. *States* that can occur within a task execution and that allow the agent to decide which action to take next. The different characteristics of a task impact framework specific processes (e.g. the selection of suitable algorithms, the process flow of the simulation component, etc.).

A task can be in three different stati (*OPEN, INPROGRESS, DONE*) that indicate whether an agent currently conducts a task or has already solved the task. Since a numerical representation of tasks may be required for (machine learning) algorithms, tasks can refer to corresponding web resources of entity embeddings via the property *hasEmbedding*, which represent the task numerically in an embedding space (see Sec. 2.3.12).

The *hasStrategy* property references semantic representations of strategies that were already trained by agents. In this way agents can access available strategies and reuse them in order to retrain and adapt the strategies for their own purposes.

The *subscribesFor* property references *Topic* entities that are required for agents that want to get informed about task-related state changes in the environment. By means of topic entities agents are enabled to subscribe to dedicated topics that encapsulate messages from sensors and peripheral devices. These topics are referenced as well in the corresponding task concept so that the agent knows which topics are relevant for the corresponding task.

Tasks that belong to one use case can also be sequentially interlinked or executed in parallel. This is expressed through the properties *hasPreviousTask, hasNextTask, hasParallelTask*.

Multiple *VirtualInflucencer* entities can be referenced by a task, as they may be required to initiate unpredictable, i.e. random, and infrequent environmental events (e.g. the power goes out or a sudden obstacle appears) that directly affect the agent's perceived state, although the agent does not directly influence these events through its own actions performed.

In single- or multi-agent environments either one or multiple agents can participate a task and influence each others state representation (see Sec. 2.1). In order to indicate whether the task is executed in a multi-agent or single-agent environment, the *hasNumberOfActors* property references the number of agents that conduct the task.

State updates can be performed by the simulation framework either synchronously or asynchronously. The property *hasCommunicationType* specifies, especially in a multi-agent environment, whether a state is updated only after all agents have performed synchronously their action or after each agent has performed its action individually, i.e. asynchronously.

Sometimes a task may only be solvable if a fixed order of actions is performed and thus these tasks are referenced as *sequential* tasks. The property *isSequential* indicates through a boolean literal whether the corresponding task is a sequential ordered one or an unordered one. All the concepts and properties presented serve the agent framework for knowledge representation and communication between agents and the framework. For example, the simulation component of the framework is able to simulate tasks based on the given concepts (T-box) and their instantiations (A-box) (see Sec. 2.2), which is presented in Sec. 4.3. In addition, agents can perform reasoning to recognise states and infer policies. If concepts are insufficient or missing, the ontology can be extended with additional concepts and properties, as well as references to other existing ontologies. For this reason, the framework ontology was developed because it brings the advantages mentioned above.

Concept: Task			
Roles/Properties	**Range**	**Mandatory**	**Cardinality**
:hasState	:State	Yes	≥1
:hasAction	:Action	Yes	≥1
:hasObservationFeature	:ObservationFeature	Yes	≥1
:hasStatus	{OPEN, INPROGRESS, DONE}	Yes	1
:hasEmbedding	:Embedding	No	≥0
:hasVirtualInfluencer	:VirtualInfluencer	No	≥0
:hasStrategy	:Strategy	No	≥0
:susbscribesFor	:Topic	Yes	≥1
:hasPreviousTask	:Task	No	≥0
:hasNextTask	:Task	No	≥0
:hasParallelTask	:Task	No	≥0
:hasNumberOfActors	xsd:integer	Yes	≥1
:hasCommunicationType	{asynchronous, synchronous}	Yes	1
:isSequential	xsd:boolean	Yes	1

Table 4.15.: Task concept and its roles respectively properties.

4.3. Simulation of MDP Tasks

Task simulation is required for agents that need to learn policies for assigned tasks before they can perform them in real environments. Thus, the simulation component, i.e. the simulator, addresses two different goals: a) avoiding the *cold-start* problem by allowing the agent to train a default strategy (i.e. policies), and b) revealing short- and long-term effects of actions to be performed in order to classify how demonstrated behaviour might play out in the future.

Figure 4.3 is adopted from the paper [155]. It illustrates the process flow between simulator and agent. First, domain experts create task descriptions as JSON-LD serialisation and the framework stores the task descriptions in a knowledge base respectively knowledge graph. The simulation framework utilises the task description in order to simulate the task. Therefore, the simulator and corresponding agent exchange messages that contain instantiations about states, rewards and actions. The outcome of the simulation process are trained policies or ML models, e.g. Deep-Q-Neural-Network (DQNN), that serve as default strategy for solving a task. The program of the simulator performs four main

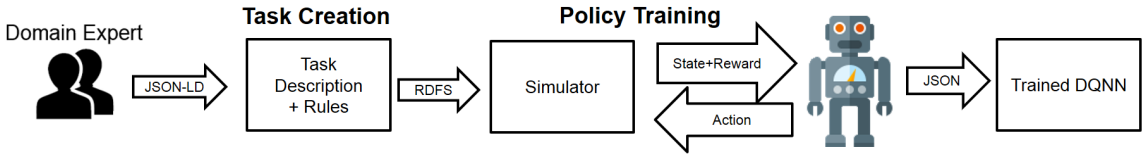


Figure 4.3.: The simulation process between agent and simulator [155].

procedures, namely (i) updating and managing states based on received actions, (ii) inferencing and providing feedback through reward signals to the agent, (iii) initialising a task once the simulation is started and terminating the task once the final state is reached.

Figure 4.4 illustrates the temporal sequence of the simulation. The sequence starts with the initialisation of the agent programme and the simulator. Then the agent sends an initial message to the simulator specifying the task to be simulated and containing the task instance as a Turtle RDF or JSON-LD serialisation with all the required information. The simulator then sends the initial state representation of the task to the agent. Since an agent usually needs many episodes until its utility function converges and it learns a desired behaviour, a program loop is executed that runs for n episodes. The number of episodes is specified in the agent's semantic entity by the developer.

Then the agent programme enters a second loop that stops as soon as the condition that a final state is reached is met. Iteratively, the agent incorporates the received reward values into its algorithm (e.g. reinforcement learning) and takes the received state representation as input to its machine learning algorithm, which returns the index of the action to be performed. The agent then sends a message with the action name to the simulator, which updates the given state based on the action and its effects and determines the new reward value based on the new updated state. Both the updated state and the reward value are

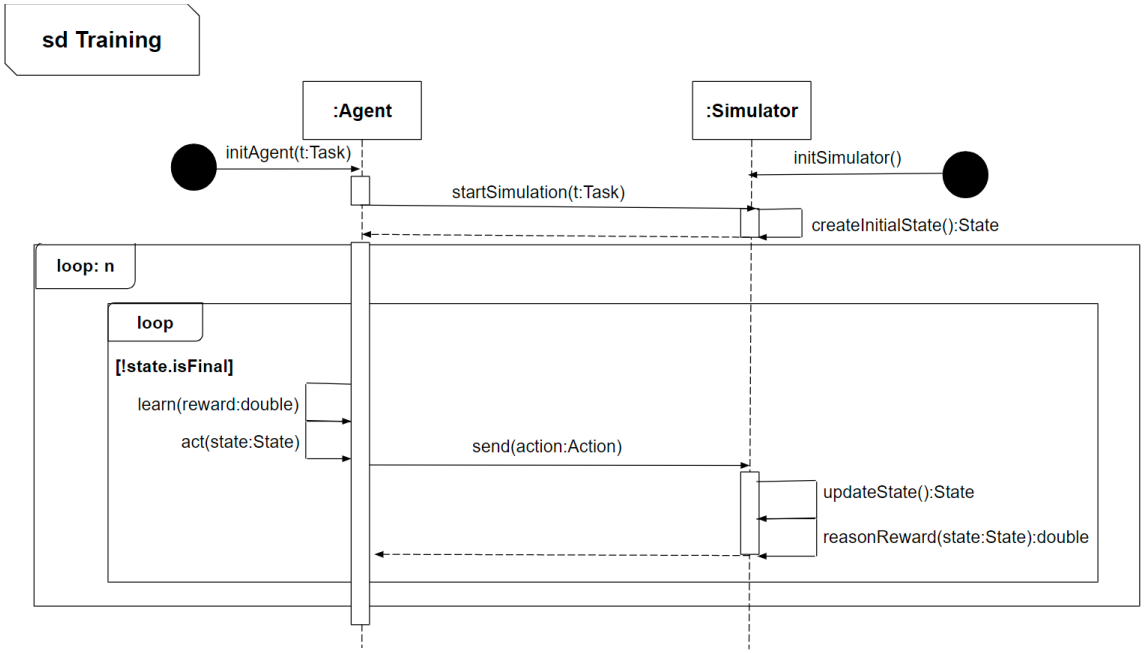


Figure 4.4.: The UML sequence diagram illustrates the flow of the simulation and the interaction between agent and simulator.

sent back to the agent. The training episode and the communication between agent and simulator are terminated once a final state is reached and once all predefined episodes have been run.

4.3.1. Message exchange

The communication between agent and simulator is conducted through JSON messages. The following example shows a message that is sent by the agent to the simulator. The message contains a time-stamp, the message status that indicates that a training is conducted and the action with its effect called *Set_LampOn* that is to be invoked. The information provided to the simulator enables it to update the given state and compute the reward value for the sent action. The following code in Listing 4.2 shows the sample message that the agent sends with the action command to switch on a lamp:

Listing 4.2: Message exchange between agent and environment.

```
{"timestamp": "2021-06-10T14:33:04.955Z",
  "status": "training",
  "action": {
    "Switch_LampOn": {
      "HasText": "Switching a lamp on.",
      "IsNegation": false,
      "HasEffect": {
        "Set_LampOn": {
          "HasImpactType": "ON",
```

```
        "HasObservationFeature": {
          "IsLampOn": {
            "HasRangeStart": 0,
            "HasRangeEnd": 1,
            "HasFeatureType": "NOMINAL",
            "HasUnit": "",
            "HasProbabilityDistribution": "NONE"
          }
        }
      }
    }
  },
  "step": 1}
```

Listing 4.3 is an exemplary response of the simulator given with information about the last state, updated state, reward value and last sent action together with its related effect. The message also indicates whether the updated state is a goal state. Furthermore, the *stateNames* key references the reasoned human readable names of the given states. It can be noted that the task context can be represented by an arbitrary number of states that describe observations. The listed example message adheres to the concepts of the previously discussed framework ontology:

Listing 4.3: Message sent to agent by simulator.

```
{"timestamp": "2021-06-10T14:49:37.601Z",
  "status": "training",
  "state": {
    "HasObservationFeature": {
      "IsLampOn": {
        "HasRangeStart": 0,
        "HasRangeEnd": 1,
        "HasFeatureType": "NOMINAL",
        "HasUnit": ""
        "HasValue": 1
      }
    },
    "HasReward": 0.25,
    "IsGoal": false,
    "IsInitialState": false,
    "IsFinalState": false,
    "HasExpression": "IsLampOn == 0"
  },
  "lastState": {
    "IsLampOn": {
        "HasRangeStart": 0,
        "HasRangeEnd": 1,
```

```
      "HasFeatureType": "NOMINAL",
      "HasUnit": ""
      "HasValue": 0,
      ...
    }
  },
  "lastAction": {
   "Switch_LampOn": {
    "HasText": "Switching a lamp on.",
    "IsNegation": false,
    "HasEffect": {
     "Set_LampOn": {
      "HasImpactType": "ON",
      "HasObservationFeature": {
       "IsLampOn": {
        "HasRangeStart": 0,
        "HasRangeEnd": 1,
        "HasFeatureType": "NOMINAL",
        "HasUnit": ""
       }
      }
     }
    }
   }
  },
  "stateNames": [
   "LampOn"
  ]}
```

A simulator, depending on the task description, can also have multiple agents subscribing to the simulator and vice versa.

4.3.2. Reasoning of States

Internally, the simulator performs two reasoning steps, one for state updates and one for identifying the reward value that depends on the given state and performed action.

The representation of the task instance provides rules, e.g. SPARQL CONSTRUCT query snippets, that allow the simulator to make inferences based on the underlying knowledge base and additional statements that are added to the knowledge base before reasoning and after state feature variables are updated. For example, if we assume a medical use case in which an agent continuously observes a person's vital signs, the person's health state is determined and recognised through the changes of the measured vital signs. Listing 4.4 provides an example that shows RDF statements that might be put to the given knowledge base by the simulator. The RDF statements express that three RDF instances (i.e. *Heartrate, SystolicValue, DiastolicValue*) of type *ObservationFeature* are given. For each observed

vital sign feature a measured value is prevalent. As soon as the simulator receives an action message by the agent, it updates depending on the related effects of the action, single *ObservationFeatures*. For instance, if the agent advises the person to go for a walk, then the corresponding task description might define that the action of going for a walk increases the heart rate and blood pressure values. In this example case, the simulator would increase the corresponding state values and create the following RDF statements to add to the knowledge base. The knowledge base with RDF statements is managed in the random access memory (RAM) and thus, can be updated immediately [159].

Listing 4.4: Measurements expressed as Turtle statements.

```
:Heartrate rdf:type :ObservationFeature.
:SystolicValue rdf:type :ObservationFeature.
:DiastolicValue rdf:type :ObservationFeature.
:Heartrate :hasValue "75"^^xsd:integer.
:SystolicValue :hasValue "155"^^xsd:integer.
:DiastolicValue :hasValue "80"^^xsd:integer.
```

To define rules that describe each state, SPARQL CONSTRUCT queries are a possible means to reason states according to certain state conditions. If the conditions in the *WHERE clause* are met, the triple in the *CONSTRUCT* part applies. Listing 4.5 shows an example *CONSTRUCT* query that suggests that an agent instance perceives the state named *PersonRelaxed*. In the WHERE clause, the first triple indicates that the variable *?agent* is of type *Agent*. Subsequently, the triples for each vital sign indicate that they have a value variable. The function called *FILTER* then defines the conditions, i.e. the value range, for each vital sign variable, while the terms are concatenated by logical *&&* operators. The conjunction by *&&* specifies that all conditions need to be satisfied for the constructed triple indicating the state. The rule expresses that the stated condition exists exactly when the value of the heart rate is between 60 and 70 and the systolic value is between 120 and 140, while the diastolic value should be between 60 and 75. In the case that the instructions previously added to the knowledge base fulfil the query conditions, the simulator derives the new state with the name *PersonRelaxed*.

Listing 4.5: A CONSTRUCT query for defining a state rule.

```
CONSTRUCT {?agent :perceivesState :PersonRelaxed.}
WHERE {
?agent rdf:type :Agent.
:Heartrate :hasValue ?heartrate.
:SystolicValue :hasValue ?systolicValue.
:DiastolicValue :hasValue ?diastolicValue.
FILTER(?heartrate >= 60 && ?heartrate <= 70
&& ?systolicValue >= 120 && ?systolicValue <= 140
&& ?diastolicValue >= 60 && ?diastolicValue <= 75)}
```

4.3.3. Reasoning of Rewards

The second reasoning task that the simulator has to carry out is the derivation of reward values that serve as reinforcing feedback for the agent. The assignment of rewards depends on various aspects, which relate on the one hand to the type of task and on the other hand to the effects that an executed action causes within a given state.

Depending on whether a sequential or non-sequential task is given, the reward function changes. In the design and implementation of this work, it has become apparent that in the sequential case, the rewards of the states have to be different in order for the states to be easily distinguished from each other, as the agent learns the sequence of actions and states with incrementally increasing rewards and has to be enabled to recognise that each correctly executed state maximises its utility. In non-sequential tasks, the agent does not adhere to a strict sequence of actions and states but mainly focuses on the states and actions that promise a high reward. The agent therefore only needs to learn actions that lead to reward-maximising states, whereby the order of execution is unimportant.

However, also penalising values are required, if the agent performs actions that lead to undesired states. Thus, a reward function requires to consider whether a) a task is sequential or not, and b) the given state and performed action lead to a desired, i.e. reward maximising, state.

Rewards are calculated by the simulator in two different ways, either by semantic reasoning using SPARQL queries or by implementing a reward function that extends the reward function API.

When an action is sent to the simulator, it creates an RDF statement about the action and adds it to the knowledge base, similar to the state reasoning discussed earlier. To give an example, the corresponding statement might look as in the following Listing 4.6.

Listing 4.6: Statements about an agent that performs an action.

```
1 ?agent rdf:type :Agent.
2 ?agent :performedAction :AddOneSpoonCoffeePowderToMachine.
3 ?agent :numberActionRepetitions "2"^^xsd:integer.
```

The RDF statement expresses that an agent instance performed 2 times an action called *:AddOneSpoonCoffeePowderToMachine* since actions can repeatedly conducted by an agent. This statement is added to the knowledge base. Subsequently, an appropriate SPARQL query is executed by the simulator that infers the reward value based on the given state and performed action.

Listing 4.7 demonstrates a SPARQL CONSTRUCT query [159] that allows the simulator to derive a positive reward of 1.0, when the statements in the WHERE clause are satisfied. The SPARQL query shows that depending on the number of action repetitions and the

effect of an action, and depending on the state that the agent perceives, a positive reward is returned. In this example, a *UNION* operator is also provided to indicate that only one of the states, either *PersonLovesStrongCoffee* or *PersonIsTired*, need to be true for the constructed and inferred statement to be valid. In this way, the UNION operator makes it possible to consider several states that could be relevant to the induced effect.

The FILTER function allows the simulator to evaluate the number of action repetitions that could affect the impact of an action.

The depicted CONSTRUCT query expresses that if a person loves strong coffee or the person is tired and the agent performs an action that increases the amount of coffee powder, the effect of the performed action to increase the amount of coffee gives a reward value of 1.0.

Listing 4.7: SPARQL CONSTRUCT query for defining a reward rule.

```
CONSTRUCT {:IncreaseCoffeeAmount :hasReward "1.0"^^xsd:double.}
WHERE {
?agent :performsAction ?action.
?agent :numberActionRepetitions ?numRepeat.
?action :hasEffect :IncreaseCoffeeAmount.
{?agent :perceivesState :PersonLovesStrongCoffee.}
UNION
{?agent :perceivesState :PersonIsTired.}
FILTER(?numRepeat < 2)
}
```

Listing 4.8 shows a counter-example. For the same action and effect, the agent receives a negative reward value of -1.0, if the person prefers weak coffee or is already turned up. Both examples demonstrate that it is context- and thus state-dependent whether an action is reward maximising or not. If the conditions in the WHERE clause and the FILTER function are not met, the query will not return a result, i.e. not a constructed statement as defined in the CONSTRUCT clause.

Listing 4.8: SPARQL CONSTRUCT query defining a rule for negative rewards.

```
CONSTRUCT {:IncreaseCoffeeAmount :hasReward "-1.0"^^xsd:double.}
WHERE {
?agent :performsAction ?action.
?agent :numberActionRepetitions ?numRepeat.
?action :hasEffect :IncreaseCoffeeAmount.
{?agent :perceivesState :PersonLovesWeakCoffee}
UNION
{?agent :perceivesState :PersonIsTurnedUp.}
FILTER(?numRepeat > 1)
```

```
10 }
```

Besides SPARQL queries, developers have also the possibility to implement a function that calculates and returns reward values. Algorithm 10 in Appendix A shows a function that exemplary computes rewards for sequential tasks.

Since the properties of a state have also to be updated by the simulation component, the corresponding Algorithm 11 in Appendix B illustrates how state updates are performed.

4.4. Adaptation and Mining of Policies

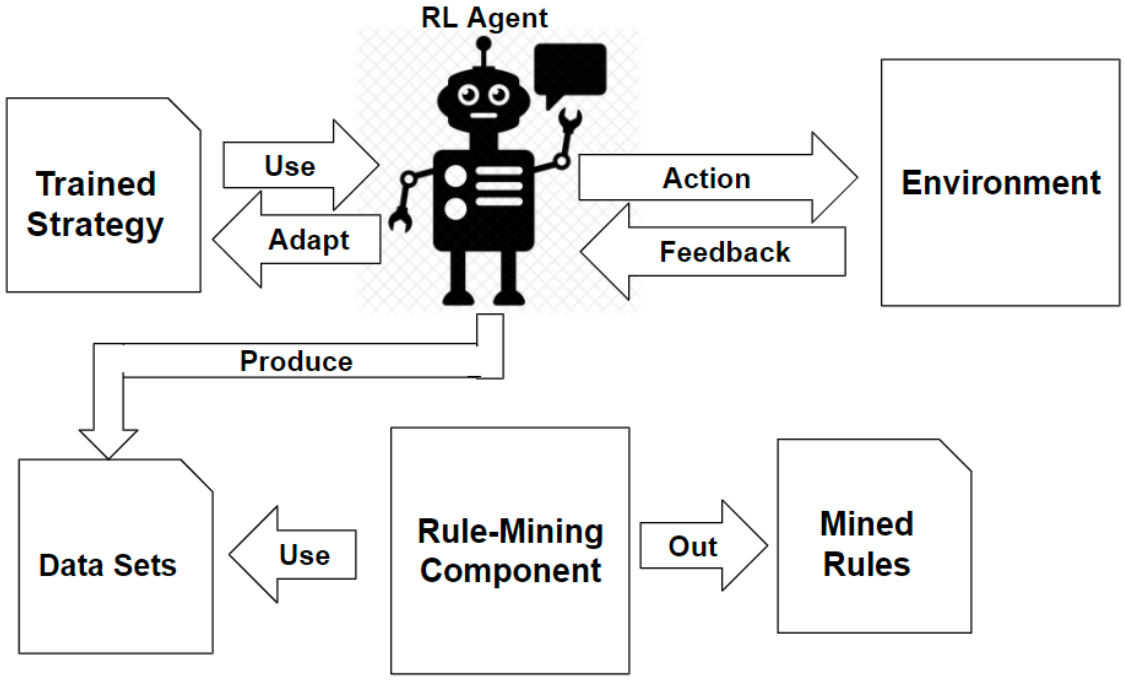


Figure 4.5.: Process flow of the rule mining approach for deriving policy rules.

After RL agents have learned generally applicable default policies, these can be applied in real operations and adapted to user- and environment-specific requirements. Typically, an environment or user provides feedback to the agent, directly or indirectly, about its performed actions. This valuable data can be used to customise or personalise default policies, as individual feedback from the user and environment can be taken into account and used to derive rules that represent personalised policies.

Mostly, datasets are used by machine learning (ML) algorithms to learn representative models that capture and reproduce the patterns of the corresponding dataset. Therefore, ML models usually need to be trained in lengthy data preprocessing and learning procedures.

To avoid lengthy ML pipelines in pattern learning, the proposed agent framework processes task-specific datasets in advance to extract policy-compliant and personalised rules that can be directly adopted and applied by agents without the need to learn policies through elaborate ML algorithms (see Sec. 4.1 and Fig. 4.1).

The approach implemented in the presented agent framework is that RL agents adapt their models (e.g. neural network parameters) directly at runtime by feedback from the environment, collect data and make them available to the framework. By applying different rule mining approaches, the framework derives representative and policy-compliant rules

and makes them available to rule-based agents through a knowledge graph. This enables agents that reuse the rules to perform a task directly.

4.4.1. Selection of Rule Mining Algorithm

Since rule mining algorithms have certain prerequisites that have to be met by the underlying data, the framework has to be able to decide when to apply which rule mining approach.

Thus, the proposed agent framework considers task properties to determine which data is given and which rule mining algorithms to apply. In Section 4.2 it was discussed which properties (e.g. (non-)sequential, feature type) distinguish tasks from each other. Table 4.16 gives an overview and lists which algorithms are applied based on the corresponding task's properties. In a task that requires a fixed sequential order of actions and whose state representation consists of categorical features, the algorithm *FPGrowth* (see Section 2.4.1) is applied in combination with Algorithm 12 and Algorithm 7, since the latter algorithm makes it possible to consider and maintain the order of actions underlying the dataset under consideration. If numerical features are also present in a sequential task, a transformation, i.e. binning, into categorical features can be performed.

For non-sequential tasks whose execution does not depend on a specific action sequence, the feature types of the prevailing states determine whether the *OneR* (see Sec. 2.4.3) or *sequential covering* (see Sec. 2.4.4) algorithm is applied. For example, the OneR algorithm can only process categorical features, which requires the conversion of numerical features into categorical ones, whereas sequential covering can process both categorical and numerical features because it mines rules that may consist of terms describing feature value ranges, e.g. $age > 18 \wedge age \leq 30$.

Sequential covering is applied in this framework in combination with a *classification and regression tree (CART)* implementation. Based on the underlying dataset, the CART determines the rules that describe the dataset while the sequential covering algorithm partitions and stores the rules that reproduce the data samples in the dataset.

Whether OneR or sequential covering is the best choice depends also on a) the prevalence of well-separating features and b) how fine-grained and accurate the decision rules require to be. If not many features for decision making have to be considered by the agent, OneR might be the choice while when the agent has to take multiple relevant conditions into account, sequential covering might be the more suitable algorithm. The agent framework executes for non-sequential tasks both algorithms and thus, the choice which decision rules to take are left to the decision of the agent that requests the rules.

4.4.2. Dataset Structure of Input

A data sample (D) that represents feedback and serves as input for the rule mining algorithms is defined by the following set of features: D := {OxAxR}, where O stands for observations, A for performed actions and R for received reward values. A derived rule is

Action Order	Categorical Features	Rule Mining Algorithm
Sequentially Ordered	Categorical	FP-Growth
Unordered	Categorical	OneR
Unordered	Categorical/Numerical	Sequential Covering + CART

Table 4.16.: Selection of rule mining algorithm based on task characteristics.

structured as in Eq. 4.5. The agent's observations define a state (see Eq. 4.4) and represent the *antecedent* that is followed by the *consequent*, i.e. action, of a rule, see Eq. 4.5.

$$\text{State}_n := \{o_1, o_2, ..., o_n\} \tag{4.4}$$

$$\text{Rule}_n := \{o_1, o_2, ..., o_n\} \sqcap \{a_n\} \tag{4.5}$$

4.4.3. The Relation between Data Samples and Rule Items

The UML[2] class diagram (see Fig. 4.7) shows a different perspective of elements that make up a dataset for the rule mining approach presented. The diagram represents that a dataset is composed by an arbitrary number of data samples. Each data sample consists of a reward value and an arbitrary number of rule-items, where each rule-item aggregates exactly one action and one state. The state, in turn, aggregates observable features that have a key string and a measured value.

The reason why there can be an arbitrary number of rules for each sample is that a task can be divided into stages, which can contain a sequence of actions that require first to be executed in order to complete the corresponding stage and thus achieve a partial goal of the task. An example of such a task with several stages and sub-goals would be, for example, the completion of an online order, in which different stages (e.g. consecutive views) have to be processed one after the other, with several action steps per stage, until the order is completed. Figure 4.6 illustrates an example of a task with three stages and corresponding sub-goals, where the agent is required to perform a sequence of actions in each stage (e.g. enter destination, enter flight date, press search button, etc.) to complete the corresponding stage and move to the next stage. The rule mining algorithms thus need to be able to identify action sequences that occur within each task stage. The presented structuring allows the processing of the datasets by the considered algorithms. An agent that wants to provide its own collected datasets have to follow this data structuring.

4.4.4. Data Preprocessing and Rule Mining Algorithms

The mining of process datasets is divided into two parts. First, a data pre-processing step is performed that filters the data for the corresponding rule mining algorithm based on the existing reward values. This requires a threshold for the rewards, which determines which samples are selected and which are filtered out. In the second step, depending on

[2] Unified Modelling Language

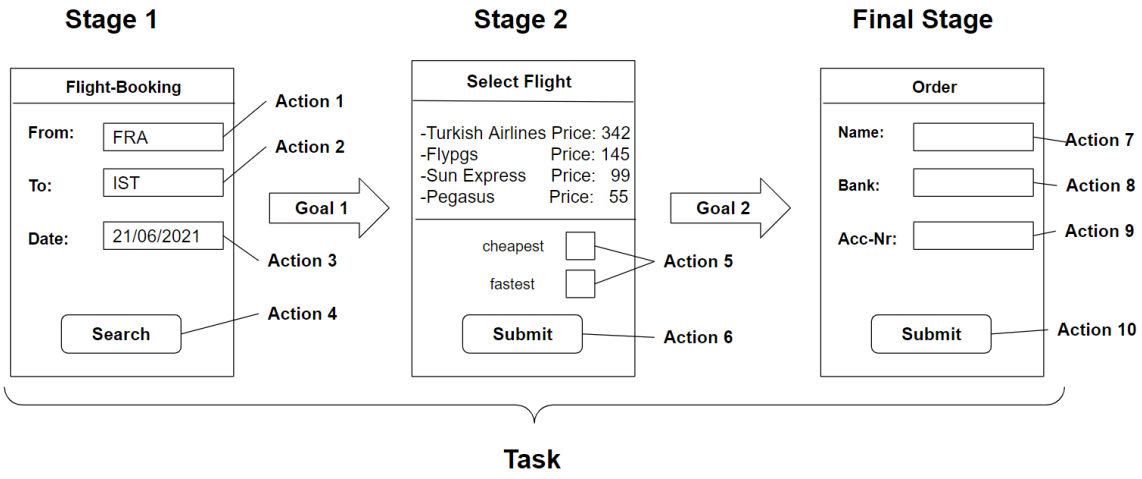


Figure 4.6.: Stages, goals and actions of a flight booking task.

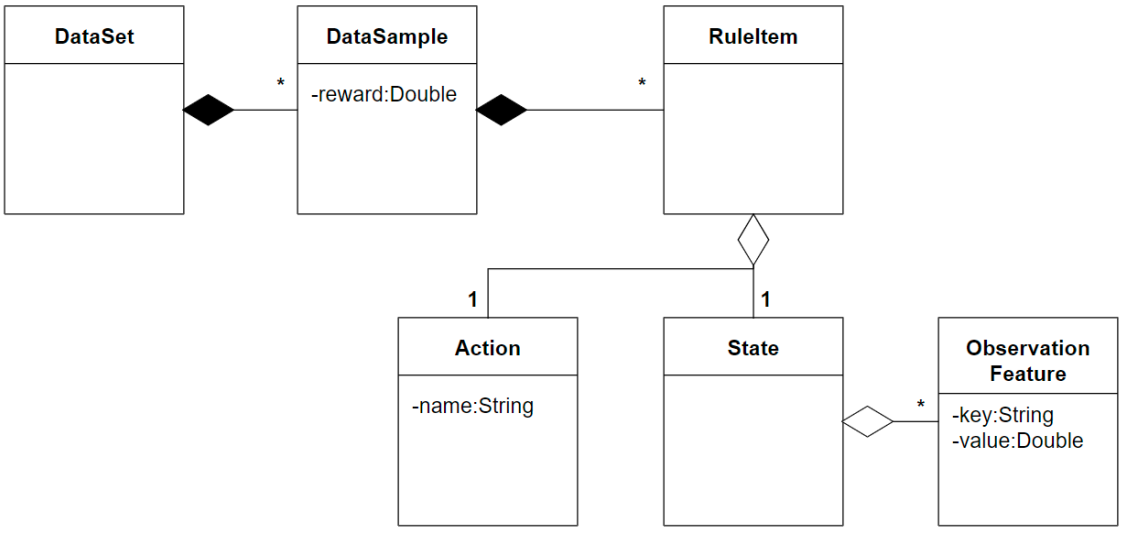


Figure 4.7.: UML class diagram representing the datasets that serve as input for the discussed rule mining algorithms.

the task properties, the framework selects the rule mining algorithm and applies it to the pre-processed data samples. The result of the rule mining algorithm is a set of rules consisting of a state (antecedent) and a corresponding action (consequence) (see Eq. 4.5).

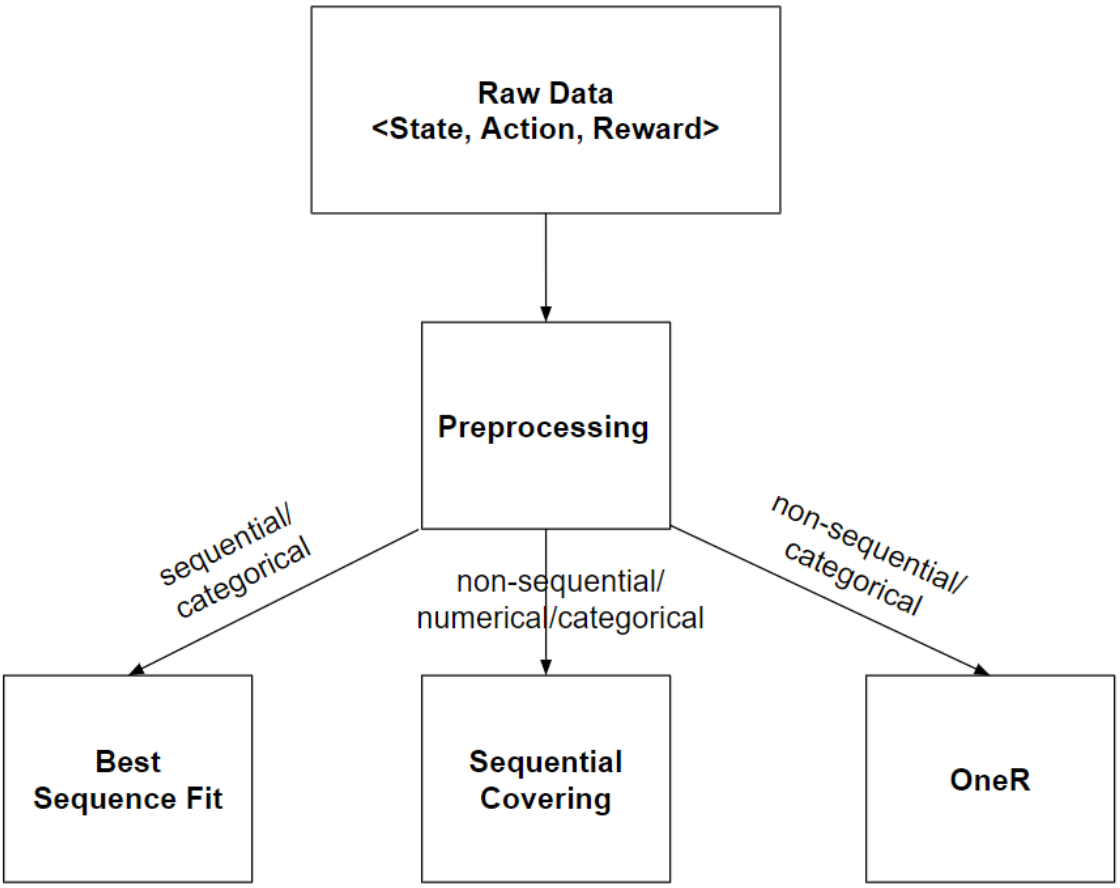


Figure 4.8.: The automated selection of the rule mining approach.

4.4.4.1. Data Preprocessing

In the preprocessing step (see Algorithm 12 in Appendix C), the FPGrowth algorithm is applied whereas the implemented algorithm distinguishes between sequential and non-sequential tasks. This data preprocessing step is required in order to reduce the amount of collected data and gain frequent item sets that are more likely to represent a rule and can be further processed in order to mine representative decision rules as outlined in the next subsection.

4.4.4.2. Rule Mining Algorithms

In Section 2.4, several algorithms were considered and adopted for the purpose of rule mining. In addition, an algorithm called *Best Fit Sequence* was devised and implemented specifically for sequential task datasets as part of this work and is presented in this section.

Since FPGrowth, OneR and Sequential Covering are adopted respectively reused algorithms and have already been presented in the foundations chapter, their application is only briefly discussed in the context of the agent framework.

FPGrowth For sequential tasks that require sequentially ordered actions, the FP-Growth algorithm that extracts frequent item sets was adopted and implemented because it is much faster than the well-known *A-Priori* algorithm and can process a large amount of data in a very short time [91]. FP-Growth is used as a preprocessing step in the agent framework to find frequent state-action combinations in the dataset and reduce the amount of data significantly since only frequently occurring samples are considered that also contain an action.

OneR is a simple decision rule mining algorithm that was first presented in the publication [106] of Holte. It extracts a representative and well-separating feature from the dataset. Given a dataset with observations and actions, the goal in the scope of the agent framework, is to find an observation feature that can be used to identify the most promising action. To find the most significant feature and the corresponding rules (i.e. feature-value pairs), the prevalence of the predictive classes is considered in a cross-table. For each feature-value pair, the most frequent prediction class is considered the correct prediction class, while the co-occurrence of other prediction classes together with the corresponding feature-value pair is considered a classification error. The cross-table counts these classification errors and allows the feature with the lowest classification errors to be selected as a representative rule candidate. A prerequisite for the application of the OneR algorithm is the conversion of numerical data (e.g. by binning) into categorical data. The advantage of OneR is its simplicity (e.g. one representative feature that determines the rules) and its comprehensibility through short rules, especially in the context of explainable ML.

Sequential Covering is a algorithm that follows the principle of *divide and conquer*. The implementation in the presented approach is based on Molnar's [173] description of the algorithm. The algorithm repeatedly eliminates rules represented by data points from the mined dataset and simultaneously maintains a rule list with learned rules that are indicated as significant. The appropriate rules are determined by another algorithm that allows to learn specific rules from the given dataset (see example in Fig. 4.9).

The agent framework implements a *classification and regression tree (CART)* as rule-learning algorithm for the sequential covering algorithm. Thus, it is not required to bin the data samples for sequential covering, since the CART algorithm learns rules that represent among categorical conditions (e.g. colour = brown) also numerical range conditions (e.g. heart rate > 120).

To build a rule expression, the algorithm starts from the tree's leaf node that features the highest accuracy and traverses backwards all subsequent nodes until the root node is reached. The most prevalent class (i.e. action) and corresponding rule is specified as default rule, that applies if no other rule applies within the maintained rule list. In

order to create a rule for each prevalent action, the algorithm concatenates the single interim node expressions maintained in the rule-list with AND-operators to create the antecedent, so that each predictable action (i.e. consequent) has exactly one rule for decision making. The traversal of the example tree in Fig.4.9 results in the derived rule: IF Infected $>$ 500 $\sqcap$ AvailableBeds $\leq$ 5% $\Rightarrow$ A1. The yellow nodes represent the path that leads to the action (A1) with the highest classification accuracy.

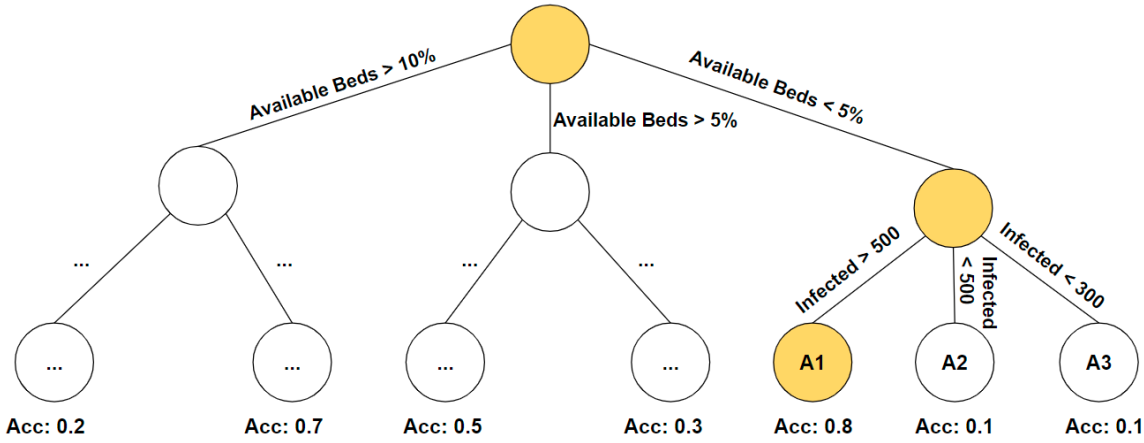


Figure 4.9.: Example of rule-learning from CART.

Best Fit Sequence In the scope of this work, an approach, i.e. algorithm, called *Best Fit Sequence (BFS)* was developed for sequential tasks with categorical data. The aim of the algorithm is to find for each stage of a task by means of logged process datasets, an ordered sequence of actions that follow matching rule conditions. The actions and corresponding rule conditions are derived from frequent item sets that are previously obtained through the FPGrowth algorithm and datasets that reproduce task activities and corresponding outcomes. Such datasets are usually acquired through imitation or demonstration learning and incorporate sensor observations, performed actions and feedback, e.g. reward values, assigned by domain experts or obtained by user feedback.

The proposed algorithm *Best Fit Sequence* is implemented through three different functions, which are presented in Algo. 12 in Appendix C, Algo. 6 and Algo. 7, 8.

Fig. 4.10 illustrates the data and process flow from function to function. The chronologically collected data samples, as described in Fig. 4.7, are first preprocessed by Algo. 12. The results of the preprocessing step are a cleaned dataset of positively rewarded actions and a collection of actions ordered by their occurrence in the dataset, as the datasets maintain a specific order of actions performed that resulted in a positive reward. The preprocessed samples serve as input to the *FPGrowth* algorithm, which creates frequent item sets that contain both observed features and one action per sample. Based on these frequent item sets, the *createRuleExpressions* function is used to create concatenated rules, each of which derives an action. Based on the rule expressions, action sequences, and state observations

received at runtime, the algorithm *Best Fit Sequence* derives action sequences for each stage of a sequential task.

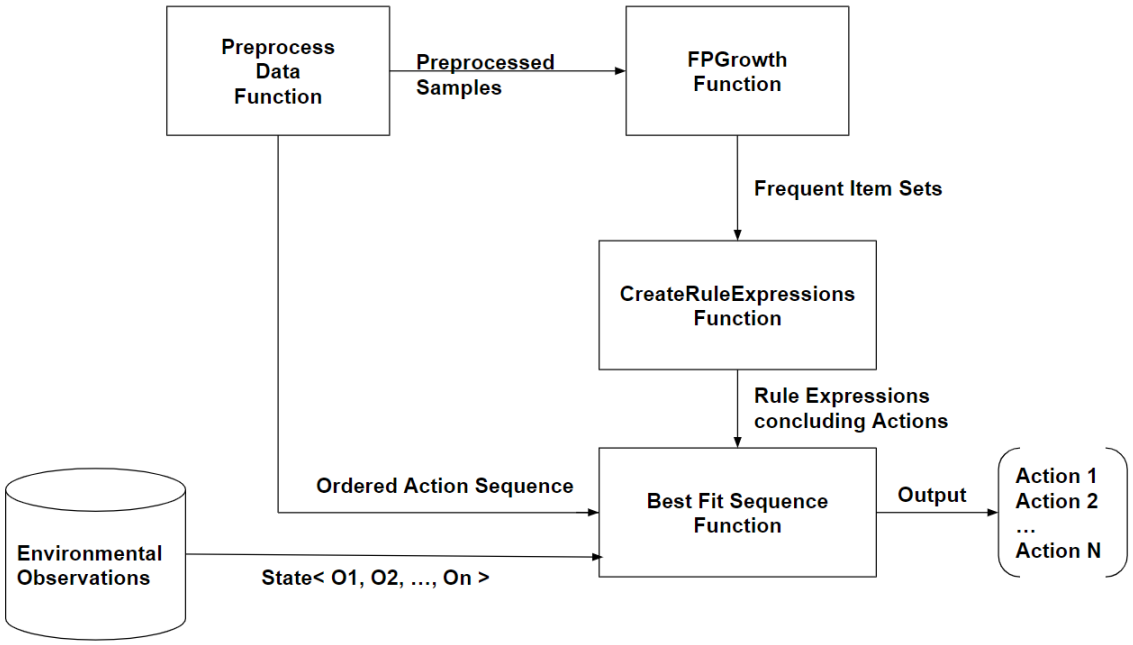


Figure 4.10.: Data flow between the presented function. First, the collected data samples are preprocessed, then item sets and ordered action sequences are derived and based on these frequent item sets and ordered action sequences the *Best Fit Sequence* algorithm derives for each sequential task stage the corresponding action sequence.

The concrete idea of the algorithm is to evaluate a set of rules T. The rules that derive possible actions for each observed state of a set S with a metric, are formed from frequent item sets. The metric measures how many of the given state features and rule items or expressions match by some length L. As already mentioned, a state is represented by observed features and their given values, whereas rule expressions describe state feature expressions as preconditions for performing an action. Thus, the more conditions (i.e. expressions of a rule) and state observation features are fulfilled, the higher the probability that an action derived by the corresponding rule is specific to a state and will accomplish the state requirements. The metric thus measures how many state conditions match the expressions of the respective action rule.

The metric takes into account the length $R = |T|$ of the rule, i.e. the number of concatenated expressions, each of which describes exactly one observed feature of the state, as well as the number of state features $L = |S|$ that correspond to the conditions of the rule. However, if an observation feature occurs in several expressions of a rule, only one of the expressions is considered for the length specification L of the rule. Eq. 4.6 defines the sets T and S of rules and states.

$$\begin{aligned} \text{Expression} &:= \text{feature}_n == \text{value}_n \\ \text{Rule} &:= \text{expression}_1 \wedge \text{expession}_2 \wedge ... \wedge \text{expression}_r \rightarrow \text{action}_m \\ T &:= \{\text{rule}_1, \text{rule}_2, ..., \text{rule}_n\} \\ S &:= \{\text{feature}_1 = \text{value}_1, \text{feature}_2 = \text{value}_2, ..., \text{feature}_l = \text{value}_n\} \end{aligned} \tag{4.6}$$

Depending on whether the number of rule expressions, i.e. $L < R$, is larger or the number of a state's feature values, i.e. $R < L$ is larger, a distinction, i.e. selection, is made between two metrics that calculate the match score of the rule. In the case that $L = R$, no distinction is made and either metric can be used for calculating the rule-state matching score.

Only the rule expressions that describe features in the state are then considered, while the additional expressions are considered undefined and are not included in the rule score calculation. Eq. 4.7 shows the corresponding calculation formula if the number of rule expressions is greater than the number of feature values of a state. Eq. 4.8 shows the corresponding calculation formula if the number of features of an observed state is greater than the number of distinct rule expressions.

The metrics take into account that the more rule conditions fulfil the observed state characteristics, the more specific and representative a rule is for a state.

$$\text{Score}_{L<R}(T, S) := \frac{1}{L} \sum_{\substack{t \in T \\ f \in S}} k, \;\; k = \begin{cases} 1 & \text{t matches f} \\ 0 & \text{otherwise} \end{cases} \tag{4.7}$$

where L is the total number of features a state consists of, t is an expression of the rule T that either satisfies or does not satisfy a feature instance f of the state S and k is a constant that is either 1 if the considered rule expression and the state feature match, or 0 if both do not match.

$$\text{Score}_{R<L}(T, S) := \frac{1}{R} \sum_{\substack{t \in T \\ f \in S}} k, \;\; k = \begin{cases} 1 & \text{f matches } t \\ 0 & \text{otherwise} \end{cases} \tag{4.8}$$

where R is the total number of expressions a rule consists of, t is an expression of the rule T that either satisfies or does not satisfy a feature instance f of the state S. k is again a constant that is either 1 if a rule expression and state feature match or 0 if both do not match to each other.

In both metrics a rule is considered a match to an observed state and its features only if the obtained rule score is 1.

The input parameters of the function *CreateRuleExpressions()* (see Algo. 6) are the previously preprocessed frequent rule item sets, each consisting of an action and a set of

features and their corresponding observed values. The output of the function includes a list of rules that are processed in the subsequently executed functions.

The function starts with variable declarations for arrays and objects that store the intermediate results and the outcomes of the function **(line 2 - 4)**.

Then a *for-loop* goes through all the frequent samples and creates a rule object for each sample containing all the associated features and their values and assigns them to the corresponding action **(line 5 - 16)**.

For each action, the corresponding features are stored in a hash map called *ruleExpressions* **(line 13 - 16)**.

A third *for-loop* removes all objects with actions that do not reference feature-value pairs from the *ruleExpressions* map **(line 17 - 19)**.

Finally, the function returns an array of stored rule elements, i.e. rule expressions, containing a list of arbitrary feature-value pairs for each action **(line 20)**.

Algorithm 6: CreateRuleExpressions

```
   Input: Map<K,V> frequentSamples
   Output: Map<K,V> ruleExpressions
1  Function CreateRuleExpressions(frequentSamples):
2      ruleExpressions = Map()
3      actions = []
4      forall sample IN frequentSamples do
5          rule = Map()
6          action = undefined
7          forall expression IN sample.rule.items do
8              if expression !== "Action" then
9                  rule[expression] = sample.rule.items[expression]
10             else
11                 action = sample.rule.items[expression]
12         if action NOT IN ruleExpressions then
13             ruleExpressions[action] = []
14         if Object.keys(rule).length > 0 && rule NOT IN ruleExpressions[action] then
15             ruleExpressions[action].push(rule)
16     forall exp IN ruleExpressions do
17         if ruleExpressions[exp].length <= 0 then
18             delete ruleExpressions[exp]
19     return ruleExpressions
```

With the array of rule objects generated, the function *BestFitSequence* is now able to obtain action sequences for any given task stage by applying the previously introduced metrics (see Eq. 4.7 and Eq. 4.8). The required input parameters of the function are the previously created and provided array of rule objects consisting of actions and related feature value pairs and the given state that an agent, within a task stage, perceives. Based on the observed state and the rule objects the algorithm determines a sequence of actions that fulfil the completion of the current task stage.

Therefore, a *for-loop* passes through the hash map containing all action objects with their associated rule items and generates an expression per rule item consisting of the given state and the given rule **(line 3 - 14)**. Inside the *for-loop*, a variable called *ratingCounter* is declared and initialised to 0 **(line 4)**. This variable is incremented by 1 each time the current features representing the given state match the associated conditions of the rule expression evaluated in the function *eval()* **(line 15 - 16)**.

Listing 4.9 shows an example expression, i.e. statement, consisting of the observations made by the agent at runtime **(line 1-2)** and a rule **(line 3)** derived and constructed in Algo. 7, which is passed to the function *eval()*. The result for this expression would be true because the two observed feature expressions match the conditions of the following rule expression, which consist of two terms concatenated by logical && (AND) operators.

Listing 4.9: Observed state and derived rule defined in JavaScript.

```
1 let PushButtonClicked = 0; # State observation
2 let DateEntered = 1; # State observation
3 PushButtonClicked == 0 && DateEntered == 1; # Constructed rule from data samples
```

For each true statement the *ratingCounter* variable is incremented **(line 16)** to determine how many conditions are met by the given state because the higher the number of conditions that match the given state, the more representative and state-specific a rule is. In the next step, the current action of the considered rule is stored in an array called *actions* **(line 18)**, if the action is not yet prevalent within the array **(line 17)**. The obtained number of matches is then incremented in the array named *actions* by adding the value of the rating counter, which indicates how many rule conditions related to an action occurred and matched the currently considered state **(line 19)**.

The array named *actions* is sorted in descending order based on the obtained rule scores **(line 20)**. The action whose rules yield the highest score is selected as it is assumed to be the most state-specific **(line 22 - 24)**. However, it may happen that rules yield equal score values. This means that these rules and their corresponding actions have the same *relevance* for the currently considered state and the order between these equal relevant actions does not matter and is interchangeable. In these cases, all actions that have the same rule score are pushed into the array of actions to be performed **(line 25 - 26)**. In the second and last part of the algorithm, the sequence of actions for each task stage is

Algorithm 7: BestFitSequence - Part 1

```
Input: Map<K,V> ruleExpressions, orderedActionSequences[], Map<K,V>
       currentState
Output: sequenceOfActions[]
Function BestFitSequence(ruleExpressions, orderedActionSequences, currentState):
    actions = []
    forall action IN ruleExpressions do
        ratingCounter = 0
        forall expression IN ruleExpressions[action] do
            expr = “
            rule = “
            forall feature IN expression do
                if feature IN currentState then
                    expr += ‘${feature} = ${currentState[feature]}; ‘
                    rule += ‘${feature} == ${expression[feature]} && ‘
            rule = rule.substring(0, rule.lastIndexOf(’&&’)).trim()
            rule += ‘;‘
            statement = expr + rule
            if eval(statement) then
                ratingCounter += 1
                if actions.length <= 0 || actions[actions.length - 1][0] !== action then
                    actions.push([action, 0])
        actions[actions.length - 1][1] += ratingCounter
    actions.sort((a, b) => b[1] - a[1]
    obtainedSequence = []
    forall i = 0, j = 1; i < actions.length; i++, j++ do
        if i == 0 then
            obtainedSequence.push(actions[i])
        if j < actions.length && actions[j][1] == actions[0][1] then
            obtainedSequence.push(actions[j])
        else
            break
```

determined. First, it is checked whether the determined sequence of actions contains more actions than only 1 **(line 30)**. If the condition is not fulfilled, i.e. only one action was found, the determined action is assigned to the variable named *sequenceOfActions* **(line 43)**. However, if the condition is met and multiple actions were found, a *for-loop* will run through the hash map called *orderedActionSquences*, which was created during preprocessing of the raw dataset and which contains all the non-identical action sequences found in the dataset **(line 31 - 40)**.

A variable called *matcher* counts how many of the actions received, match the actions in the ordered action sequence **(line 32, 36)**. If the number of matches is equal to the length of the ordered action sequence, the sequence is stored in the array called *seq* along with its sequence number **(line 39)**.

Subsequently, the outer for-loop is exited by a *break* instruction, as exactly one matching action sequence is required for a task stage **(line 40)**. Next, the sequence of actions obtained is sorted in ascending order by the sequence number, since the order of actions is to be maintained **(line 41)**.

Finally, the action or sequence of actions obtained is assigned to the variable named *sequenceOfActions*, which is returned to the caller of the function **(line 41, 43, 44)**. The

Algorithm 8: BestFitSequence - Part 2

```
29 seq = []
30 if obtainedSequence.length > 1 then
31     forall sequence IN orderedActionSequences do
32         matcher = 0
33         if Object.keys(sequence).length >= obtainedSequence.length then
34             forall obtSeq IN obtainedSequence do
35                 if obtSeq[0] in sequence then
36                     matcher += 1
37             if matcher == obtainedSequence.length then
38                 forall act IN sequence do
39                     seq.push([act, sequence[act]])
40                 break
41     sequenceOfActions = seq.sort((n, m) => n[1] - m[1])
42 else
43     sequenceOfActions = actions
44 return sequenceOfActions
```

Best Fit Sequence algorithm is applied during task execution and thus executed directly in real-world operations to obtain action sequences based on datasets collected at runtime

through user feedback or in advance through demonstration learning.

Listing 4.10 shows an example of a possible end result of the algorithm serialised in JSON format, consisting of action names that refer to a list of state features respectively conditions that require the corresponding action. The keyword *default* indicates whether the action will be executed by default if no state and thus no suitable action match the observed state of the agent. It can be seen that a rule expression consists of expressions linked by ampersands representing logical *And* operators. Each expression is made up of a feature name and a corresponding assigned value.

Agents can parse this JSON representation containing the rules and compare their observed feature representations with the conditions in the rule expressions and, in case of a match, perform the corresponding action.

Listing 4.10: Action rules.

```
{
"Find_textbook": {
   "default": false,
   "rules": [
    {
      "rule": "IsWalk_living_room == 1 && IsWalk_textbook == 1
      && IsFind_textbook == 0 && IsGrab_textbook == 0
      && IsWalk_chair == 0 && IsFind_chair == 0
      && IsSit_chair == 0 && IsRead_textbook == 0"
    }
   ]
  },
  "Grab_textbook": {
   "default": false,
   "rules": [
    {
      "rule": "IsWalk_living_room == 1 && IsWalk_textbook == 1
      && IsFind_textbook == 1 && IsGrab_textbook == 0
      && IsWalk_chair == 0 && IsFind_chair == 0
      && IsSit_chair == 0 && IsRead_textbook == 0"
    }
   ]
  },
  ...
}
```

4.5. Retrieval of Policies

As described in Sec. 2.2, the Semantic Web is a networked infrastructure that enables the structured exchange of knowledge through Linked Open Data (LOD). The agent framework uses the Semantic Web stack to enable computational agents the sharing and retrieval of meta data about tasks and policies, that have proven successful in solving a corresponding task. During the life cycle of the agent framework, one aim is to provide already learned default policies, that can be adapted and reused by agents to perform their own task for which they are responsible.

An important aspect of this is to find suitable policies that allow an agent to complete its task successfully and in a reward-maximising manner until all target states and the terminating final state are reached.

The agent framework thus implements two methodologies that a) make tasks numerically representable and comparable in terms of their similarity and b) allow the combination of policies based on the current context, i.e. state, of the acting software agent. In the following sections, the mentioned methodologies for finding and composing suitable policies are discussed.

4.5.1. Retrieval of Reusable Task Policies

To assist agents in finding suitable, i.e. reusable, task policies, the proposed framework determines the semantic similarity between already existing task instances and the agent's provided semantic task instance, as it is assumed that similar tasks require similar policies for their execution. If several similar tasks are found by the agent framework, the requesting agent reuses the policies of the most similar task and adapts them as necessary to solve its own task.

Tasks and their properties are formally represented by semantic task entities that adhere to the task ontology as outlined in Sec. 4.2. These task entities are transformed into a numerical representation in the framework as soon as a task entity is created and referenced in the corresponding knowledge graph by training an entity embedding vector for each newly created task instance. Entity embedding vectors and their properties have already been introduced in Sec. 2.3.12. The use of entity embedding vectors allows the framework to represent semantic properties or similarities of task instances in numerical form in an embedding vector space. This numerical vector representation is used to calculate the similarity or spatial distance of two tasks using common metrics, such as *Cosine distance* (see Eq. 2.38) or *Euclidean distance* (see Eq. 2.39).

In the context of the agent framework, entity embedding vectors are trained by a deep neural network (DNN). However, it would also have been possible to represent task instances numerically by one-hot encoded vectors. However, it turns out that the one-hot encoding approach has two disadvantages, which are overcome by entity embeddings. The first disadvantage is that the dimension of one-hot encoded vectors can easily become

unmanageable when the number of unique tasks becomes very large, since the vector size has to be increased for each new task. The second disadvantage is that one-hot encoded vectors do not take into account the semantic similarity or spatial distribution of tasks, as the representation says nothing about the spatial arrangement of the task within a vector space. Thus, no conclusions can be drawn about the semantic relatedness and similarity of tasks. Since the framework aims at finding similarities between tasks, it makes sense to utilise entity embeddings for representing task instances and their related properties. Figure 4.11 illustrates the process flow for encoding and storing task embedding vectors.

The idea of using entity embeddings to calculate task similarity and recommend the most similar policies was inspired by the case studies in the *Deep Learning Cookbook* [187].

Entity Embeddings are used in the scope of the agent framework in two application scenarios that are a) finding similar, i.e. suitable task policies (see Sec. 4.5.1), and b) composing policies based on the agent's context, i.e. state (Sec. 4.5.2).

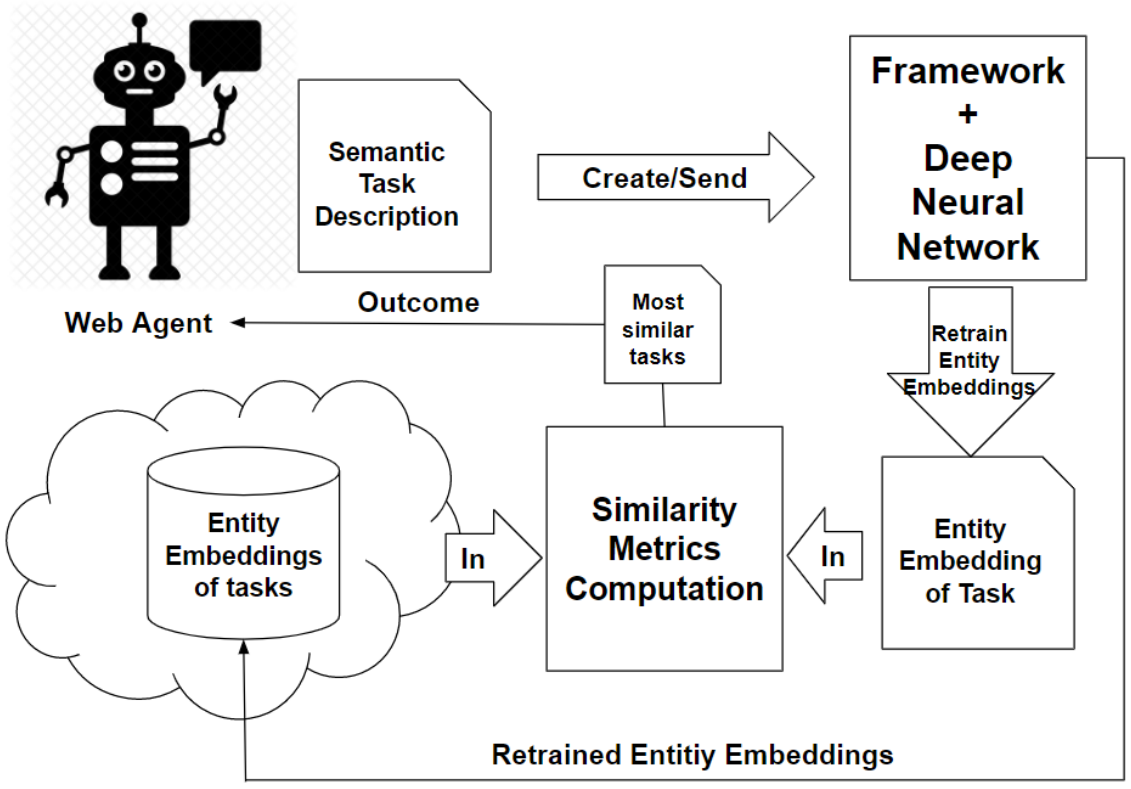


Figure 4.11.: Retrieval of reusable policies by Entity Embeddings. An agent sends its task entity description to the agent framework. The corresponding DNN is retrained with the new task entity description. The encoded entity embedding of the new task is stored in the database and used to find the most similar task embeddings in the database using similarity metrics. The top-n of the most similar tasks are returned to the requesting agent who had provided the new task entity description.

To train task specific entity embeddings, a DNN with 5 network layers is implemented in the agent framework. The corresponding DNN topology is depicted in Fig. 4.12. The first layer represents the input layer. Each task is represented by an index number that serves as the input for the DNN. The approach requires to train entity embedding vectors also for linked semantic action entities, because each task strategy ultimately consists of a sequence of actions and the more actions that occur together in two compared tasks, the more similar the required policies are likely to be. The connection of actions and tasks through trained embedding vectors in turn means that a binary classification problem is given, since the DNN trains whether a task and action appear together. The assumption made is that if a task and an action are related, they are closer together in the embedding vector space than actions and tasks that have no relationship to each other, and tasks that have several actions in common with other tasks are semantically closer to each other than tasks that have no overlap in terms of their actions.

The inputs, i.e. task indices, are provided in batches that contain positive and negative data samples for the output of the DNN. A *negative* data sample means in this context that the intended task entity has no property relation to a considered action entity. Vice versa, a *positive* data sample means that the task entity references the corresponding action.

The most important layer is the second layer, whose weights represent the entity embedding vector of a task in a multi-dimensional vector space, which are used to determine the similarity of tasks and thus policies. The embedding layer can include any number of units for each task and action. However, experiments have shown that 50 units are sufficient to achieve promising results.

The third layer is a *dot product* layer that merges the embedding weights of tasks and actions. The fourth layer reshapes the dot products of the embedding weights to a single number, so that a *Sigmoid* activation function that leads to the binary output of the DNN, can be applied. The output indicates whether a task and action occur together. The adopted optimisation algorithm then adjusts the weights of the embedding layers.

Due to its advantages, the optimisation algorithm used is *Adam Optimiser* [120] by Kingma et al. which is an extension of the stochastic gradient descent optimisation. The advantages are stated in the corresponding paper [120] of Kingma et al. In addition, *Binary Cross Entropy* is suitable as a loss function because the co-occurrence of tasks and associated actions is trained, resulting in a binary classification problem. However, for classification problems of a different nature, other loss functions may be more appropriate.

The most important result of the DNN training is the trained weights of the embedding layers for each task. To determine the similarity of tasks, it is then necessary to apply a similarity or distance metric on compared entity embedding vectors. In this case, as discussed earlier, *Cosine distance* and *Euclidean distance* are used as both are common metrics that give equal reliable results.

To retrieve suitable policies for its own task, an agent requires to provide a semantic task entity representation that provides at least the name of the task and its property relations to the considered actions. The DNN then computes the corresponding embedding vector of the considered task and retrieves the most similar tasks as previously outlined. The tasks found are sorted and returned according to their decreasing similarity.

The trained weights of the task's embedding layer are stored in an accessible knowledge base and linked to the corresponding semantic task entity in the knowledge graph.

A limitation of this methodology is that tasks are required to have a known, i.e. intersecting, state and action space. However, this is usually given because in a corresponding knowledge graph, state and action entities are shared between tasks. Moreover, the knowledge graph can be extended with new entities, and a mapping between different state and action spaces is also possible through, e.g. *owl:sameAs* relationships, between semantic entities of different knowledge graphs that exhibit related concepts and entities.

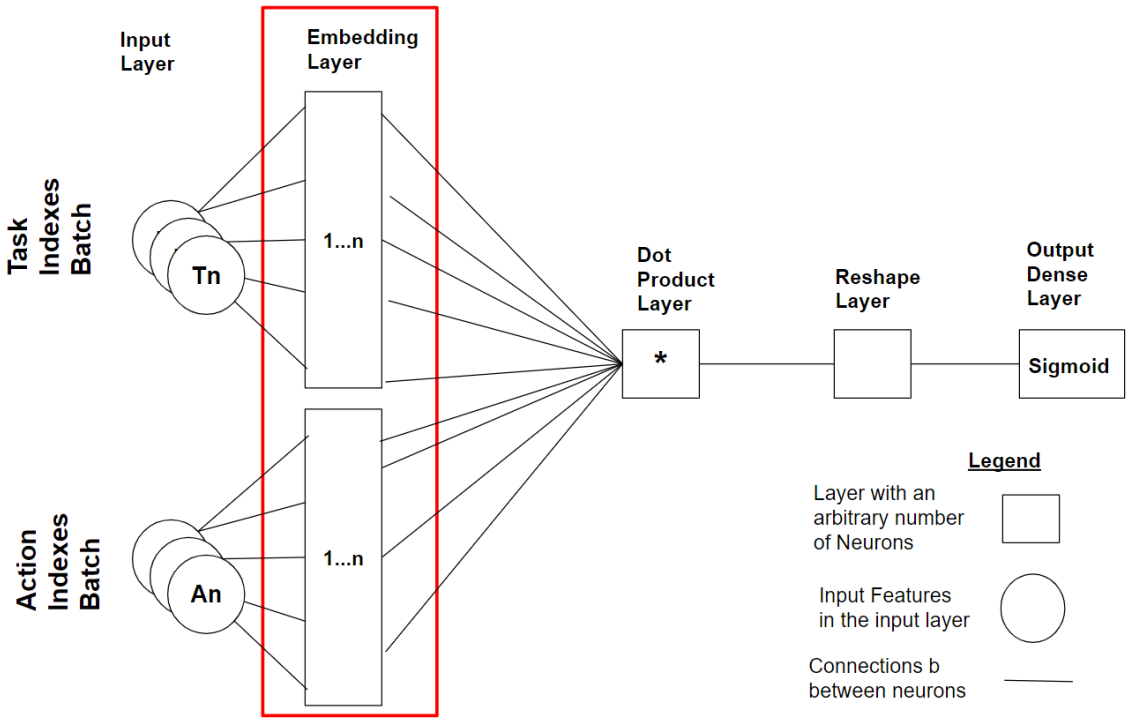


Figure 4.12.: Deep Neural Network Architecture for training task and action embeddings.

4.5.2. Context-Dependent Combination of Task Policies

The approach presented in this section as well as in Sec. 5.4 and Sec. 6.3 stems from a recently accepted journal paper[3] [152] entitled *"Context-Aware Composition of Agent Policies*

[3] https://semantic-web-journal.net/content/context-aware-composition-agent-policies-markov-decision-process-entity-embeddings-and-0

by Markov Decision Process Entity Embeddings and Agent Ensembles"[4]. The corresponding paper is also published as a pre-print on Arxiv[5][153]. Additional material to the paper such as the developed and evaluated ontology, datasets and source code are documented and openly available on Github[6] and Figshare[7] [154].

The context, i.e. the (internal and external) state, of an acting agent determines which actions should be performed. However, sometimes *one-fits-all* solutions to policies are not sufficient, and since tasks can be performed in different, i.e. alternative, ways, pre-trained task policies may not always match contextual requirements one-to-one. In addition, agents, (e.g. in service robotics or World Wide Web contexts), sometimes need to work in stochastic environments and across tasks, so their contexts can change frequently during their runtime and they need to react to this quickly and act correctly. The solution to such a problem is addressed by the agent framework through the approach presented here that enables the flexible and context-aware combination of policies without the requirement that agents train policies or know which activities or tasks to perform in advance. In this use case scenario, the agent only observes and knows its current state, while possible tasks or activities of the agent are determined only by its context. The agent requests from the agent framework the best actions to perform in its current state. Since alternative actions may be useful in a given context, the framework determines the most context-appropriate, i.e. the most useful, actions in a context and informs the agent about them.

The proposed approach allows a) to restrict the state- and action search space, b) to avoid, in contrast to e.g. reinforcement learning, lengthy training procedures for learning policies and c) to anticipate likely contextual states and alternative actions that may occur in specific task contexts. This enables agents to act flexibly and across contexts to complete their tasks in multi-context environments.

The proposed approach combines two methods. One method enables context representation through entity embeddings and the second method, implemented by an algorithm, enables the search for the nearest state-related actions. However, before discussing the two methods, the application scenario and process flow of the *policies combination* approach are introduced (see Sec. 4.5.2.1).

4.5.2.1. Application Scenario

Fig. 4.13 illustrates the application scenario of the approach, which consists of 12 process steps. The general use case foresees that one or more agents within an environment are constantly observing their environment. Based on these observations, an agent makes a request to a web server **(step 4)**, which forwards the request with the corresponding state information to a component called *Agent Ensemble Manager* **(step 5)**. The *Agent Ensemble Manager* loads pre-trained entity embedding vectors representing actions and states in an

[4] https://www.semantic-web-journal.net/
[5] https://arxiv.org/
[6] https://github.com/nmerkle/SW_Journal
[7] https://doi.org/10.6084/m9.figshare.22215079.v1

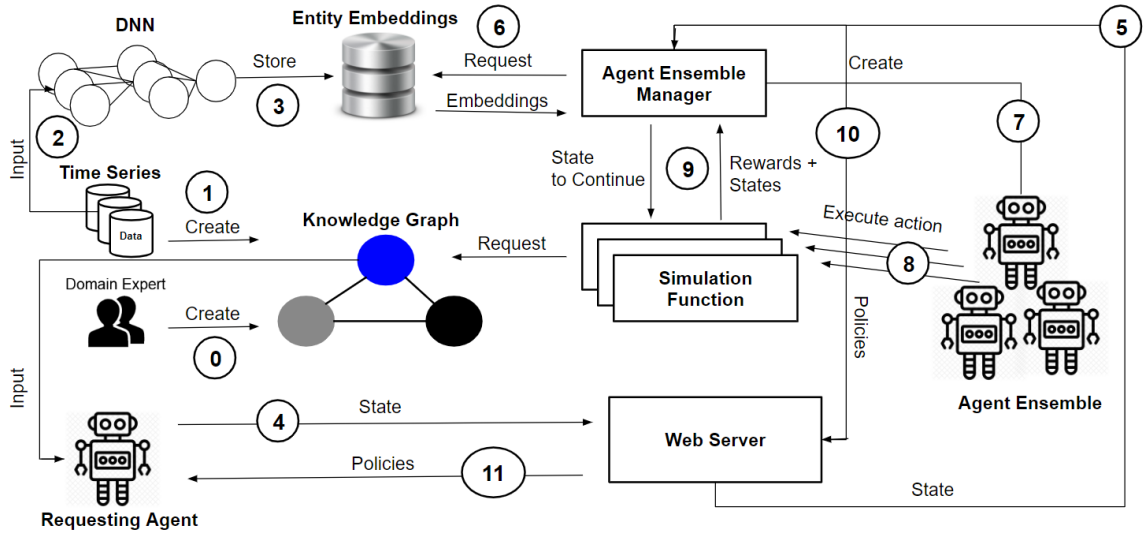


Figure 4.13.: Process flow and architecture of policies combination approach [152].

n-dimensional entity embedding vector space from a database **(step 6)**.

These entity embedding vectors are continuously trained for agent tasks[8], their associated states and atomic actions as soon as a new semantic task entity is either directly created by a domain expert **(step 0)** or automatically derived from provided datasets (e.g. log data) **(step 1)** reflecting performed task processes (e.g. during imitation or demonstration learning).

Section 4.5.2.2 outlines how such MDP activities are derived from corresponding datasets. For the simulation of arbitrary activities and contexts, the simulation component uses the activity knowledge graph, which is built and extended both by domain experts and from derived *Hidden Markov Models (HMM)* [205, 206].

The *Agent Ensemble Manager* determines the actions closest to the given state based on their given entity embedding vectors. A set of the nearest actions is selected for execution. Then, the *Agent Ensemble Manager* generates an ensemble of agents **(step 7)**. The ensemble size depends on the set of selected actions, since each agent of the ensemble executes exactly one action in each execution step **(step 8)**. The action commands in turn are sent by the respective ensemble agents in parallel threads to their simulation component, which updates for the corresponding ensemble agent the corresponding states depending on the executed actions. Therefore, the ensemble agents implement and invoke a simulation function that implements the simulation process. The simulation function sends the updated state representation along with the resulting reward for each action-state pair of each ensemble agent to the *Agent Ensemble Manager*, which then ranks and selects the actions that yield the highest reward values **(step 9)**. The semantic representation of activity, state and action concepts is defined by the underlying framework ontology, which represents

[8] The terms *Task* and *Activity* are considered synonymous in this work

MDPs and is the basis for the corresponding KG. It is important to note that the simulator knows all contexts in advance, as it can query all prevailing task, state and action entities. For this reason, the simulator can manage and update state representations and provide feedback through rewards, as all the necessary information is stored in the knowledge graph.

The found policies, i.e. action sequences, are managed by the *Agent Ensemble Manager* in a data structure, e.g. a list, until the executed activity is terminated by its goal and final state, which determines the end of the activity. Then, the policies are assembled as a sequence of actions in the *Agent Ensemble Manager* component, and sent back via the web server **(step 10)** to the agent that initiated the request and provided the corresponding state information **(step 11)**.

It is important to note that the web server provides two different modes for requesting policies. The first mode called *single-policy-mode*, allows the agent to request a rated single policy for the observed state at each execution step, while the second mode called *task-completion-mode*, allows the agent to request all policies at once that have been combined to complete the corresponding activity. Thus, the first mode requires for each new state a request, while the request of the second mode returns a complete solution, i.e. sequence of policies, for an activity.

In *single-policy-mode*, only one action is returned per agent request. However, even here, the 2nd, 3rd or n-best actions are also returned, ordered by the reward value they received when executed. Only single actions that resulted in a positive reward value are returned as alternative actions to the best action found in *single-policy-mode*.

In this way, agents in heterogeneous environments are able to request policies from the web server based on their observed states. This saves the agents time-consuming learning processes, as the required knowledge is already implicitly encoded in task or activity specific entity embedding vectors representing corresponding contexts.

In the used MDP model representation, semantic activity entities encapsulate states and actions that have an implicit semantic relationship, since an activity may consist of different states and require different actions that affect the observed states. These activity entities are accessible through the corresponding KG.

A simulation component, adhering to the framework's ontology, is thus able to simulate activities based on the provided activity entities and logged datasets that replicate the feedback and preferences of the target user. As outlined in Sec. 4.3, the main purpose of the simulation component is to update states based on executed actions and provide feedback on the utility of the executed action. This feedback is represented by reward values, where a reward value < 0 indicates undesirable or incorrect actions.

It is important to note that the simulation component might also use task descriptions extracted and generated from real operational data produced by demonstrations of human

agents. In these cases, the simulation component could incorporate different MDP contexts, allowing for a wider range of individual contexts that reflect different environments and activities of the agents.

4.5.2.2. MDP Model Generation by Activity Datasets

To enable the simulation of contexts and the prediction and evaluation of action effects in the framework, task or activity descriptions have to be either provided by domain experts or derived from datasets that emulate activities of humans or agents. Activities respectively their inherent processes can be described in terms of the probability distribution (i.e. *probability mass function (PMF) and probability density function (PDF)*) of states, actions performed, and observations made by agents within a state. The implicitly given probability distribution of entities (i.e. states, actions, observations) in datasets enables the derivation of Markov decision process models. In the following, it is explained how the agent framework derives MDPs from operational datasets in order to be able to use these MDPs for the simulation of action effects and state changes.

The prerequisite for this approach to work is that agents[9] collect and provide data during their runtime that they perceive from their environment. The basic assumption made in the framework is that agents collaborate by providing their knowledge or experience in the form of activity log data and meta data about the log data. The meta data is needed to describe the properties of the record so that the wrapper component of the framework can process the record and convert it into a task entity description. Therefore, it is proposed in this approach to represent the meta data as semantic annotation, i.e. linked data that can either be embedded as RDFa[10] in the web page of the referenced dataset or provided via a SPARQL endpoint that maintains the linked meta data. The advantage is that the meta data can be automatically retrieved and interpreted by the framework to understand different log datasets and process them into task entity descriptions.

It is important to mention that the meta data provided should not be sensitive data that can be misused by third party access. The protection of personal data should be ensured by additional privacy and security measures to prevent unauthorised data access. Therefore, an access restriction should be set up where personal or private data is to be used or provided as meta data. However, it is out of the scope of this work to address security and privacy issues.

The ontology concepts and properties of the proposed meta data are depicted in Table 4.17.

To show the instantiation of the proposed meta data concepts for log records, an example in Turtle format is shown in Listing 4.11. The dataset that serves as the basis for the example listing was adopted from Kaggle[11]. It contains 4736 samples for 64 countries on confirmed

[9] Agents can also be human beings or other living beings that perform activities and display behaviour.
[10] see for more information: `https://www.w3.org/TR/rdfa-primer/` and `https://rdfa.info/`
[11] `https://www.kaggle.com/davidoj/covid19-national-responses-dataset?select=countermeasures_db_johnshopkins_2020_03_30.csv`

Roles/Properties	Domain	Mandatory	Cardinality
Concept: Dataset			
:hasSample	:Sample	Yes	≥1
:hasReferenceURL	xsd:string	Yes	1
Concept: Sample			
:hasFeature	:Feature	Yes	≥1
:hasAction	:Action	Yes	≥1
:hasSampleSize	xsd:integer	Yes	1
:hasStartDate	xsd:date	No	1
:hasEndDate	xsd:date	No	1
Concept: Feature			
:hasType	{:Continuous, :Ordinal, :Nominal}	Yes	1
:hasUnit	xsd:string	No	1
:hasRangeStart	xsd:double	No	1
:hasRangeEnd	xsd:double	No	1

Table 4.17.: Ontology concepts and properties of the meta data describing log records.

COVID-19 cases and deaths, and the interventions implemented by governments. In total, the dataset consists of 28 columns. Of these, the confirmed COVID-19 cases and deaths are considered as features, while the remaining 24 columns describe the actions taken. The data samples are delimited by the feature *country*. Listing 4.11 shows turtle statements that describe the samples for the country Germany. Analogously, this is done for every country in the dataset. Data samples can also be divided according to features other than the country. In this example, however, it made sense to classify and select the data based on the corresponding country.

Listing 4.11: Meta data example for log records.

```
prefix p: <http://example.org/Property#>
prefix c: <http://example.org/Concept#>
prefix e: <http://example.org/Entity#>
prefix xsd: <http://www.w3.org/2001/XMLSchema/>
e:COVID-19_Country_Measures a c:Dataset;
  p:hasSample e:Germany;
  p:hasReferenceURL "http://example.org/data/covid-19.csv"^^xsd:string.
e:Germany a c:Sample;
  p:hasSampleSize "102"^^xsd:integer;
  p:hasStartDate "2020-01-23"^^xsd:date;
  p:hasEndDate "2020-05-04"^^xsd:date;
  p:hasFeature e:Confirmed_Cases_Germany,
       e:Deaths_Germany.
e:Confirmed_Cases_Germany a c:Feature;
  p:hasType e:Continuous;
```

```
16     p:hasUnit "people"^^xsd:string;
17     p:hasRangeStart "0"^^xsd:integer;
18     p:hasRangeEnd "83149300"^^xsd:integer.
19  e:Deaths_Germany a c:Feature;
20     p:hasType e:Continuous;
21     p:hasUnit "people"^^xsd:string;
22     p:hasRangeStart "0"^^xsd:double;
23     p:hasRangeEnd "83149300"^^xsd:integer.
24  e:Germany p:hasAction e:Symptomatic_isolation_-_targeted,
25    e:Symptomatic_isolation_-_blanket,
26    e:Asymptomatic_isolation_-_targeted,
27    e:Asymptomatic_isolation_-_blanket,
28    e:Domestic_travel_restriction,
29    e:Nonessential_business_suspension,
30    e:International_travel_restriction,
31    e:Testing,
32    e:Contact_tracing,
33    e:Mask_wearing,
34    e:Hand_washing,
35    e:Gatherings_banned,
36    e:Healthcare_specialisation,
37    e:Public_education_and_incentives,
38    e:Assisting_people_to_stay_home,
39    e:Public_cleaning,
40    e:Miscellaneous_hygiene_measures,
41    e:Public_interaction_and_hygiene,
42    e:School_closure,
43    e:Activity_cancellation,
44    e:Resumption,
45    e:Diagnostic_criteria_loosened,
46    e:Diagnostic_criteria_tightened,
47    e:Testing_criteria.
```

The example Listing 4.11 describes that the dataset named *COVID-19_Country_Measures* consists of samples belonging to the country *Germany* **(line 5 - 6)**. The samples contain two observable features named *Confirmed_Cases_Germany* and *Deaths_Germany* **(line 12 - 13)**. Both features are continuous features that have as their unit the number of people who are either confirmed cases or have died **(line 15 - 16, 20 - 21)**. The range or carrying capacity of the features depends on the population size of Germany at the time the samples were observed and created **(line 18, 23)**. The remainder of the samples from Germany relate to various government interventions that have been implemented to contain cases and deaths **(line 24 - 47)**. In addition, the number of samples and the date range in which the samples were created are given **(line 9 - 11)**. With this meta-information, the wrapper component of the framework can now derive MDPs and task entity descriptions, which will be discussed later in this chapter. Agents delivering log records to the agent framework are required to provide a meta-description of their record that conforms to the concepts

and properties of the proposed linked meta data representation provided in Table 4.17.

It is assumed that only data is provided by agents that reflect the successful behaviour of the agent in the corresponding environment. Success or failure is determined by reward values that are either directly assigned by the environment or implicitly derived by the agent.

Deviating from the proposed meta data representation, each row of the dataset has to consist of the following attributes: $S_n, A_n, O_{n,m}, R_n, S_{n+1}$, where S_n is an observed state at time n, A_n is an executed action in state S_n, $O_{n,m}$ is an arbitrary number m of observed features that constitute state S_n at time n, R_n is the reward or feedback value received for the executed action A_n in state S_n and S_{n+1} is the subsequent state observed with the observation features $O_{n+1,1\ldots n+1,m}$.

It is important to note that datasets usually do not have state labels that summarise and describe the current values of the features. For this reason, state labels are artificial and have to be derived and assigned through various heuristics. This is also the reason why the meta data specification for datasets does not include a concept for *State*. To circumvent this problem, possible approaches are now considered that allow states to be derived based on the corresponding dataset.

For instance, one problem that might occur in recorded data is when only continuous observation values representing a state are provided, but not the associated state label itself. In such situations, the dataset needs to be pre-processed and the ranges of values of the continuous observation features split into states by binning. In this way, different states covering different value ranges of the feature can be defined for each individual observation feature. The binning approach could be guided by the actions performed at each time step. Based on the collected feature values observed before an action is performed, the lower and upper limits of these feature values can serve as range limits for a state. This in turn means that the actions prescribe the derived states.

Another approach for deriving states for continuous feature values would be to divide the mathematical progression curves of the data into sections at their inflection points, which then represent the corresponding state changes and thus new states. An inflection point is given when the growth factor of the curve becomes 1. Mathematically, a non-linear curve changes direction at the corresponding inflection points or curvatures[12] and can thus be divided into the following 3 sections or states: *Ascending*[13], *Descending*[14], and *Constant*[15]. One approach for determining the curvature of function graphs approximating the measured data is to consider, for each observed data value, the two neighbouring data values measured at the previous and subsequent time step, see Fig. 4.14 and Eq. 4.9. This means

[12] In general, curvatures of a function graph are calculated from the second derivative of the function. Individual data points may be used to approximate the trend of a function.
[13] i.e. concave upwards
[14] i.e. concave downwards
[15] i.e. inconclusive

that the *average difference of differences* between the currently observed data point $f(x)$ and the neighbouring data points $f(x+h)$ and $f(x-h)$ can be calculated to determine whether the function at the corresponding data point $f(x)$ is ascending, i.e. $\frac{df}{dx} > 0$, or descending, i.e. $\frac{df}{dx} < 0$, or inconclusive, i.e. $\frac{df}{dx} = 0$. Equivalently, the second derivative, i.e. $\frac{d^2f}{dx^2}$, (see Eq. 4.10) [251, 238] can be calculated for each acquired data point $f(x)$ in relation to the neighbouring data points $f(x+h)$ and $f(x-h)$. As a side effect, the second derivative also allows to recognise whether an extremum of a curve is a local maximum or minimum [251].

The advantage of this categorisation is that non-linear curves can be simplified by sampling, so that state changes and linear correlations[16] between different features can be detected and conclusions about the mutual effects of actions on features and state changes can be investigated, (see Fig. 4.15).

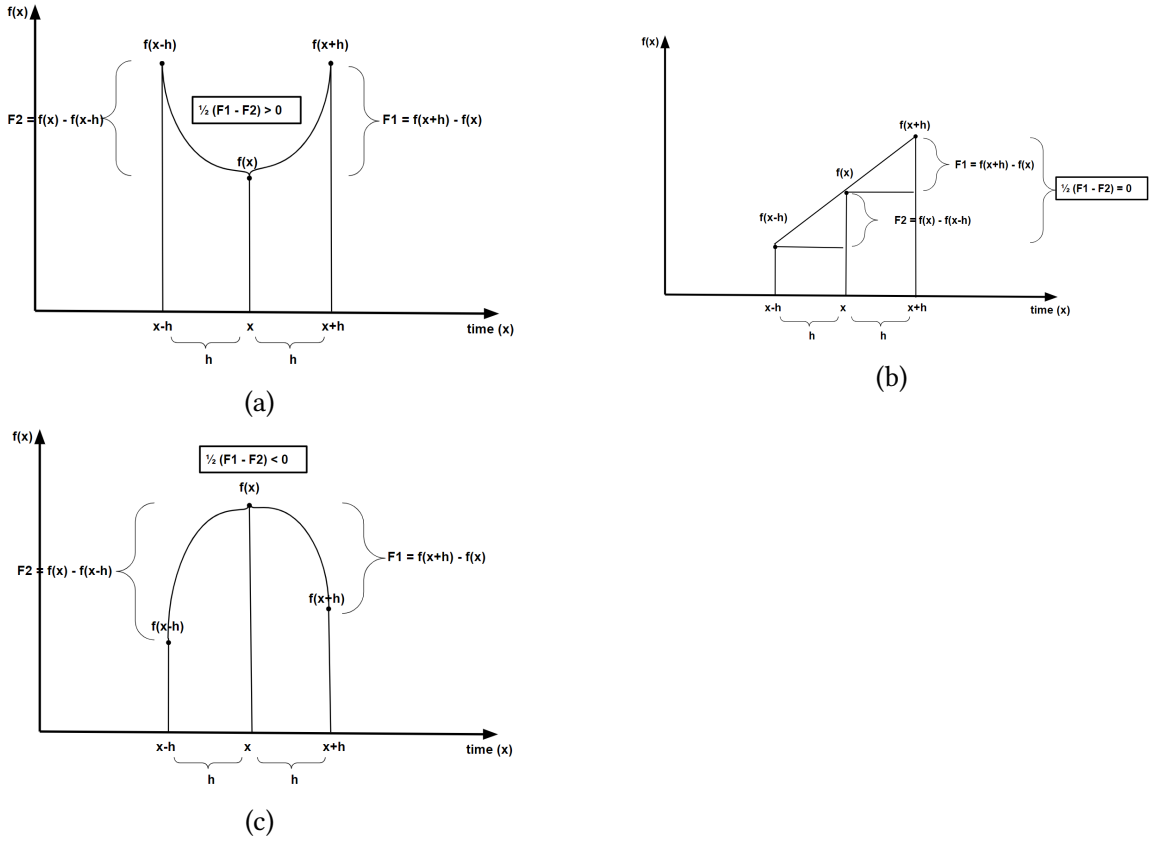


Figure 4.14.: The second derivative at single points (here x) indicates whether the underlying function is increasing, decreasing or constant. E.g.: (a) Shows an increasing, i.e. *concave upward*, slope of a function with a *local minimum*. (b) Illustrates that the slope of the function is constant, i.e. *inconclusive*. This may indicate that an *inflection point* is reached. (c) Illustrates a decreasing, i.e. *concave downward*, slope of a function with a *local maximum*.

[16] see Pearson's correlation

$$\frac{\mathrm{df(x)}}{\mathrm{dx}} = \frac{1}{2}\left(\Big(f(x+h) - f(x)\Big) - \Big(f(x) - f(x-h)\Big)\right) \tag{4.9}$$

where $\frac{\mathrm{df(x)}}{\mathrm{dx}}$ is the derivative w.r.t. data point x and h is the distance on the x-axis between point $f(x)$ and its neighbouring points $f(x+h)$ and $f(x-h)$. The equation describes the *average difference of the differences*.

$$\frac{d^2f}{dx^2}(x) \lim_{h \to 0} = \frac{f(x+h) - 2f(x) + f(x-h)}{h^2} \tag{4.10}$$

where $\frac{d^2f}{dx^2}(x)$ is the derivative w.r.t. x, h is the distance from data point x on the x-axis to each of its two neighbouring points. h goes against zero.

The agent framework could thus detect effects from data, via calculation of the (mean) *growth/decay rate* (see Eq. 4.11) and *growth/decay factor* (see Eq.4.12), and update states or features depending on the given state and action performed. In Eq. 4.11, $F_{current}$ is the current feature value, while F_{start} is the last feature value considered. n is the number of samples that lie between the current and the last feature value considered. The basic assumptions here are that a) the feature values are measured at frequent time intervals and b) that the features of the dataset are correlated and directly influenced by the actions performed by the agent.

$$\mathrm{GR_{mean}} = \left(\frac{F_{current}}{F_{start}}\right)^{\left(\frac{1}{n}\right)} - 1 \tag{4.11}$$

$$\mathrm{GF} = 1 + \mathrm{GR} \tag{4.12}$$

Based on the current feature value and the growth factor for a certain period of time, the next feature value can be calculated with Eq. 4.13 if an *exponential growth* is assumed. The exponential growth/decay function can also be expressed as in Eq. 4.14 where $N(t)$ is the observation feature at time t, $N(0)$ is the observation feature at time zero, e is the Euler number, α is the growth/decay rate and t is the time steps to be considered [237]. However, depending on the problem, i.e. task, a *logistic curve* is usually more suitable, since the growth of a considered feature can reach a maximum capacity which stops the growth. For completeness, the corresponding *logistic function* for computing the *rate of change* is shown in Eq. 4.15, where k is a growth constant, N_t is the current feature value, N_{total} is the carrying capacity that feature N_t cannot exceed, and t is the time interval for which the updated feature value is computed.

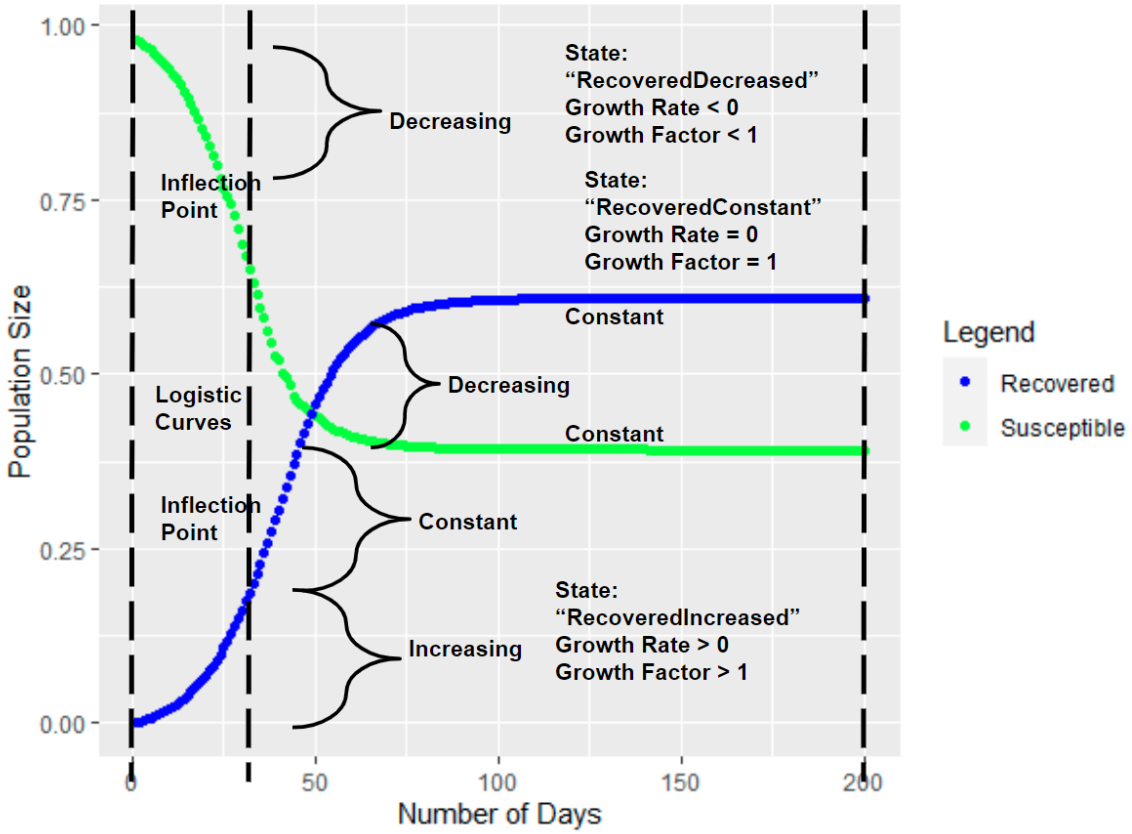


Figure 4.15.: The sampling of states by inflection points. The dashed lines mark and delimit the samples in the time series data that represent different states of a given feature (y-axis) among time steps (x-axis). A total of 4 states (e.g *SusceptibleDecreased, SusceptibleConstant, RecoveredIncreased, RecoveredConstant*) are sampled in the example, i.e. 2 for each of the two observation features.

$$F_{next} = F_{current} * GF^{t} \tag{4.13}$$

$$N(t) = N(0) \cdot e^{\alpha \cdot t} \tag{4.14}$$

$$\frac{dN}{dt} = k\left(1 - \frac{N_t}{N_{total}}\right)N_t \tag{4.15}$$

Based on such a dataset and the *Bayes theorem*, transitions, i.e. conditional probabilities, between states and observation features and actions performed can be determined. The goal is to derive a *Hidden Markov Model (HMM)* (see [236]) that represents the stochastic dynamics of the underlying activity or task. Using Markov chains, the simulation component is able to determine probable state transitions based on observations made and an action performed to simulate the corresponding activity dynamics. Thus, depending on a given state S_n and an action performed, the next most likely state can be inferred. Fig. 4.16 shows which transition probabilities within the HMM are determined and considered by the simulation framework. Starting from a state S_n, probable transitions to subsequent states are determined by the conditional probability $P(S_{n+1} \mid S_n)$. The conditional probabilities for observations and actions are determined accordingly with $P(O_n \mid S_n)$ for observations and $P(A_n \mid S_n)$ for actions. Based on such a HMM, the expected maximum probability of transitions can be either estimated by the *Viterbi* [75] or *Baum-Welch* [205] algorithm, which allow the simulation component to simulate the most probable sequences of actions and states for any given state observation within the HMM.

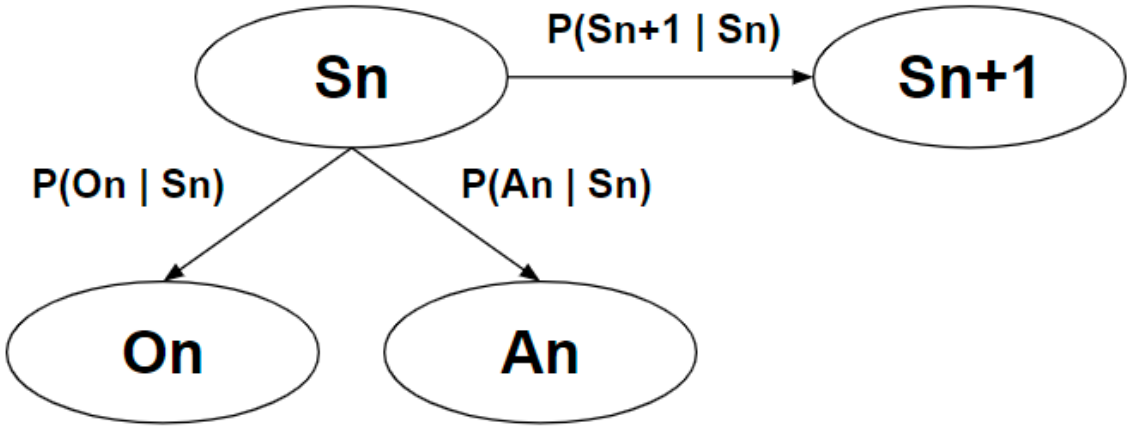


Figure 4.16.: A (hidden) Markov model that represents hidden states, state transitions and state observations [152].

For determining the conditional probability of observation features, a distinction has to be made between different types of features and data distributions. For example, in the case of

categorical and binary features, the (relative) frequency of occurrence is counted, whereas in the case of continuous features, e.g. the *Gaussian function* (see [275]) is applied. In these two ways, the conditional probability of each feature in each state can be determined. However, pre-condition is that the data is normally distributed, i.e. Gaussian. Though, if no prior assumptions can be made about the data distribution and probability density function, then *kernel density estimation*, also named *Parzen-Rosenblatt Window* [211, 192], can be applied with different kernel functions (e.g. Gaussian, tophat, exponential, Epanechnikov, linear, cosine) and a pre-defined *bandwidth* parameter h that smooths the estimated density function curve. For assumed Gaussian distributions the optimal bandwidth parameter h can be estimated by Silverman's equation [231], see Eq. 4.16.

$$\mathsf{h} = \left(\frac{4}{3}\right)^{\frac{1}{5}} \sigma \mathsf{n}^{-\frac{1}{5}} = 1.06 \sigma \mathsf{n}^{-\frac{1}{5}}, \tag{4.16}$$

where σ is the standard deviation and n is the sample size.

Equation 4.17 shows the corresponding *kernel density estimator* [211, 192] function of the probability estimate,

$$\mathsf{pK}(\mathsf{y}) = \frac{1}{hN} \sum_{i=1}^{N} \left(\frac{y - x_i}{h}\right) \tag{4.17}$$

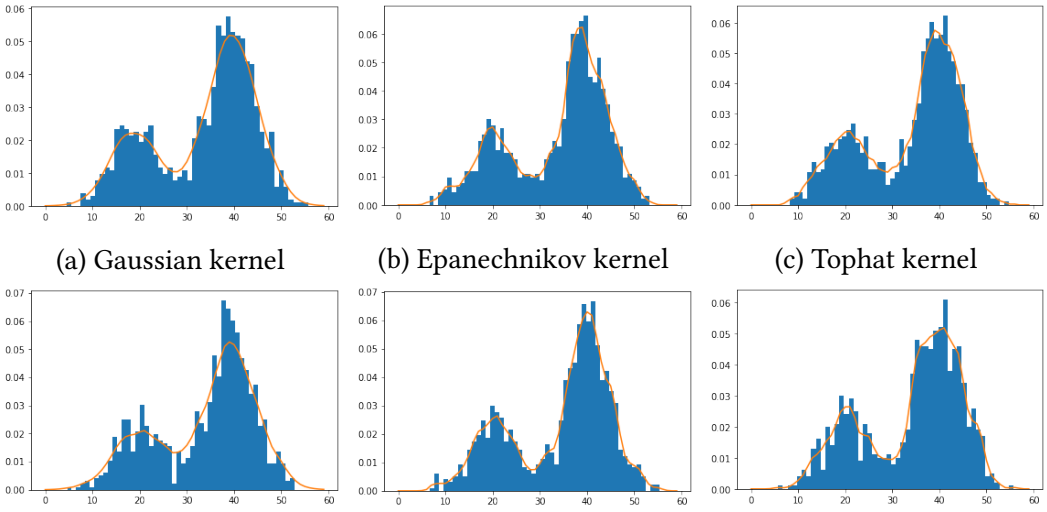
where K is the kernel function, h is the bandwidth parameter for smoothing the curve, y is a considered data point and $x_i \ldots x_n$ is a group of surrounding data points from the dataset for which the distribution is to be estimated[17]

Then, by generating samples from the present sample space the mean squared error between the original data and sampled data can be computed for each kernel and density estimator in order to select the kernel with the lowest error rate.

Fig. 4.17 illustrates an example of a bimodal data distribution (see blue distribution) and the estimated density curves (see red curves) of the applied kernel functions. It can be seen that the kernels adapt differently to the given data distribution depending on their estimation and relatedness to the data distribution.

For personalisation purposes user feedback is either directly requested from the user or defined and annotated by domain experts. The obtained feedback is stored in the collected datasets as reward values. Since the reward values are supposed to reflect the value or utility of all the state-action pairs present, the proposed approach calculates the *mode*

[17] In this way, each data point is used to estimate the underlying data distribution. Based on the estimated data distribution, the probability of the observed (continuous) features in the collected datasets can be approximated to obtain transition probabilities of the Markov model.



(a) Gaussian kernel (b) Epanechnikov kernel (c) Tophat kernel

(d) Exponential kernel (e) Cosine kernel (f) Linear kernel

Figure 4.17.: Bimodal data distribution (in blue) and different estimated density curves (in red) by different kernel density estimators and with bandwidth 2.

values of the rewards associated with each state-action-state transition in the dataset. The proposed agent framework's simulation component uses this information, i.e. mode value of rewards, in order to determine the expected reward for a state and provide feedback to the agent that trains policies. Thus, it is assumed that an ordered, finite and unique set of reward values is collected and provided in the dataset that can be obtained in each state.

The derived and created HMM is transferred into a semantic task entity, see Sec. 4.2, that serves as input for the *context-dependent policy combination* [152] approach, so that entity embedding vectors can be trained, as explained in Sec. 4.5.2.3. In addition, the simulation component uses the task entity of the HMM to provide feedback to the ensemble of agents for their performed actions, see Sec. 4.5.2.4.

4.5.2.3. Learning of Entity Embeddings for MDP Entities

The DNN that is presented in this section is an extension of the DNN that was presented in Sec. 4.5.1 and Fig. 4.12. Instead of only considering tasks and corresponding actions, also task-state, task-action, and state-action co-occurrences are taken into account in the extended DNN architecture.

The prerequisite for this approach to work is that datasets that reproduce sequences of tasks and activities, e.g. through imitation learning or real-time data from agents, are used as the basis for training entity embeddings. A corresponding dataset would consist of samples $D_i := \{T_m, S_{n+1}, A_n, R_n\}$, where D_i is a data sample, T_m is an executed task or activity, A_n is a performed action, S_{n+1} is a state subsequent to the performed action A_n, and optionally R_n is a reward value indicating the success of the performed action. The reward value might be necessary for data sampling because only data samples that yield a

positive reward are of interest for learning promising, i.e. context-appropriate, policies.

From the data samples the following features are extracted: (*activity id, action id, subsequent state id*) in order to serve as input to the deep neural network (DNN) that trains the entity embeddings. The mentioned features represent binary features and can therefore either have the value 0 for *no co-occurrence* and 1 for *co-occurrence*. For example, a data sample consisting of (activity_i = 1, state_{n+1} = 1, action_n = 0) indicates that the referenced activity and state co-occur but the action does not. A co-occurrence is determined by the fact that activity entities reference (i.e. link) the corresponding state or action entities, while a co-occurrence between state and action entities exists when an action has directly led to the corresponding subsequent state. Thus, the data samples are generated by two different sources of information, namely the activities and their relationships to states and actions, and observations indicating which actions were performed and which state changes were caused by them. To obtain representative entity embeddings and avoid imbalanced datasets, it is necessary to provide as many positive as negative examples of co-occurring entities.

The concerned entities are encoded by index numbers since a DNN can only process numerical inputs. Therefore, a dictionary for each entity concept has to be maintained that maps the entity names to the corresponding index numbers. In each training iteration, batches of training samples are assembled in order to serve as input for the DNN. The DNN that trains the entity embeddings for each concept, i.e. activity, state and action, is depicted in Fig. 4.18 and consists of the following network layers: input, embedding, dot product that joins the embedding layer outcomes, a layer for reshaping the dot product outcomes to a shape of one and a fully connected dense layer as output layer that projects by the Sigmoid activation function, the reshaped vector value to a binary output of 0 or 1.

The embedding layer can consist of an arbitrary number of units, but experiments have shown that 50 units are sufficient to achieve convincing results. The optimisation of the embedding layer was performed by the adopted *Adam optimiser* [120] algorithm. However, other optimisation algorithms (e.g. stochastic gradient descent) are also possible. It is important to note that for each entity of activities, states and actions, an embedding vector (see example embedding below) is trained that represents the corresponding entity in the embedding vector space. Thus, the main goal is to obtain representative embedding vectors that indicate the semantic relatedness of entities through their observed context. Fig 4.19 visualises entity embeddings in *Tensorflow's Embedding Projector*[18] web application.

$$\vec{(}embedding_n) = \begin{pmatrix} 0.344394855487582 \\ -0.35454000765544 \\ 0.339430778676755 \\ \dots \\ n \end{pmatrix}$$

[18] https://projector.tensorflow.org/

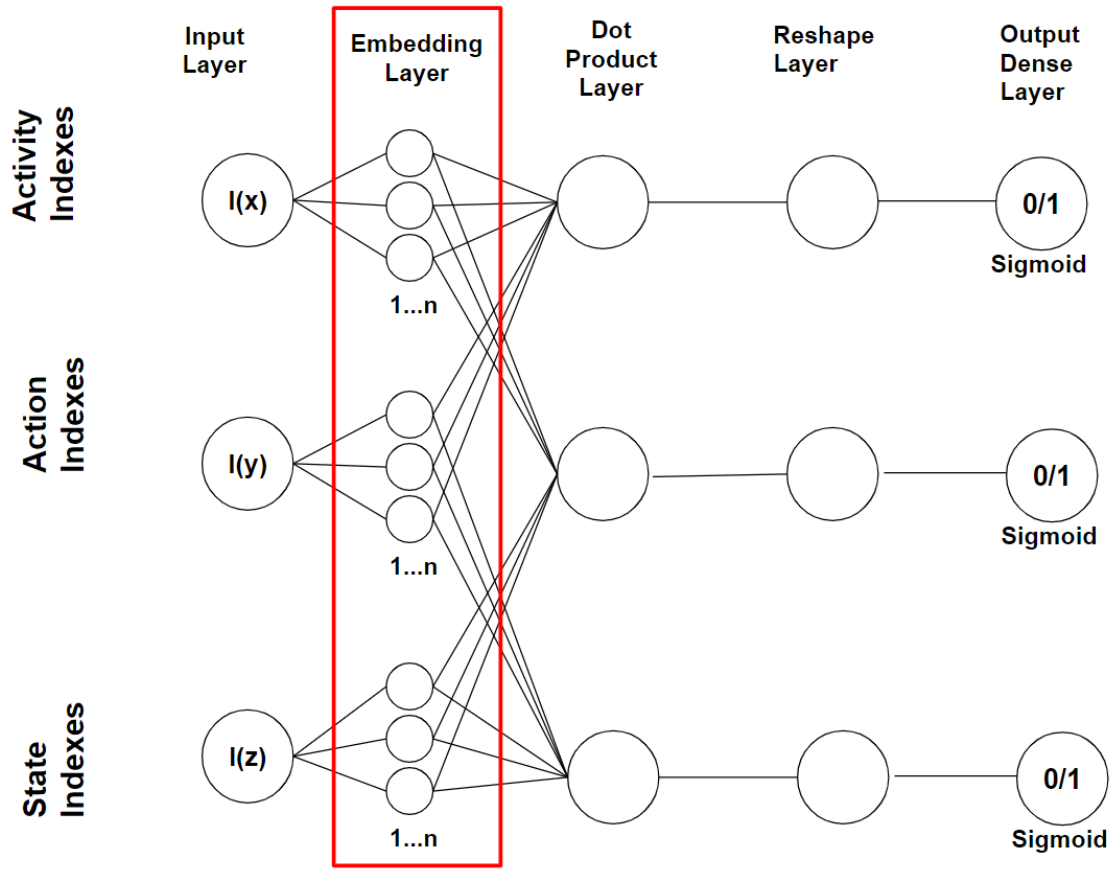


Figure 4.18.: Arrangement of the deep neural network for training entity embedding vectors that represent activity contexts [152].

Figure 4.19.: The trained embedding vectors of activities, states and actions are merged and visualised in a 3-dimensional space in the Tensorflow Projector. The colouring of the embedding vectors (shown as dots) indicates how spatially far away other embedding vectors are from a selected embedding vector (here, e.g. in light red Wash_drinking_glass_1_Done). The darker the hue of a neighbouring embedding vector, the closer it is to the selected vector. The distances in this illustration were measured with the Euclidean distance [152].

As loss function that measures the error rate during training, *cross-entropy* is recommended since it is intended for binary classification problems as required in the given DNN.

It is important to note that as soon as new tasks are deployed into the environment and agent framework, *Transfer Learning* [253, 73], i.e. a retraining of the DNN, is performed in order to incorporate the new task into the trained embedding model.

4.5.2.4. Policies Combination Algorithm

Once the entity embedding vectors are trained and the corresponding task entities are deployed into the agent framework, policies can be combined by ensembles of agents based on the sent state information of a requesting agent. It is worth noting that the agent does not need to send task-related information in its request, since it does not know in advance what activity to perform and it is determined by the context what behaviour is expected from the agent. Thus, the activity, and corresponding sequence of actions, is derived only from the observed state information. The algorithm for composing state-based policies is shown in Algo. 9.

The input parameters required for the algorithm are the given state representation sent by the requesting agent and the trained entity embedding vectors of each state and action entity referenced in the KG maintained by the agent framework **(line 1)**. The returned output of the algorithm is an array of policies. The algorithm tracks and stores the given state, which is the starting point for each ensemble agent **(line 3)**. To store the policies found, a data structure, i.e. a list of arrays (see Table 4.18), is declared **(line 4)**. Then a *while-loop* is started, which stops as soon as a goal state, that terminates the current activity, is reached **(line 5)**. The *while-loop* starts with the selection of the embedding vector representing the sent state **(line 6.)**. The embedding vector of the state is needed to find the nearest embedding vectors of the nearest actions **(line 7)**. Therefore, a similarity metric (e.g. cosine distance, Euclidean distance) is used to calculate the distance between embedding vectors in an n-dimensional embedding vector space. An array stores, for each agent in the ensemble, its feedback received from the simulation component (i.e. the updated state and the reward received) **(line 9)**.

For each action that is within the specified search radius, a new agent is created and initialised with the corresponding state information and the action to be performed **(line 8 - 10)**. Then, each agent calls the function *simulate* which executes the respective action of the agent **(line 11)**. The function *simulate()* then returns the updated state with the corresponding reward value that is sent back to the *Agent Ensemble Manager*. The updated state object is stored in the results array **(line 12)**. This is done within a *for-loop* for each agent of the ensemble **(line 9-12)**.

The advantage of using agent ensembles is that actions can be executed in parallel threads and each thread with its own simulation function updates and manages its own state. The ensemble agents are then synchronised in the main thread by selecting the best, i.e. reward maximising, state to continue the MDP and initialise the next generation of ensemble

Algorithm 9: PoliciesCombination

```
Input: initialState, stateEmbeddings, actionEmbeddings, maxDistance
Output: policies
Function PoliciesCombination(initialState, stateEmbeddings, actionEmbeddings,
  maxDistance):
    radius = maxDistance
    currentState = initialState
    policies = []
    while !currentState.isGoal do
        stateVector = stateEmbeddings[currentState]
        closestActions = findClosestActions(stateVector, actionEmbeddings, radius)
        results = []
        forall action IN closestActions do
            agent = new Agent()
            updatedState = simulate(agent.exec(currentState, action))
            results.push(updatedState)

        bestResult = selectBestResultByReward(results)
        if bestResult.reward <= currentState.reward then
            radius += 0.25
        else
            policies.push(result.state, result.action)
            currentState = bestResult
            radius = maxDistance
            policies.push(bestResult.state, bestResult.action)

    return policies
```

Rank (m)	**Sequence (n)**			
1	$\text{Action}_{1,1}$	$\text{Action}_{1,2}$	$\cdots$	$\text{Action}_{1,n}$
2	$\text{Action}_{2,1}$	$\text{Action}_{2,2}$	$\cdots$	$\text{Action}_{2,n}$
$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$
m	$\text{Action}_{m,1}$	$\text{Action}_{m,2}$	$\cdots$	$\text{Action}_{m,n}$

Table 4.18.: Data Structure with the combined actions resp. policies. The rank determines the quality of the actions based on the reward values obtained [152].

agents.

After all ensemble agents have performed their assigned actions, a function called *selectBestResultByReward()* is called that sorts all obtained states in descending order by their reward values **(line 14)**. The action that achieved the state with the highest reward value compared to the last obtained reward value is then selected as the best action and the *policies* array stores this action together with the corresponding previous state and updates the variable *currentState* to the new state that provided the best result **(line 17-19)**. In Algo. 9 only the best action sequence is selected and no alternative action sequences. This means that a *rank 1*-combination of policies is performed. In case the agent should also be provided with alternative action sequences, the algorithm can be adapted to store the second, third, n-best action sequence in the policies data structure.

In **(line 20)**, the variable *radius* is reset to the originally set value of *maxDistance*, since only the closest actions should be considered again for the next policy search, which saves computing time and resources, since the number of actions to be performed by agents increases or decreases proportionally to the search radius of the state. However, if the current rewards received are less than or equal to the last best reward, then the variable *radius* is increased by 0.25 each time the algorithm gets stuck and does not return reward-maximising actions **(line 15-16)**. The parameter named *maxDistance* is thus a hyper-parameter and initially defines the radius boundary within which the nearest action embedding vectors have to be located **(line 2)**. The function exits and returns the array of policies once the target state of the current activity is reached **(line 22)**.

One problem that can occur with this approach is that when the agent executes the proposed policies, the state changes in the environment may not match the state changes of the received policies. In these particular cases, where the states of the received policies are inconsistent with the actual evolving states, the agent has to make a new request to the platform with the currently detected state so that a new policy or set of policies can be determined by the policies combination platform. This in turn means that the requesting agent has to compare the actual detected state and the state of the provided policy in each execution step.

In addition, the platform offers in its response to the *single-policy-mode* query, several alternative actions ranked according to the reward value they have caused and received. Thus, if the action determined to be the best leads to an inconsistent state, the action ranked as the second best, for example, can simply be executed by the requesting agent and checked whether an intended, i.e. policy-compliant state, could be achieved.

The presented approach aims at enabling agents to act across contexts in a predictive manner and thus arrive at decisions more quickly. Moreover, it addresses policy requirements of agents and allows them to perform activities without having to train policies beforehand [152].

4.6. Conceptualisation of Agents

In the previous sections, concepts and approaches of the agent framework were introduced. However, agents that wish to communicate with the agent framework and use its services, have to comply with certain conditions and prerequisites also in order to provide the framework with data and policies that are to be of use to other cooperating agents. This means that participating agents require to adhere to procedures and interfaces that enable a smooth interaction between agents and the framework. Thus, developers of worker, consumer or hybrid agents have to implement these proposed procedures and interfaces in order to communicate and comply with the agent framework.

In the next section, the general structure of agent programs will be discussed, then both the worker agent and the consumer agent and their functionality (e.g. training policies, providing provenance data) and purpose will be outlined.

The approach that is presented here, is based on previous work [155] and was already published at the K-Cap[19] 2019 conference.

4.6.1. General Structure and Flow of the Agent Program

As already described in Sec. 4.1, the agent framework distinguishes between three types of agents, namely the *worker agent*, the *consumer agent* and the *hybrid agent*. Due to their different roles, purposes and functionality, the first two types of agents have to comply with different prerequisites, while the third type is an amalgamation of the two and therefore does not need to be discussed further. However, all types of agents have some processes in common and therefore need to implement certain basic functions or interfaces that correspond to the workflow steps defined by the agent framework. Independently of this, all agents can implement any algorithms (e.g. Naive Bayes classifier, genetic algorithms, neural networks, etc.) that are required for decision-making and knowledge generation (see Sec.2.3 and 2.4).

Figure 4.20 illustrates the general agent program that an agent of the framework has to implement. The first thing a deployed agent has to do, is to request its agent profile and the appropriate task entity description that is provided via a SPARQL endpoint, since agent profiles as well as task descriptions are maintained in a knowledge base or knowledge graph.

Based on the referenced topics within the task entity, the agent subscribes for the corresponding topics to an Internet of Things (IoT) or Web of Things (WoT) (see Sec. 2.2.7) platform that handles the event-driven, bi-directional communication between agent and environment.

[19] `https://www.k-cap.org/2019/`

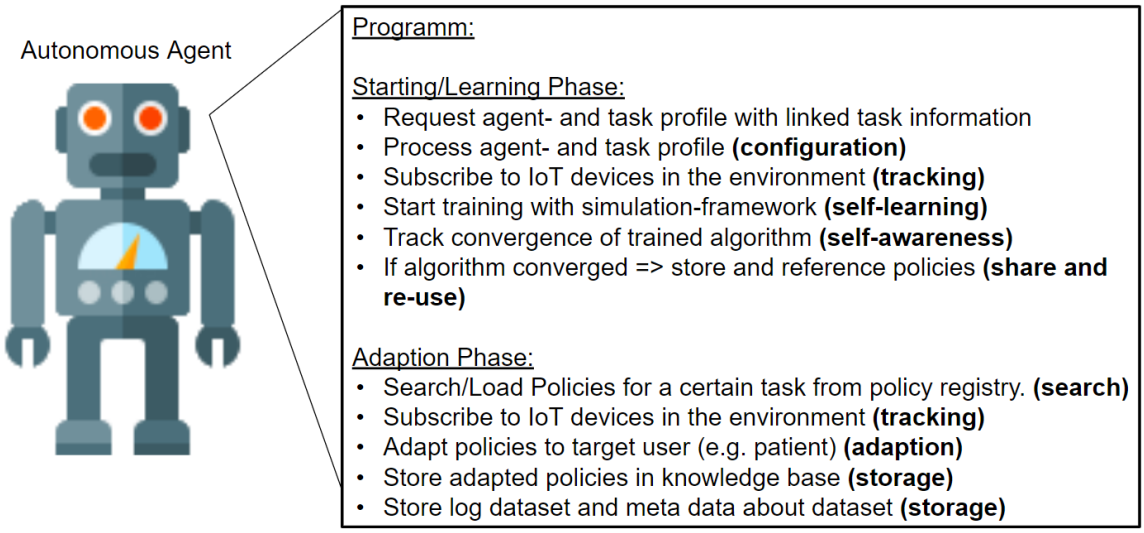


Figure 4.20.: The general programme to which an agent of the framework is obliged to adhere.

Depending on whether default policies already exist or not, the agent either executes the policies or first trains policies by interacting with the corresponding simulation component. In case the worker agent trains first the task policies, it will track the convergence of its value, utility, loss or fitness function. At convergence, the agent assumes that its policies are sufficiently trained and stores them together with the corresponding provenance data in the knowledge base.

Listing 4.12 and Listing 4.13 are taken from the published K-Cap paper [155] and show an example of a policy represented as both a SPARQL CONSTRUCT query and a Notation3 rule. The sample policy belongs to an evaluated web task (see Sec.5.1.2), in which the corresponding task asks the agent to check a check-box with a synonymous identifier. Both formal representations therefore express the same policy and can be used depending on which language, i.e. SPARQL or Notation3, an agent would like to support and implement. One condition, however, is that the serialisation supported is a standardised one that corresponds to the Semantic Web, since agents are supposed to be able to exchange knowledge through shared concepts and languages.

The two representations of the policy express that the action labelled *ClickCheckBoxBySynonym* should be executed if there is a check box in the given state representation but it is not yet ticked and the label next to the check box does not appear in the instruction text of the web page.

Agents who train policies have the requirement to create formal representations of their policies and make them available to other agents via the framework. The two examples for SPARQL and Notation3 show how this can be done. In the SPARQL query, the CONSTRUCT expression with the corresponding statement defines what action is to be performed, while

the WHERE clause lists the triples and conditions that specify a corresponding state representation.

Listing 4.12: SPARQL CONSTRUCT query example for defining a policy [155].

```
CONSTRUCT {?state :hasAction :ClickCheckboxBySynonym.}
WHERE {
?state rdf:type :State.
:IsCheckboxExisting :hasValue ?exists.
:LabelsIsInInstructionText :hasValue ?intxt.
:IsCheckboxClicked :hasValue ?checkboxClicked.
FILTER(?exists = 1 && ?intxt = 0 && ?checkboxClicked = 0)}
```

The Notation3 rule first specifies an existential quantifier (i.e. @forSome), followed by observable features, i.e. *IsCheckBoxExisting*, *CheckboxClickedBySynonym*, whose values determine the corresponding condition. If the listed conditions, i.e. triples, are true, the consequence is that the action, i.e. *ClickCheckboxBySynonym*, is derived.

Listing 4.13: Policy example defined as N3 rule [155].

```
@forSome :IsCheckboxExisting, :CheckboxClickedBySynonym,
:LabelsIsInInstructionText.
{:IsCheckboxExisting :hasValue "1".
:CheckboxClickedBySynonym :hasValue "0".
:LabelsInInstructionText :hasValue "0".}
=> {[:hasAction :ClickCheckboxBySynonym.]}
```

If the agent is a *consumer*, it requests policies from the framework that match to its assigned task and adapts them based on feedback that is either provided by a real environment or by the simulation component that utilises the corresponding task entity representation. To adapt the default policies, the agent has to be able to incorporate feedback, i.e. reward signals, into its model representation. It turns out that agents that implement reinforcement learning are predestined for policy adaptation. However, other algorithms, such as genetic algorithms, are also possible, since they can adapt their model on the basis of a fitness function and thus incorporate feedback from the environment.

Once the agent recognises the convergence of its positively evolving reward function, it assumes that the reused model is adapted to its own task and can be applied. In addition, the agent needs to create a formal representation of policy and provenance data that can be interpreted and reused by other agents with similar tasks. In this way, the framework meets the requirement that agents can cooperatively share and reuse their acquired policies. The provenance or meta data about the policies acquired should a) ensure accountability and b) enable agents to assess the suitability and quality, i.e. Quality of Service (QoS), of the policies provided. Therefore, the provenance data has to provide statistical information about how the policies were trained and obtained. The RDF serialisation (see Listing 4.14) shows an example of provenance data referencing the origin of the policies and describing

statistical metrics for policies of a task.

The meta description of the policies contains the version number of the task description and the number of episodes needed to train the policies. It also specifies the threshold value that separates positive from negative reward values and indicates the percentage of positive and negative reward values that were achieved during an evaluation run. The obtained average positive/negative reward value is also given. This is followed by information about the originator (agent) of the policies as well as information about the algorithm used and the configuration parameters, i.e. hyper-parameters with which the corresponding algorithm was initialised. The example also references the developer of the agent and the task for which the agent trained the policies. At the end, a URL is given that references the web resource location of the policies. Of course, the provenance or meta data can be extended with additional relevant information.

Listing 4.14: Example of a semantic policies meta description trained for a task [155].

```
:CheckboxBySynonymTaskPolicy :version "1.0".
:CheckboxBySynonymTaskPolicy :numberTrainedEpisodes "1000"^^xsd:integer.
:CheckboxBySynonymTaskPolicy :hasReward _:reward.
_:reward :hasValue _:positiveValue.
_:reward :hasValue _:negativeValue.
_:reward :hasThreshold _:threshold.
_:threshold :hasValue "0.7"^^xsd:double.
_:positiveValue :greaterThanOrEqual _:threshold.
_:positiveValue :occurenceFactor "0.89"^^xsd:double.
_:negativeValue :lowerThan _:threshold.
_:negativeValue :occurenceFactor "0.11"^^xsd:double.
:CheckboxBySynonymTaskPolicy :hasMeanPositiveReward "0.8"^^xsd:double.
:CheckboxBySynonymTaskPolicy :hasMeanNegativeReward "0.2"^^xsd:double.
:CheckboxBySynonymTaskPolicy :trainedBy _:agent.
_:agent rdf:type :Agent.
_:agent rdf:label "DQNNAgent".
_:agent :appliesAlgorithm "DQNN".
_:agent :discountfactor "0.75"^^xsd:double.
_:agent :learningRate "0.01"^^xsd:double.
_:agent :greedyValue "0.1"^^xsd:double.
_:agent :trainedForTask :MiniWobTask.
_:agent :developedBy :Jon_Doe.
:CheckboxBySynonymTaskPolicy :url <http://sem-agent-fw#Policy1>.
```

Since agents can also monitor their activities and changes in the environment, they can also provide valuable log records that can be used to derive MDPs and corresponding task entity descriptions. As outlined in Sec. 4.5.2.2, meta data about the dataset is required for the automatic derivation of MDPs via a wrapper component. For this reason, agents who want to make their datasets available to the agent framework also have to provide meta

data about their produced datasets.

It is up to each developer how their agent implements decision-making processes internally. However, in order to be compatible to the agent framework presented here, the process should correspond to the main process described here and the purpose, i.e. knowledge generation and exchange, of an agent should correspond to the goals of the agent framework.

In the following, it is illustrated how the main program of an agent that is participating in the framework, might look like. The first part of the agent program deals with querying the task and agent profile to initialise the agent and provide information about the topics the agent wants to subscribe to.

Subsequently, a client instance is generated that allows the agent the event-driven and bi-directional communication with the devices or control elements of the task environment. The client establishes a permanent connection with the provided server URL and the agent subscribes for the topics referenced in the task profile.

Then some variable declarations have to be made in order to store interim results.

Since the communication between agent and environment is event-driven, a callback function is implemented that handles incoming messages from topics the agent subscribed for. It is then in the agent's responsibility to handle, i.e. parse and pre-process, the received messages that contain information about state and environmental feedback. The agent extracts from the message the updated state information and the corresponding reward value that serves as feedback from the environment to the agent.

The agent also keeps track about whether a final state was reached that terminates the task and closes the client. However, as long as no final state is reached, the agent maintains within an array or list the obtained data consisting of the last state, the new state, the last performed action and the obtained reward value for the last performed action.

The agent incorporates the new reward value into its learning algorithm in order to update its internal model representation, e.g. weights of a DNN.

After the agent has incorporated the new feedback into its utility function and model, it selects an action based on the given state and sends via the client instance a message containing information about the action to conduct.

In Sec. 4.5.2.2, semantic concepts and properties were proposed to also specify and provide meta data about records that can be autonomously generated by agents based on their observations. This meta data is used by the wrapper component of the framework to derive corresponding MDPs and task entity descriptions from the log records. The function *createMetadataForDataset()*, as explained in Sec. 4.5.2.2, creates semantic RDF statements,

over the corresponding dataset.

Finally, the agent's main program returns the rules, provenance data and generated records together with their meta data for further use.

4.6.2. Worker Agent

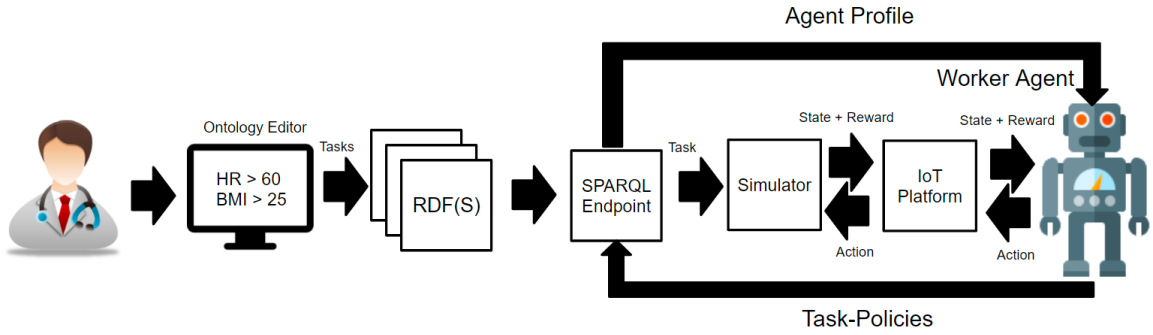


Figure 4.21.: Simulator-based online training of a worker (e.g. DQNN) agent [155].

Worker agents have the main purpose to explore and generate reusable knowledge, i.e. policies and meta data, and thus pass three different phases during their execution as listed in the following.

i. Knowledge generation – i.e. training/mining of default policies.

ii. Tracking of their training progress.

iii. Share of default policies and provenance data – i.e. provide meta data about the learning process.

Worker agents have to be able to incorporate feedback from the environment and to train their model based on received reward signals. Since reinforcement learning agents fulfil the mentioned requirements, all worker agents of the framework implement reinforcement learning.

The trainings procedure for the worker agent is as shown in Fig. 4.21. A domain expert creates a task profile and stores it as RDF serialisation in a RDF knowledge base that is accessible via a SPARQL endpoint. The agent requests the corresponding task- and agent profile from the knowledge base. Simultaneously, the simulator platform subscribes at the IoT platform for the agent's ID and requests the task profile from the knowledge base. The agent and simulator communicate via the IoT platform until a final state is reached and the task is terminated. Afterwards, the worker agent stores its trained default policies and the corresponding provenance data in the knowledge base and adds for the policies and provenance data a semantic reference to the task entity description.

4.6.3. Consumer Agent

Consumer agents are not closely related to the agent framework and can be agents that operate in heterogeneous environments, e.g. on the web, at home, in cyber-physical environments, etc. Thus, they do not belong to the agent framework, but consume the knowledge provided by the framework. However, they are encouraged to also provide their researched knowledge or experience in the form of real operational data so that all agents can benefit and reuse their expertise.

By adapting provided default policies, consumer agents can further improve these policies and eliminate possible ambiguities that may arise during task execution. Ambiguity is also a problem that arises when human domain experts provide task specifications that are either not concrete enough regarding the state representation and provide incomplete information or contain logical errors. For this reason, agents require the ability to adapt policies in order to a) close the gap of incomplete information and b) eliminate the impact of (logical) erroneous state specifications that do not reflect the contextual requirements.

In this way, on the one hand a diverse collection of policies can be provided for different tasks and on the other hand the validity of policies can be ensured. The process steps the consumer agent has to comply with, are as follows:

i. Retrieval of suitable policies.

ii. Adaptation of policies – i.e. adapting policies from feedback of real-world environments.

iii. Reuse of adapted policies.

iv. Provision of real operational data with corresponding meta data and provenance data to the agent framework.

Figure 4.22 illustrates that the consumer agent requests via the agent framework policies that might match to its task properties and requirements. The consumer agent reuses then the policies and adapts them based on the feedback, i.e. rewards, that is provided by the environment. Section 4.5 already outlined different methods implemented in the agent framework that allow the retrieval of suitable policies.

In order to receive state updates and feedback from the environment, the consumer agent subscribes as well as the worker agent to a IoT platform and communicates via the platform with its task environment. Therefore, wrapper components of the IoT platform transform the action commands into device specific commands and device events that provide measured quantities into topic features. However, the transformation between a high level representation into a low level, i.e. device specific, transformation is out of this work's scope and is therefore not investigated further. Though, a recommendation is to use the Web of Things (WoT) (see Sec. 2.2.7) as an approach in order to allow a mapping between semantic entities and device specific implementations.

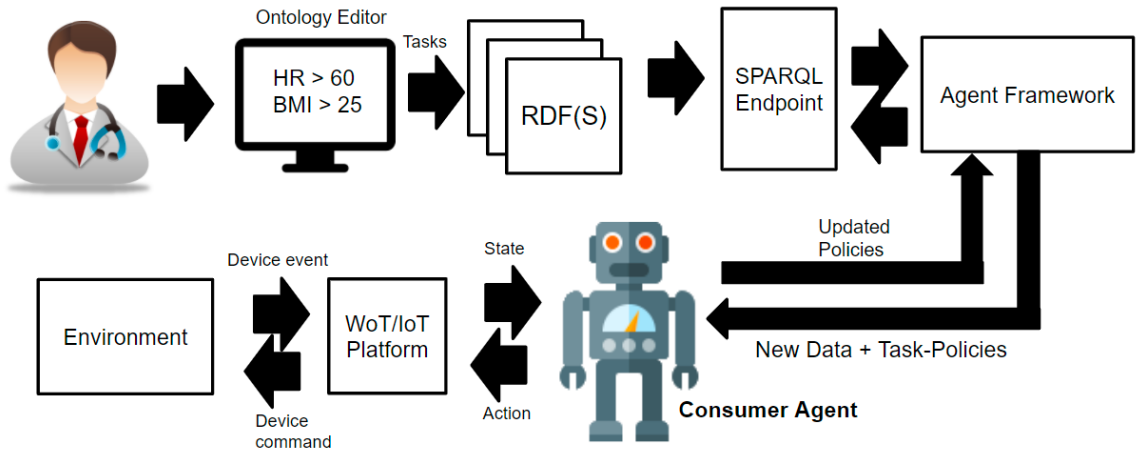


Figure 4.22.: Adaptation and reuse of policies for performing novel tasks conducted by the consumer agent [155].

4.7. Assumptions and Limitations

Any scientific work is usually based on design decisions and assumptions made beforehand, which also set the boundaries of the work, especially if the assumptions can only cover a subset of the problems to be solved in the scientific work. This chapter discusses the assumptions made and limitations of the proposed approach.

A1. Domain experts and developers know what they do. The proposed agent framework covers many aspects, such as domain-independent knowledge representation of heterogeneous tasks, training and deployment of agents, and adaptation or personalisation of policies. However, it does not address control mechanisms for avoiding risks or such issues as privacy and security problems that may arise when, for example, agent developers introduce new program code that may violate the privacy and security of target users. In addition, people may make incorrect assumptions out of ignorance or carelessness that could put the target audience at risk. For instance, domain experts might give wrong benchmarks for states in their task description or fail to consider or forget relevant and important information.

Thus, one assumption made in this work is that the information or task descriptions and configurations provided by domain experts and agent developers are evaluated, safe, correct or well-intentioned and correspond to what the agent is supposed to do.

A2. Deployed tasks are MDPs The proposed agent framework only supports, i.e. simulates, tasks that are MDPs, see Section 2.3.9. This implies that task entities are modelled as MDPs consisting of specific Markov properties and concepts, such as states, actions, rewards for states and actions performed. It is also assumed that agents operate in a *fully observable environment* [213], which means that all sensors necessary for the tasks to be solved are available to observe the environment, while the sensor measurements are denoised in a pre-processing step so that agents do not have to deal with noisy and error-prone data. In addition, a state should be defined in each MDP that occurs as soon as an uncertain or undefined observation occurs. In these cases, the agent either does not need to perform any action or only one action that leads the agent to a safe and defined state.

A domain expert defining a task description needs to know what states can occur within the task, by what features the states can be measured and how the states are rewarded. In addition, he/she requires to know which actions can be performed by an agent that is supposed to solve the task and what immediate effects the actions have on the environment and thus on the fully observable states. Tasks that are not specified as MDPs cannot be processed by the agent framework. Moreover, performed local optimal actions are assumed to lead to a global optimal strategy of the agent, since *dynamic programming* [17] is applied using the *Bellman equation* [17], which is used for the agent's reward function. The iterative execution of actions in different states contributes to the internally maintained reward function reflecting the (long-term) impact and value of the executed actions in each visited state. This also means that every task that can be handled by the agent framework, is a deterministic or stochastic decision process.

A3. Only finite and discrete action spaces are supported Action spaces can consist of either discrete or continuous actions. In the agent framework, it is assumed that each task has a finite set of discrete actions that can be performed by an agent. To give an example of continuous actions, we could imagine that a driving agent needs to control the steering wheel of a car. In this case, the degree of the steering angle is between 0 and 360 degrees and can take on an infinite number of real numbers to control the steering wheel. In contrast, a discrete action does not require a real number of a continuum to determine the scope of the action. The reason why only finite and discrete actions should be supported in the agent framework was that infinite, continuous actions would have greatly increased the complexity of the agent framework and ontology. Thus, dealing with infinite, continuous action spaces is beyond the scope of this work. However, it is possible to convert continuous actions into discrete actions if, for example, binning is performed. In the example of the steering wheel, two actions, e.g. Stir_Left_1_Degree, Stir_Right_1_Degree, might be sufficient to convert continuous actions into discrete actions. Thus, most problems can be solved with discrete actions determined by the finite number of actuators an agent is allowed to control.

A4. Only sequential, i.e. time series data with state labels are supported. Datasets that map MDPs are usually sequential data that show a development based on states and actions performed over time. For this reason, datasets used to generate task entities have to be time series datasets that contain, for each time step, the previously recorded observation features, the action(s) performed and the subsequent observation features showing the change caused by the action(s) performed. One additional requirement is that to the previous as well as subsequent observations, state labels are assigned that characterise different possible states. For instance, the power of entity embeddings, which encode task, state and action entities and are used in the context of task and policy retrieval, can only be exploited if continuous observation features are annotated by predefined state labels.

A5. Communication via semantic task entities The communication between agents and the agent framework is only possible if the agents adhere to the provided framework ontology. This means that task entities and associated entities are required for communication. Therefore, domain experts or agent developers need to know the concepts and properties of the ontology in order to use the services of the agent framework and implement their tasks and agents. For instance, in Section 4.6 et seq., a procedure is defined that needs to be implemented by an agent programme to be used in the framework. Another example can be found in Section 4.5.1, which explains that when searching for similar tasks and their associated policies, a consumer agent needs to provide the framework with its own task entity that corresponds to the ontology and allows comparison with given tasks.

A6. Openness and accessibility of the agent framework The agent framework is intended to be open, i.e. agents can access it via the World Wide Web, and the underlying ontology is available to agents and human users. This implies the need for domain experts and agent developers to make their task descriptions and agent program implementations openly available, e.g. via a knowledge graph, so that others can also benefit from the knowledge

generated, i.e. the task descriptions and shared policies. Thus, proprietary use and access is not supported by the agent framework. The idea is that the knowledge generated by process records, agents and domain experts should be iteratively extended with new task entities and policies that can be shared and reused by everyone on the web.

4.8. Summary of this Chapter

In this chapter, concepts and methods have been introduced to address various identified problems and challenges (e.g. cold-start problem, policy adaptation and search) using the proposed agent framework.

The agent framework envisions different roles (i.e. domain expert, developer, target user, worker/consumer/hybrid agents) involved in different life-cycle steps supported by the agent framework. One particular role supported by the framework are the worker and consumer agents, which serve different purposes within the framework. While the worker agent is closely related to the framework, the consumer agent can be any kind of agent that uses the services of the framework, e.g. for policy retrieval and the reuse of policies for own tasks.

The proposed agent framework supports different process steps such as the deployment of task entity descriptions and agent entity descriptions by domain experts and agent developers as well as the training, provisioning, discovery and customisation of default policies. Each process step was explained in the context of the agent framework.

It has been motivated that the agent framework has the requirement to support among logical rules, the integration of mathematical equations to describe and simulate natural phenomena and processes. The intention was to illustrate that domain experts require the possibility to describe tasks by formal representations such as logical rules or mathematical equations.

Then an ontology developed for the agent framework was presented, which has the purpose of ensuring a shared understanding between different agents and the framework. Furthermore, the ontology is used to enable domain experts, developers and agents to create task and agent entity profiles. The defined concepts and properties provide a means to model relevant knowledge for the agent framework and also to make implicitly encoded knowledge explicit by reasoning techniques. It has been shown that the concepts are designed to represent Markov decision processes in an abstracted way, enabling agents to exchange and evaluate information.

An important component of the agent framework is the simulation component. It has the task of training reinforcement learning agents on newly created tasks by using the knowledge provided in semantic task entity descriptions to update states based on actions performed by agents. In this way, reinforcement learning agents train standard policies to solve the corresponding tasks. The simulation component also enables predictions to be made about short- and long-term effects of behaviour. Agents who follow certain strategies, for example, can thus determine what effects their actions will have and can also select among several policy options those that are the most reward-maximising for them.

For this reason, the simulation component has to be able to determine states based on observable features, update them and infer reward values based on effects and subsequent states. In addition, the simulation component has to be able to handle different types of tasks, e.g. sequential, (a)synchronous. Approaches, e.g. devised algorithms, have been discussed that address exactly these simulation tasks.

The agent framework also uses behavioural data from user demonstrations and consumer agents to extract knowledge, i.e. policy rules, from this behavioural data. Several proven approaches (e.g. FPGrowth, OneR, sequential coverage) have been presented that pre-process the data and derive policies or decision rules depending on the type of observed features and the type of task the corresponding agent has to perform. Thus, the framework has to be able to select the right rule mining algorithms based on the formalised task characteristics. This chapter has outlined selection strategies that enable the framework to select the appropriate rule mining algorithms.

As part of this work, an algorithm (i.e. Best Fit Sequence) was also developed that allows rules to be derived from behavioural data that correspond to an intended, fixed sequence. Stages within tasks are considered and reward-maximising sequences extracted from the corresponding dataset are converted into rules. In this way, it is possible to derive specific policies or rules that can then be reused by agents for their tasks.

It has been argued that (time series) datasets reflecting user behaviour and task processes can also be used to derive hidden Markov models and action effects that can be used for simulation through the agent framework. For this reason, the corresponding datasets have to a) follow a specific representation and b) be pre-processed to sample HMMs and underlying probability distributions that allow the determination of feature trends and labelling of associated states. In order for the framework to interpret the records, meta data describing the characteristics of the records is needed. Semantic concepts and properties have been proposed for structuring meta data to describe and comprehend log datasets. This meta data structure is also relevant for agents who want to provide records and need to describe the records accordingly, see Sec. 4.6.1.

Since the agent framework provides policies for general use, methods are needed to enable agents to find suitable policies based on the characteristics of their task. In this work, two approaches are presented that enable the discovery of suitable policies. Both approaches use entity embedding vectors to transform tasks and their property relationships into a numerical representation and constrain the agent's context, i.e. the agent's state and action space. The first approach is to identify similar tasks using entity embeddings to find strategies that can be reused to solve a task. For this purpose, the task entity description of an agent is used and the similarity is compared with already existing and solved tasks. The policies of the most similar task that is then found are returned to the requesting agent.

In the second approach presented, single policies or sequences of policies for solving a task are assembled based on the current context, i.e. the state of the agent. To do this, the entity embedding context is considered and the actions closest to a given state are executed in a

simulation using an ensemble of agents. The action that promises the highest reward is then selected and stored in a list. This is repeated for each state in the sequence until a final state is reached. The list of determined policies is then returned to the requesting agent. As part of this work, an algorithm called *Policies Combination* was devised and presented that performs this very task of composing policies based on determining closely related states and actions.

The penultimate section on the approach discussed the conceptualisation of agents, as agents using the framework are required to fulfil certain criteria and interfaces. The corresponding section presented the general programme structure of agents that want to use the framework. A distinction was made between *worker agents* and *consumer agents*, as both serve different purposes. An agent developer who wants to develop and integrate a new agent for the framework has to follow the proposed guidelines presented.

The final section on the approach illustrated the assumptions and constraints made in the proposed agent framework. A total of six different limitations were identified, concerning privacy, security, the data model used, the ontology and the communication with and accessibility of the agent framework. To overcome some of these limitations future work would be required.

5. Use Cases

The objective of this work is to provide a framework for software agents that is domain-independent applicable for heterogeneous tasks. To show the universal applicability of this approach, this chapter provides some descriptions of use cases that served as case examples for the implemented agent framework. The considered application domains comprise the healthcare (HC), the epidemiological domain as well as domestic activities and the execution of automated web tasks. The following sections provide a detailed description of the use cases that were considered during the conceptualisation, implementation and evaluation of the presented approach with respect to the posed research questions (RQ1-RQ3).

The considered tasks do not only vary regarding their domains but as well regarding their task characteristics, that are listed in Table 5.1. The task characteristics comprise *sequential* versus *unordered* execution, *synchronous* versus *asynchronous* execution and *single-agent* versus *multi-agent* participation.

Sequential means in this context that the appropriate task is only solvable if ordered action sequences are executed. For instance, the *automated web tasks* are sequential tasks for which the agent has to learn ordered action sequences, while the other use cases do not require fix ordered action sequences for achieving the envisioned task objectives.

A *synchronous* task is a task where the environment's state only changes after every participating agent has performed its action. Thus, the framework waits until all agents completed their actions, while in an *asynchronous* task, every agent acts independently from the other agents. There are also use cases in which multiple actors (agents) cooperate synchronously by their actions and influence the state of the environment and lastly the outcome of all actors. The state of the environment is updated only if every agent has performed its action in each time step. Thus, discrete execution steps of all agents over time characterise the task progress.

Tasks can contain rules as well as differential equations that enable the framework to simulate phenomena and their changes with respect to time and external impact factors. This means that some tasks require mathematical models that base on simplified assumptions in order to represent the evolution of states. The *COVID-19 measures* task is a task that is simulated by differential equations (i.e. compartmental models) that describe how the number of COVID-19 cases evolve depending on the taken measures respectively actions. The difference here compared to the other tasks is the adoption of differential equations in order to simulate the task dynamics while the other tasks are specified by simple rules represented by boolean expressions. Moreover, the *COVID-19 measures* task

has contradicting objectives, i.e. public health versus growth and stability of the economy, leading to a trade-off that has to be addressed by the acting RL agent.

Task	Communication	Single/Multi	Execution	Diff. Equations
Web Tasks	Asynchronous	Single Agent	Sequential	No
Clinical Pathway	Asynchronous	Single Agent	Unordered	No
COVID-19 Measures	Asynchronous	Single Agent	Unordered	Yes
Domestic Activities	(A)synchronous	Single/Multi Agent	Sequential	No

Table 5.1.: Characteristics of the use case tasks.

5.1. UC1: Automated Execution of Web Tasks

The following use case specification is taken from the paper [155] that has been published and presented at the K-Cap 2019 conference in Marina del Rey, California.

Web agents can perform on behalf of the user, tasks on the web, such as scraping websites or filling out web forms in order to submit orders or book flights. To do so, an interpretable state representation of the corresponding web page is required. Web pages are usually structured as a Document Object Model (DOM) tree. This DOM tree consists of numerous web interaction elements, such as buttons, text fields, selectable lists, check-boxes, radio buttons, etc. Based on this DOM tree the current state of a web page is represented and can be parsed. Every user interaction on a web page invokes a state change on the representation of the web page. For instance, if a radio button was clicked, then the radio button changes into the state *selected* and the next interaction can be performed in order to finalise a web task.

Certain web pages require the execution of sequential actions that follow a predefined process path. This implies that web agents require to learn ordered action sequences for solving a web task successfully.

In addition, textual instructions are necessary that provide information and guide the agent through the task completion process. These textual instructions are considered also as part of the state representation and can vary depending on individual requirements of the task.

To evaluate the performance of web agents, the *MiniWob++*[1] benchmark is utilised. MiniWob++ is stated on its Github page as an extension of the *OpenAI MiniWob* [228] benchmark. MiniWob++ provides a set of mini web tasks that can be accessed and solved via the *Selenium WebDriver*[2]. The Selenium WebDriver is a browser automation tool that empowers web agents to test, request, parse and interact with web applications. Selenium supports

[1] `https://stanfordnlp.github.io/miniwob-plusplus/`
[2] `https://www.selenium.dev/`

different web browsers, such as Firefox, Chrome, Internet Explorer and Opera.

For this work, different web tasks were tested and evaluated. The tasks are: *clicking checkboxes by their label or by synonymous labels, selecting list options by their label, clicking buttons by their label, clicking button sequences and booking either the cheapest or the shortest flight.* The tasks vary from easy tasks (e.g. select list options) to more complex tasks (e.g. book a flight). The complexity is given by the length of required action sequences that have to be performed. In the following sections every evaluated task is shortly introduced.

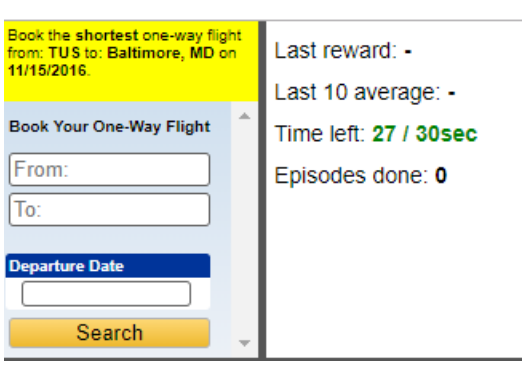
One of the RL agents was pre-trained using the simulation component of the proposed agent framework in order to later evaluate whether the pre-training by the proposed agent framework has a positive impact on the performance of the evaluated RL agent using the MiniWob++ benchmark. The goal of the corresponding evaluation is to test whether the *cold-start problem* can be solved using the proposed agent framework and its integrated simulation component.

5.1.1. Flight Booking Task

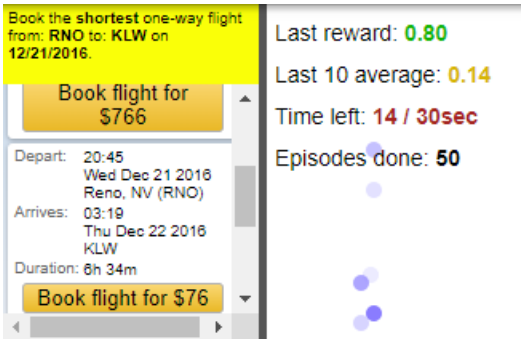
The flight booking task requires interactions on two subsequently following pages. On the first page, the agent has to select the departure airport and the destination airport as well as the flight date from a date picker element. Afterwards, the given *search-button* has to be clicked. A second page is loaded if the previous actions were successfully performed. On the second page, the agent gets randomly the instruction to book either the shortest or the cheapest flight. Only if the correct flight is chosen and submitted by the agent, it is rewarded by a positive value for the finalisation of the task. The maximum reward, an agent can get is 1 and the maximum punishment value is -1.

Every task is time restricted. For instance, for the flight booking task, the agent has at maximum thirty seconds time to finalise the booking. Is the time passed and the agent has not yet finished the task, a penalty that yields in a negative value (i.e. -1), is assigned to the agent by the benchmark.

Figure 5.1 depicts the MiniWob++ task representation of the flight booking task. The corresponding web elements on the first page (see Fig. 5.1a) are two lists for selecting the departure and destination airport as well as a date picker element for selecting the flight date. The search button is for submitting the flight search request. Subsequently, the second page (see Fig. 5.1b) is loaded with the found flights that are categorised by the costs and duration of a flight. The envisioned flight can be selected by the corresponding *Book Flight* button. On top of the page in the yellow text area, textual instructions are listed. In the flight booking task, the agent has either to select on the second page, the cheapest or the shortest flight. On the right panel, some statistics (e.g. last reward, average reward, already done episodes) and the time limit and remaining time for finalising the task are depicted. The web tasks can be executed in several iterations in order to get the average performance of the acting agent.



(a) The first page of the flight booking task.



(b) The search results of the flight booking task.

Figure 5.1.: The flight-booking task execution sequence

5.1.2. Click Check-box by Label or Synonymous Label

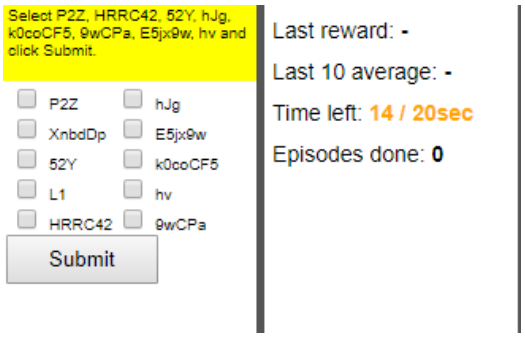
The check-box task has two variants. In the first variant, the agent has to select one or multiple check-boxes, with respect to the labels that are listed in the instruction text. In the second variant, the agent has to select check-boxes that relate to the synonyms of the labels that are listed in the instruction text. Therefore, an internal dictionary dissolves the appropriate synonymous labels.

The difficulty for the agent is that the DOM state representation is equal between both tasks, while the textual instructions and actions for solving the task are different. This leads to an ambiguous state representation that is first resolved during the execution of the task by the dynamically appearing text instructions. However, when the agent learns the policies by means of the agent framework's simulation component, it only perceives the general DOM tree as a state representation without the instruction text, since the instruction text is unknown during the simulation and learning phase and only appears depending on the goal and context of the task. This ambiguity of the task can be used to evaluate whether the agent is able to dynamically adapt its policies to the emerging instructional requirements.

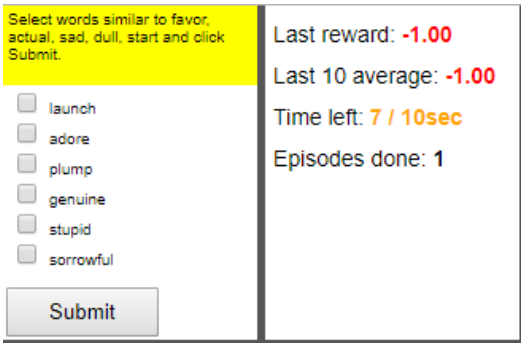
The objective is to demonstrate, that the agent is able through RL to adjust its default policies to upcoming requirements and to resolve ambiguity. Figure 5.2a and Figure 5.2b show the two variants of the check-box selection task.

5.1.3. Click Button and Button Sequence

In the *click button* task, the objective is to click appearing labelled buttons depending on the labels that are listed in the instructions text, while in the *button sequence* task, the objective is to click buttons in a certain predefined sequence. The difficulty in the latter task is that sometimes buttons are partly covered by other buttons so that the agent has



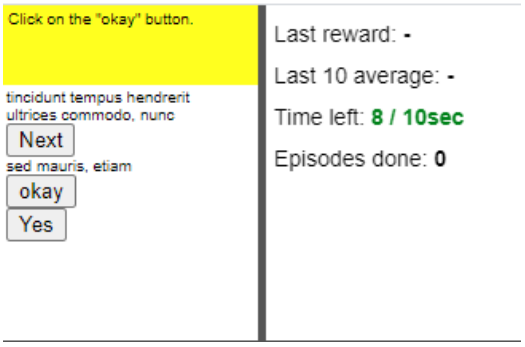
(a) Clicking check-boxes by the listed labels.



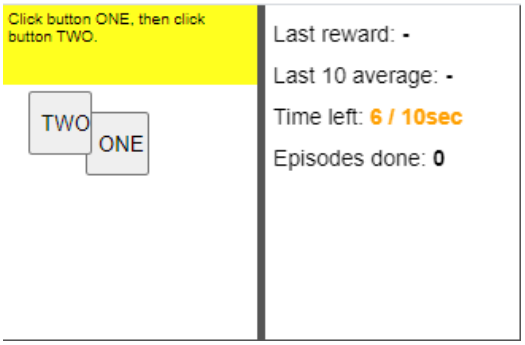
(b) Clicking check-boxes by synonymous labels.

Figure 5.2.: The two variants of the check-box task.

to decide which of the buttons to click first. Figure 5.3 depicts the views of both button related tasks. The time for finalising this task is 10 seconds.



(a) Clicking buttons by label.



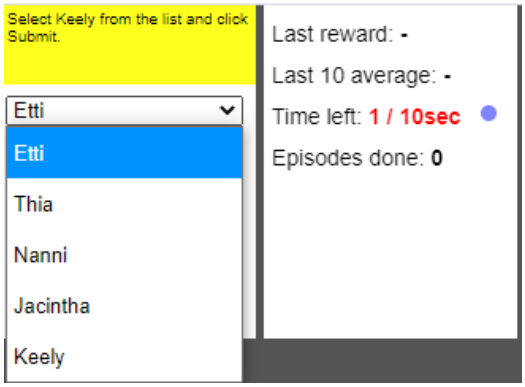
(b) View of the button sequence task.

Figure 5.3.: Button related web tasks.

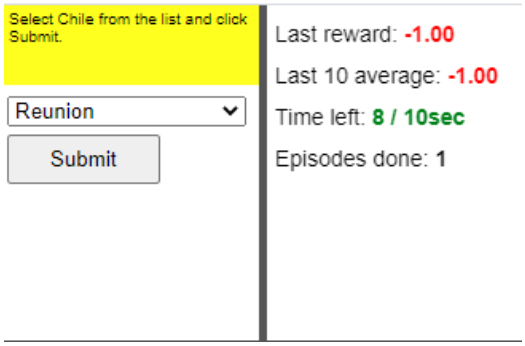
5.1.4. Click List Option by Label

In this web task, the agent has to select list options by their label depending on the listed labels in the instruction text. Subsequently, the agent has to click a *Submit* button for finalising the task. The time for solving the task are 10 seconds.
The previously discussed web tasks and their properties are described in an aggregated task specification (JSON-LD file)[3] consisting of possible states, state rules, properties and actions. The corresponding possible actions are listed below, while the state rules are specified afterwards. The task representation consists of twelve states with corresponding rules, which are used to detect whether a certain defined state applies or not. The state *WebTaskFinished* is the target or final state that ends the task. Each state rule consists of

[3] `https://raw.githubusercontent.com/nmerkle/K-Cap-2019-Demo/master/task.json`



(a) Selection of list options



(b) Open List view

Figure 5.4.: List Option Task

categorical features (e.g. IsLabeledButton) that can either have the value 1, which stands for *true* and 0, which stands for *false*. The features are linked via comparison and Boolean operators, i.e. $<$, $>$, $\leq$, $\geq$, ==, AND, OR, XOR.

For the web tasks, a total of eleven actions were defined that can be executed to interact with the corresponding web elements, such as labelled buttons, submit button, list options, check boxes, date picker. The corresponding actions trigger different user interface (UI) elements and allow selection by label or by synonym. An implemented demonstrator with installation and execution instructions can be found on Github[4].

The following listing shows the actions that an agent can perform within the considered and evaluated web tasks, see also [155]:

(1) SelectOptionByLabel – Selects a list option by the label given in the instruction text.

(2) ClickCheckboxByLabel – Clicks check-boxes by the label given in the instruction text.

(3) ClickCheckboxBySynonym – Clicks check-boxes by the synonymous words given in the instruction text. The synonyms can be obtained from a dictionary or can be requested from the web[5].

(4) ClickSubmitButton – Clicks a submit button. This finishes the web task.

(5) ClickButtonByLabel – Clicks a button by its label specified in the instruction text.

(6) EnterFlightDate – Enters the flight date, specified in the instruction text.

(7) EnterFlightTo – Enters the flight destination, specified in the instruction text.

[4] `https://github.com/nmerkle/K-Cap-2019-Demo`

[5] see `https://www.openthesaurus.de/` or `https://api-ninjas.com/api/thesaurus`

(8) EnterFlightFrom – Enters the departure location, specified in the instruction text.

(9) ClickSearchButton – Clicks the search button for searching flight offers.

(10) BookCheapestFlight – Books the cheapest flight from the given list of search results.

(11) BookShortestFlight – Books the shortest flight from the given list of search results.

The following listing represents the rules that determine the corresponding states (here in in bold) of the web task. The rules help the framework's simulation component as well as rule-based agents to recognise environmental states:

1. **OptionSelectableByLabel:**
   IsOption == 1 AND ListOptionClickedByLabel == 0 AND LabelsInInstructionText == 1 AND IsCheckbox == 0 AND IsFlightDatePicker == 0 AND IsFlightFromField == 0 AND IsFlightToField == 0

2. **CheckboxSelectableByLabel:**
   IsCheckbox == 1 AND LabelsInInstructionText == 1 AND CheckboxClickedByLabel == 0 AND IsOption == 0 AND IsFlightDatePicker == 0 AND IsFlightFromField == 0 AND IsFlightToField == 0

3. **CheckboxSelectableBySynonym:**
   IsCheckbox == 1 AND LabelsInInstructionText == 1 AND CheckboxClickedBySynonym == 0 AND IsOption == 0 AND IsFlightDatePicker == 0 AND IsFlightFromField == 0 AND IsFlightToField == 0

4. **SubmitButtonClickable:**
   (IsSubmitButton == 1 AND SubmitButtonClicked == 0) AND ((IsCheckbox == 1 OR IsOption == 1) AND (CheckboxClickedByLabel == 1 OR ListOptionClickedByLabel == 1))

5. **ButtonSelectableByLabel:**
   (IsLabelButton == 1 OR IsSubmitButton == 1 OR IsSearchButton == 1) AND ButtonClickedByLabel == 0 AND LabelsInInstructionText == 1 AND IsCheckbox == 0 AND IsOption == 0 AND IsFlightDatePicker == 0 AND IsFlightFromField == 0 AND IsFlightToField == 0

6. **FlightDateSelectableByPicker:**
   IsFlightDatePicker == 1 AND FlightDateEntered == 0 AND IsCheckbox == 0 AND IsOption == 0 AND IsSubmitButton == 0 AND IsLabelButton == 0 AND IsSearchButton == 1 AND IsFlightFromField == 1 AND IsFlightToField == 1

7. **SearchButtonClickable:**
   (IsSearchButton == 1 OR IsSubmitButton == 1 OR IsLabelButton == 1) AND SearchButtonClicked == 0 AND IsCheckbox == 0 AND IsOption == 0 AND IsFlightDatePicker == 1 AND IsFlightFromField == 1 AND IsFlightToField == 1 AND FlightFromEntered == 1 AND FlightToEntered == 1 AND FlightDateEntered == 1

8. **FlightFromEnterable:**
   IsFlightFromField == 1 AND FlightFromEntered == 0 AND IsCheckbox == 0 AND IsOption == 0 AND IsSubmitButton == 0 AND IsLabelButton == 0 AND IsFlightDatePicker == 1 AND IsSearchButton == 1 AND IsFlightToField == 1

9. **FlightToEnterable:**
   IsFlightToField == 1 AND FlightToEntered == 0 AND IsCheckbox == 0 AND IsOption == 0 AND IsSubmitButton == 0 AND IsLabelButton == 0 AND IsFlightDatePicker == 1 AND IsFlightToField == 1 AND IsSearchButton == 1

10. **ShortestFlightBookable:**
    IsBookCheapest == 0 AND IsBookShortest == 1 AND FlightDateEntered == 1 AND FlightFromEntered == 1 AND FlightToEntered == 1 AND SearchButtonClicked == 1

11. **CheapestFlightBookable:**
    IsBookCheapest == 1 AND IsBookShortest == 0 AND FlightDateEntered == 1 AND FlightFromEntered == 1 AND FlightToEntered == 1 AND SearchButtonClicked == 1

12. **WebTaskFinished:**
    SubmitButtonClicked == 1 OR ButtonClickedByLabel == 1 OR CheapestFlightBooked == 1 OR ShortestFlightBooked == 1

5.2. UC2: Virtual Coaching based on Clinical Pathways

Nowadays, in the healthcare (HC) domain, intelligent agents (i.e.*virtual coaches*) play an increasing role for supporting sustainably the treatment and well-being of patients and elderly persons. The European Union (EU) commission promotes and funds many calls and projects (i.e. vCare[6], council of coaches[7]), in the healthcare domain, with the long-term objective to provide high-end assisting technologies in hospitals and at home that support on the one hand healthcare professionals in their work and on the other hand, patients and elderly people to live a healthy, self-determined, socially active and independent life as long as possible. For this reason, the health sector automates medical tasks concerning examinations, treatments and after-care so that patients get an all around care, while healthcare professionals are relieved of their workload. In the long-term, costs and (human) resources are intended to be decreased by technological, i.e. IoT and smart devices and intelligent agents. This trend requires that medical treatments and clinical pathways (CPs) have to be reproduced by formalised healthcare processes. Agents have to be enabled to follow the CP inherent processes and make context and user dependent decisions regarding supportive actions that contribute to the recovery and health of the patient.

In particular, the objective is to empower virtual coaching agents to make activity recommendations based on medical guidelines, the patient's medical records, the current health state and context. Thereby, different data are provided that drive the decisions and recommendations of the virtual coaching agent. These data comprise categorical and numerical features, e.g. diseases, preferences, health states and corresponding rules, based on medical guidelines. However, before this section dives into the details of medical use cases that are considered within this work, first a definition of clinical pathways shall be adopted that lays the foundation for a common understanding. A clinical pathway according to De Bleser et. al. is defined in [57] as:

Def.: "... a method for the patient-care management of a well-defined group of patients during a well-defined period of time. A clinical pathway explicitly states the goals and key elements of care based on Evidence Based Medicine (EBM) guidelines, best practice and patient expectations by facilitating the communication, coordinating roles and sequencing the activities of the multidisciplinary care team, patients and their relatives; by documenting, monitoring and evaluating variances; and by providing the necessary resources and outcomes. The aim of a clinical pathway is to improve the quality of care, reduce risks, increase patient satisfaction and increase the efficiency in the use of resources."

Taking this definition into account, clinical pathways can be described by MDPs, since they consist of decision points, respectively (health) states that drive subsequent treatment actions. The patient history can be represented in predefined states that summarise the given health status of the patient to fulfil the Markov property of MDPs. Pathway-based

[6] https://vcare-project.eu/

[7] https://council-of-coaches.eu/

coaching is expected to have a positive effect on the patient's health status in the long term. Usually, one or several caregivers and physicians work together while care professionals provide a patient profile that abstracts relevant information for the agent's decision making.

Demographic data about the patient as well as medical records of disease history are proving important for tailoring treatment plans, activity and exercise recommendations that are integrated into activities of daily living to promote the patient's recovery. The problem, however, is that these various data about the patient and the corresponding medical guidelines are usually not available electronically and machine-readable, and treatment recommendations are often based on different expert opinions and ultimately on the individual experience of the physicians. Thus, a computationally processable representation is needed that reflects commonly agreed medical guidelines and relevant medical data.

The agent framework requests from domain experts a formal representation that abstracts from the complexity of information and allows to incorporate and aggregate heterogeneous data from different sources, i.e. medical records, guidelines, vital signs observations, etc., into a simplified task representation that guides the agent's policy learning procedure. The personalisation is then accomplished by adapting the learned default policies by data obtained from user feedback.

To demonstrate how the agent framework facilitates the coaching abilities of virtual coaching agents, one CP use case is considered and evaluated on the basis of medical guidelines, datasets and clinical pathways.

The considered and evaluated *Cardiac Disease (CD)* use case is inspired by Kaggle's *heart disease* dataset[8] that consists of recorded vital sign data of cardiac patients.

The presented use case represents a clinical pathway (CP) that is applied for CD patients. The following section discusses in detail the corresponding task, its task specific representation and its objectives as well as how the task is incorporated into the agent framework.

5.2.1. Cardiac Disease Pathway

The derived CD pathway that is based on Kaggle's heart disease dataset[9], consists of vital sign indicators (blood pressure, blood sugar, cholesterol, etc.), demographic data (i.e. gender, age) and states (e.g. *UserHasDiabetes, UserHasCholesterol, UserHasHeartdisease* etc.) that reflect the health state of the patient. The dataset provides a means for deriving characteristic patterns and recognising cardiac diseases in order to intervene by medical treatment measures.

[8] `https://www.kaggle.com/tentotheminus9/what-causes-heart-disease-explaining-the-model`

[9] `https://www.kaggle.com/tentotheminus9/what-causes-heart-disease-explaining-the-model#The-Data`

The dataset features are adopted for the specification of states that reflect possible health states of the patient. The activity recommendations follow medical guideline sources that were published online[10] [11] [12]. According to the guidelines, indicators and risk factors for a heart disease are high blood pressure, overweight, high cholesterol, diabetes and the prevalence of heart disease cases in family history. Moreover, factors like stress and activities like smoking and frequent alcohol consumption can foster the emergence of heart diseases. Demographic factors, i.e. age, gender can also have an impact to the susceptibility to heart diseases. Moreover, life-style behaviour culminating in a lack of physical activity, poor nutrition, etc. can also negatively influence the health of the heart. To counteract heart disease facilitating behaviour, appropriate recommendations and treatments can be made that improve the patient's health state.

For the CD pathway, demographic features were omitted, since they turn out to be not reliable as risk factor indicator for heart disease. Furthermore, the objective is that an agent shall only learn to make recommendations based on its made vital sign observations. The considered features, states, state rules, actions and effects for the decision process are listed in the Tables 5.2, 5.3, 5.4:

Features	**Type**	**Range**
Cholesterin	Categorical	0 or 1
Diabetes	Categorical	0 or 1
Smoker	Categorical	0 or 1
Physical Active	Categorical	0 or 1
Heart Disease	Categorical	0 or 1
BMI	Numerical	15 to 35
Heartrate	Numerical	30 to 160
Systolic Value	Numerical	50 to 220
Diastolic Value	Numerical	30 to 160

Table 5.2.: Features of heart disease pathway.

Table 5.2 lists observable features that are either categorical or numerical and can have specific ranges of values. For categorical features, the possible values that a feature can take are either 0 or 1.

Table 5.3 shows possible states and their corresponding rule expressions that determine the appropriate state. The rules allow the agent framework the recognition of states. For instance, if the feature *PhysicalActive* is set to 1, it means that the patient is currently physically active. The expressions can be also more complex by concatenating the expressions

[10] https://www.mayoclinic.org/diseases-conditions/heart-disease/diagnosis-treatment/drc-20353124

[11] https://www.heartfoundation.org.au/heart-health-education/are-you-at-risk-of-heart-disease

[12] https://www.heart.org/en/health-topics/heart-attack/understand-your-risks-to-prevent-a-heart-attack

with boolean operators (e.g. see *FinalHealthState*).

States	**Rules**
PatientIsActive	PhysicalActive == 1
PatientIsInactive	PhysicalActive == 0
FinalHealthState	Cholesterol == 0 AND Diabetes == 1 AND (SystolicValue >= 100 AND SystolicValue <= 120 AND DiastolicValue >= 60 AND DiastolicValue <= 80) AND (HR >= 60 AND HR <= 80) AND (HeartDisease == 1 OR HeartDisease == 0) AND (PhysicalActive == 0 OR PhysicalActive == 1)
InitialHealthState	Cholesterol == 1 AND Diabetes == 1 AND (SystolicValue >120 AND DiastolicValue >80) AND (HR >80) AND HeartDisease == 1 AND PhysicalActive == 0
NormalBP	SystolicValue >= 100 AND SystolicValue <= 120 AND DiastolicValue >= 60 AND DiastolicValue <= 80
HighBP	SystolicValue >120 AND DiastolicValue >80
LowBP	SystolicValue <100 AND DiastolicValue <60
NormalHR	HR >= 60 AND HR <= 80
HighHR	HR >80
LowHR	HR <60
HasHeartDisease	HeartDisease == 1
HasDiabetes	Diabetes == 1
HasNoDiabetes	Diabetes == 0
HasCholesterol	Cholesterin == 1
HasNoCholesterol	Cholesterin == 0

Table 5.3.: States and rules of the heart disease pathway. For implementation purposes, *InitialState* and *FinalState* have been introduced as states, although they violate the requirement that an MDP cannot be in two different states at the same time. However, it can be said that a context or state consists of all partial states defined here and all partial states are observable in each time step and specify an overall state and context.

Table 5.4 lists the actions that cause state changes by their effects, see Sec. 4.2.2. An action can have one or more effects to the current state of the environment. For instance, eating low fat nutrition will yield long-term in a decreased BMI. Every effect description allows the agent framework to update states based on performed actions.

Actions	**Effects**
EatLowFat	DecreaseBMI
EatFruitAndVegetables	DecreaseBMI
EatLowSodium	DecreaseSystolicValue, DecreaseDiastolicValue
ModerateExercise	IsPhysicalActive
QuitSmoking	IsNotSmoker
TakeBPMedication	DecreaseSystolicValue, DecreaseDiastolicValue
TakeCholesterolMedication	IsNotCholesterol
TakeDiabetesMedication	IsNotDiabetes
MuscleRelaxation	IsPhysicalActive
DeepBreathing	DecreaseHR

Table 5.4.: Actions and their effects in the heart disease pathway.

5.3. UC4: COVID-19 Measures for Public Health and Economy

In the beginning of the year 2020, the emerging COVID-19 pandemic led to numerous attempts of the government and investigations of science [22, 80, 84, 35, 272] to develop strategies, means and computational applications that aimed at understanding and containing the pandemic. Every country in the world reacted more or less restrictive to the pandemic by imposing measures with the aim to keep the COVID-19 cases low. The reasons for this were the high number of infections and the scarce resources of the healthcare system. The high contagion rate forced the government to impose lock-downs and severe restrictions of public life. As the example of Italy showed, an overload of the healthcare system led to high death rates, especially among the older population. The reproduction number (R) was taken by epidemiologists as indicator for the spread or decrease of COVID-19 infections. In this context, the objective was to keep the reproduction number as low as possible, i.e. $R < 1$, which means that one infected person infects less than one additional person. However, the reproduction number was at the beginning of the pandemic higher than 3 due to a lack of knowledge and experience with the novel virus and its consequences. The fast spread of the COVID-19 virus forced some people to shut down all social and public life. The entire economy was more or less down in some countries, except the system relevant branches, such as e.g. health sector and food markets. This development in turn led to heavy economic losses, such as the saving of jobs, unemployment, home office and short-time work. The lack of knowledge as well as the trade-off between public health and economy, made it difficult for politicians to make wise and target-oriented decisions that address both interests, i.e. saving health and lives of the citizens as well as fostering the stability and at best the growth of the economy.

The attempts of science to combat COVID-19 led to different investigations in order to understand the impact and dynamics of the COVID-19 pandemic. Considering the science of epidemiology, the prediction and simulation of pandemics in order to understand them, play a crucial role. In epidemiology, there are known compartmental models (e.g. *SIR, SEIR, SIS, SIRD, MSIR, SEIS, MSEIR, MSEIRS*[13]) that allow the simplified mathematical modelling of infectious diseases and facilitate the simulation and knowledge about the development of pandemics, whereby in this work only two of these models, i.e. SIR and SEIR are considered, since they are sufficient for demonstrating their applicability within the agent framework. Researchers apply these models to predict how pandemics can evolve, depending on certain influential factors. Understanding the dynamics of an infectious disease is crucial for taking countermeasures that help to control and combat the pandemics.

In the following two sections the SIR and SEIR compartmental models are introduced, since the SEIR model was utilised in order to train a RL agent to take measures (e.g. total confinement, soft confinement, some restriction, no restriction) that address the previously discussed trade-off between saving the public health and keeping up the economy. In particular, the objectives are to keep the number of infections lower than the number of

[13] https://en.wikipedia.org/wiki/Compartmental_models_in_epidemiology

available hospital beds as well as not to restrict entirely the public life and economy.

The agent can perform in total four measures to control the economy and the spread of the virus. Table 5.5 explains the performable measures. Every measure can be classified by a restriction level from 1 to 4, while 1 stands for the highest governmental restriction and 4 for no restriction.

This use case was inspired by and adopted from this paper [168] that was written by Miralles-Pechuán et. al. and published at the CIKM 2020 conference. One objective of this thesis, is to demonstrate that the representation and implementation of the COVID-19 use case that was described by Miralles-Pechuán et. al., is feasible by the proposed agent framework, in that sense that RL agents are empowered to learn strategies that address different objectives, even if these objectives are contradictory. For this purpose, the considered use case was transferred into a task representation that allows the agent framework to simulate the dynamics and impacts of the COVID-19 disease as it is described in this work [168].

Measures (Actions)	**Description**	**Restriction Level**
Total Confinement	Total Lock Down. Only system-relevant branches of economy are running.	1
Soft Confinement	Masking obligation. Only doctor visits or going to market are allowed. Only contact with persons of the same household.	2
Some Restriction	Meeting of people from other households are allowed up to 5 persons. Masking obligation only in closed buildings. Activities outdoors are allowed.	3
No Restriction	No restrictions. Everything is as before COVID-19. No safety measures. Economy is fully active.	4

Table 5.5.: COVID-19 Measures with decreasing restriction level.

5.3.1. The SIR Model

SIR stands for Susceptible Infectious Removed, respectively Recovered. The SIR model divides the considered population into the three mentioned categories. At the beginning of an epidemic the number of susceptible persons is usually high while the number of

infectious people is low. In the course of time, the number of susceptible persons decreases the more people are infected by the infectious part of the population. The infected people then change over to the category of recovered or removed people which means that the infected persons are either recovered or dead by the effects of the disease. Moreover, the assumption in the SIR model is that recovered people are immune and cannot become again susceptible or infectious. Persons who are vaccinated fall also into the category of recovered people. It is important to note that the population size in total never changes while only the size of the corresponding categories is changing, depending on the number of infections and recoveries. The members of the population are on any time in one of those 3 categories and switch from one category to another, while the size of the categories can be never less than zero. The changes respectively transitions (see Fig. 5.5) that happen between these three categories are proportional to each other and depend on differential equations that represent the *rate of change* by partial derivatives.

Every category can be described by these partial derivatives (see Equations 5.1 - 5.5) in Leibniz notation. The rate of change for the susceptible category is computed by the differential Equation 5.1, whereby β represents the *contact rate* of one person with other persons and I the number of infected persons. Since the number of susceptible persons S decreases with each infection, the minus sign in front of β indicates the decreasing slope of the susceptibility curve.

$$\frac{dS}{dt} = -\beta IS \tag{5.1}$$

Equation 5.2 allows the computation of the infections curve, whereby γ represents the duration rate of infected persons until their recovery. For instance, if the average recovery time of a infected person lasts D days, then γ is determined by $1/D$. Since the decreasing number of susceptible people is proportional to the number of infections through infected people, the number of infected people I is multiplied with the number of susceptible people S, whereby the rate of recovered people γI is subtracted from the number of new infections.

$$\frac{dI}{dt} = \beta IS - \gamma I \tag{5.2}$$

Equation 5.3 represents the recovery rate that is the recovery rate γ times the number of infected people.

$$\frac{dR}{dt} = \gamma I \tag{5.3}$$

5.3.2. The SEIR Model

The SEIR model is an extension of the SIR model. It introduces *Exposed* as an additional category that reproduces the fact that people may be infected but not yet infectious, since it depends on the incubation time of the virus, when an infected person is also infectious

to susceptible persons. Thus, this model is more fine-grained and accurate than the SIR model which is why the SEIR model was preferred in this work rather than the SIR model. Equation 5.4 is computed by the contact rate β times the number of susceptible people times the number of infectious people minus the rate of people that passed by the incubation time, whereby a is the average incubation time period defined by $1/a$ respectively a^{-1}.

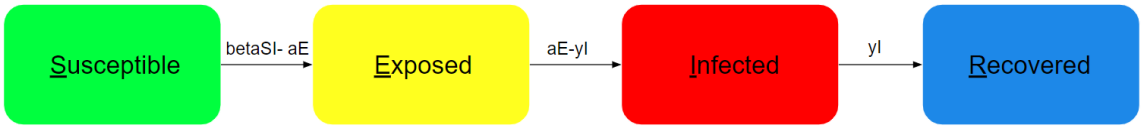


Figure 5.5.: Transition model of the SEIR [1] categories.

$$\frac{dE}{dt} = \beta SI - aE \tag{5.4}$$

The rate of infections I with respect to time t, is reproduced by Equation 5.5. The equation depicts that from the rate of exposed people who passed the incubation period, the number of infected people who recovered, is subtracted.

$$\frac{dI}{dt} = aE - \gamma I \tag{5.5}$$

Since the introduced differential equations maintain a closed system, neither people are lost nor added to the population N which is reflected in Equation 5.6.

$$S + E + I + R = N \tag{5.6}$$

Since in this use case scenario, the number of available or missing hospital beds shall be related to the development of COVID-19 cases, additional rates and equations have to be considered within the representation of the task specification. To reproduce and simulate how the rate of available hospital beds is increasing or decreasing over time with respect to infections and recoveries, differential Equation 5.7 was devised in the scope of this work, whereby B is the current number of available hospital beds, B_{total} the total number of hospital beds within a country, δ the average rate of infections that require hospitalisation, I the number of infections and ω the average recovery rate of hospitalised patients.

$$\frac{dB}{dt} = -(I * \delta) + ((B_{total} - B) * \omega) \tag{5.7}$$

Additionally, it has to be modelled how taken measures respectively actions (see Table 5.5) influence the *rate of change* of susceptible and exposed persons. The action related impact factor that expresses the influence of the given measures to the number of cases, is

represented by the parameter α. The lower the action impact factor α is, the higher is the restrictive character of the measure and this in turn lowers the number of new infections. Therefore, Equation 5.1 and 5.4 have to be expanded by parameter α to Equation 5.9 and Equation 5.10, whereby the value of α ranges between 0 and 1, see Definition 5.8. As example, the action *TotalConfinement* might have an impact factor of 0.25, while taking the action named *NoRestriction* would probably yield in an impact factor of 1.0. In this case, the number of susceptible persons decreases by factor 3/4 faster than in the case of a total confinement measure.

$$\{\alpha \in \mathbb{R} : 0 \leq \alpha \leq 1\} \tag{5.8}$$

$$\frac{dS}{dt} = -\alpha\beta IS \tag{5.9}$$

$$\frac{dE}{dt} = \alpha\beta SI - aE \tag{5.10}$$

With the previously defined differential equations based on the SEIR model, the agent framework is enabled to simulate the dynamics of a COVID-19 pandemic with respect to the given SEIR categories and the number of available hospital beds. However, what still is missing, is a metric that allows taking into account the impact of the given measures to the economy. Here the agent framework provides by its feedback mechanism a means that allows the expression and simulation of economic importance.

The importance of economy can be represented by reward values that are assigned to agents, for imposing a measure. The more restrictive a measure might be, the lower is the reward an agent receives for imposing this measure. However, this also depends highly on factors such as the number of severe illnesses with respect to available hospital beds. The provision of healthcare is likely to be more important than the stability and growth of economy. It depends finally on the value system, priorities and objectives how reward values are assigned for the corresponding measures. If the public health in a country has a high priority, then imposing most restrictive measures will yield in a high reward while less restrictive or no measures will yield in a low reward or even punishment.

The domain expert respectively user of the agent framework has finally to decide the prioritisation of objectives and the accompanying measures. Table 5.6, Table 5.7 and Table 5.8 show three example configurations for rewards that are assigned, depending on the performed measure's restriction level and available hospital beds. For instance, if less than 5% of the existing hospital beds are available and a total confinement is conducted, then the configuration of Table 5.7 suggests to assign a reward value of 10 to the agent.

Since the objective is to keep the exposures and infections low, the proposed configuration of Table 5.7 fosters a high restrictive measure by the high reward value, while the configuration of Table 5.6 punishes in total the upcoming situation of missing hospital beds.

Table 5.8 is a variation of Table 5.6 that keeps the range of rewards and the punishments small, i.e. between -1 and 1 instead of between 1 and -10.

Considering the economic objectives, in all three configurations a high restrictive measure with more than 5% available hospital beds yields in a lower reward value than a less restrictive measure. In this way, the trade-off between saving the public health and the preservation of economy are addressed by fostering measures that contribute to both contradicting objectives.

Later, in the evaluation, the configuration of Table 5.6, Table 5.7 and Table 5.8 are adopted to show what impact the reward assignment configurations and varying hyper-parameter configurations have to the SEIR categories and the performed action sequences.

Measures (Actions)	**Available Hospital Beds**	**Assigned Reward Values**
Total Confinement	≤5%	-2.5
Total Confinement	> 5%	0.25
Soft Confinement	≤ 5%	-5,0
Soft Confinement	> 5%	0.5
Some Restriction	≤ 5%	-7.5
Some Restriction	> 5%	0.75
No Restriction	≤ 5%	-10.0
No Restriction	> 5%	1.0

Table 5.6.: First configuration of reward assignment based on hospital beds and taken measure.

Measures (Actions)	**Available Hospital Beds**	**Assigned Reward Values**
Total Confinement	≤5%	10
Total Confinement	> 5%	-1
Soft Confinement	≤ 5%	5
Soft Confinement	> 5%	-0.5
Some Restriction	≤ 5%	-5
Some Restriction	> 5%	0.5
No Restriction	≤ 5%	-10.0
No Restriction	> 5%	1.0

Table 5.7.: Second configuration of reward assignment based on hospital beds and taken measure.

Measures (Actions)	Available Hospital Beds	Assigned Reward Values
Total Confinement	$\leq 5\%$	-0.25
Total Confinement	$> 5\%$	0.25
Soft Confinement	$\leq 5\%$	-0.5
Soft Confinement	$> 5\%$	0.5
Some Restriction	$\leq 5\%$	-0.75
Some Restriction	$> 5\%$	0.75
No Restriction	$\leq 5\%$	-1.0
No Restriction	$> 5\%$	1.0

Table 5.8.: Third configuration of reward assignment based on hospital beds and taken measure.

5.4. UC5: Domestic Activities

Agents, e.g. in service robotics, operating in domestic environments need to be able to switch between different contexts, i.e. heterogeneous locations, goals and activities, and therefore, have to be able to decide on an appropriate action for each state. A prerequisite here is that agents have to know all action sequences and their relatedness to states in advance before they can be applied in domestic environments. Heterogeneous domestic activities require heterogeneous strategies and the larger the state and action space, the more difficult it is for agents to learn context-appropriate policies. Moreover, contexts may be closely related and actions may be available as options across contexts. In addition, a task or activity can be performed in alternative ways and a challenge for the agent is to find, depending on observed states, the best action sequence that the domestic context requires.

However, the capabilities an agent might have, depends also on its environment and context, since context and environment determine which devices, i.e. actuators, are available and accessible. For instance, in the kitchen an agent may have other devices and thus, options for action than in the living room.

In order to evaluate the earlier discussed problems and challenges in domestic environments, the Virtual Home (VH) [202] dataset serves as ground-truth, as it provides agent programmes consisting of ordered action sequences for domestic activities. Furthermore, it provides for many of the activities also alternative agent programs, so that varying action sequences for same or similar activities and purposes can be considered within the foreseen evaluation (see Sec. 6.3).

The VH dataset consists of 1563 agent programs (i.e. action sequences) of domestic activities, e.g.make coffee, with 1973 atomic actions that can be performed by virtual agents. The dataset is needed for simulating environmental contexts and feedback, enabling the evaluated agents to learn or compose policies.

An agent programme always has the same structure in the given dataset, as shown in Listing 5.1. The agent programme first starts with the name of the activity, then a description text is given in the second line, which characterises the activity. Finally, an action with associated objects is defined in each new line. The enumeration of actions gives a fixed order that is to be followed from top to bottom. The index numbers in each line refer to the corresponding objects on which the action is performed. The index numbers are necessary because the objects are to be addressed distinguishably by the corresponding actions. An activity (see example in Listing 5.2) is thus determined by its action sequence. For this reason, in the context of this thesis activity and task are viewed synonymously:

Listing 5.1: Virtual Home dataset structure [153].

```
1 Name of the activity
2 Natural language description of the activity.
```

```
3
4 [Action 1] <object> (index of object)
5 [Action 2] <object> (index of object)
6 ...
7 [Action n] <object> (index of object)
```

Listing 5.2 illustrates an example activity named *Watch_TV_49* implementing the structure of the VH dataset.

Listing 5.2: Watch_TV_49 data sample [153].

```
1  Watch TV
2  walk to living room, find couch, sit on couch, find remote control,
3  turn on tv by pressing button
4
5
6  [Walk] <living_room> (1)
7  [Walk] <couch> (1)
8  [Find] <couch> (1)
9  [Sit] <couch> (1)
10 [Find] <remote_control> (1)
11 [Find] <television> (1)
12 [TurnOn] <television> (1)
```

What is noticeable is that no states are taken into account in the VH dataset. In the context of the agent framework, the lack of state definitions makes it necessary to define a task description with states that determine the completion of a single action. This is because an agent interacting with the simulation component of the agent framework cannot proceed to the next action and continue the activity until the last action has been successfully executed and the new state has been reached or, detected by the simulator, in which the next action can be executed. Listing 5.3 shows a task entity description of the previously shown domestic task (*Watch_TV_49*, see Listing 5.2) with an example state, observation feature, optional transition, action, and effect entity.

Listing 5.3: The semantic entity description of the activity named Watch_TV_49 in Turtle format [153].

```
1  # Prefix, i.e. Namespace, Definitions
2  @prefix entity: <http://example.org/Entity/> .
3  @prefix property: <http://example.org/Property/> .
4  @prefix concept: <http://example.org/Concept/> .
5  @prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
6  @prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
7  @prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
8
9  # Activity Entity (Watch_TV_49) (Mandatory)
```

```
entity:Watch_TV_49 a concept:Activity;
property:isSequential "true"^^xsd:boolean;
property:hasNumberOfActors "1"^^xsd:integer;
property:hasCommunicationType "Asynchronised"^^xsd:string;
property:hasState entity:Walk_living_room_1_Done;
property:hasState entity:Walk_couch_1_Done;
property:hasState entity:Find_couch_1_Done;
property:hasState entity:Sit_couch_1_Done;
property:hasState entity:Find_remote_control_1_Done;
property:hasState entity:Find_television_1_Done;
property:hasState entity:TurnTo_television_1_Done;
property:hasState entity:InitialState_Watch_TV_49;
property:hasState entity:FinalState_Watch_TV_49;
property:hasObservationFeature entity:IsWalk_living_room_1;
property:hasObservationFeature entity:IsWalk_couch_1;
property:hasObservationFeature entity:IsFind_couch_1;
property:hasObservationFeature entity:IsSit_couch_1;
property:hasObservationFeature entity:IsFind_remote_control_1;
property:hasObservationFeature entity:IsFind_television_1;
property:hasObservationFeature entity:IsTurnTo_television_1;
property:hasAction entity:Walk_living_room_1;
property:hasAction entity:Walk_couch_1;
property:hasAction entity:Find_couch_1;
property:hasAction entity:Sit_couch_1;
property:hasAction entity:Find_remote_control_1;
property:hasAction entity:Find_television_1;
property:hasAction entity:TurnTo_television_1;

# An Exemplary State Entity (Mandatory)
entity:Walk_living_room_1_Done a concept:State;
property:isGoal "false"^^xsd:boolean;
property:isFinalState "false"^^xsd:boolean;
property:isInitialState "false"^^xsd:boolean;
property:hasExpression "IsWalk_living_room_1 == 1"^^xsd:string;
property:hasReward "0"^^xsd:double;
property:hasObservationFeature entity:IsWalk_living_room_1;
property:hasAction entity:Walk_couch_1 .
...

# An Exemplary Observation Feature Entity (Mandatory)
entity:isWalk_living_room_1 a concept:ObservationFeature;
property:hasRangeStart "0"^^xsd:double;
property:hasRangeEnd "1"^^xsd:double;
property:hasFeatureType "NOMINAL"^^xsd:string;
property:hasUnit ""^^xsd:string .
...
```

```
# An Exemplary Action Entity (Mandatory)
entity:Walk_living_room_1 a concept:Action;
property:HasTransition entity:bec16c1e-b08e-496b-ba10-95e85be65fb9;
property:HasEffect entity:SetWalk_living_room_1.
...

# An Exemplary Transition Entity (Optional)
entity:4cd8f07f-c79f-45f9-b872-25b8cbb0e42f a concept:Transition;
property:HasPreviousState entity:Walk_living_room_1_Done;
property:HasNextState entity:Walk_couch_1_Done;
property:HasAction entity:Walk_couch_1;
property:HasTransitionProbability "1"^^xsd:double.
...

# An Exemplary Effect Entity (Mandatory)
entity:SetWalk_living_room_1 a concept:Effect;
property:hasImpactType "ON"^^xsd:string;
property:hasObservationFeature entity:IsWalk_living_room_1 .
...
```

The heterogeneity of contexts in domestic settings is a desirable feature for the intended evaluation, especially with regard to a) the composition of policies in multi-context settings (see Sec. 6.2.3.2), and b) the retrieval and adaptation of similar task policies (see Sec. 6.2.3.1).

5.5. Summary of this chapter

In this chapter four different use cases and tasks of heterogeneous domains were discussed. The use cases respectively tasks range from domestic activities, automated web tasks, clinical pathways to measures for combating the COVID-19 spread while saving the economy. In this context, it was shown which formal concepts are required to implement the use cases within the agent framework. Furthermore, the MiniWob++ benchmark was introduced that has been adopted and utilised for evaluating the automation of web tasks, such as for instance, booking flights. Furthermore, DOM trees as state representation of automated web tasks were introduced.

Considering the medical domain, a clinical pathway (CD) and its task representation with respect to MDPs have been outlined. We have seen that heterogeneous data (i.e. medical records, vital sign data) have to be used and consolidated for clinical pathways, in order to allow agents patient-specific (i.e. personalised) treatment and activity recommendations.

The previously discussed related works regarding the COVID-19 pandemic, inspired and motivated the penultimate use case of this work. In this scope, different existing compartmental models, i.e. SIR and SEIR, were introduced and utilised for the corresponding COVID-19 task. The main objective in this use case that was adopted for demonstration purposes from [168] is to address the trade-off between the safety of public health and stability of economy. In particular, the aim is to demonstrate that the agent framework is able to formally represent and implement (i.e. simulate) different mathematical models that reproduce natural phenomena and long- or short-term effects with respect to time.

The presented COVID-19 use case serves as means for the evaluation of the agent framework. The objective is to demonstrate the applicability and flexibility of the framework in heterogeneous domains and for different types of tasks, as outlined in Table 5.1.

Domestic activities play a particularly important role in service robotics. Agents in domestic environments are confronted with different and changing contexts and have to react adequately. This can be a challenging task, especially with a high number of possible states and actions. The agent framework also tries to implement approaches here that enable the agents to adapt policies on the one hand and to request context-adapted action sequences, i.e. policies, on the other hand.

As a description of domestic activities, the Virtual Home[14] dataset was introduced, which was transformed into semantic task descriptions for evaluation purposes and serves as ground-truth for several of the conducted evaluations in Sec. 6.

Lastly, the understanding of the use cases provides the foundation for understanding and assessing the evaluation of the agent framework with respect to the presented use cases.

[14] `http://virtual-home.org/`

6. Evaluation

This chapter presents the evaluation carried out as part of the doctoral thesis and its results with regard to the use cases presented in Chapter 5 and the research questions and hypotheses presented in Chapter 1.

Each evaluation aspect introduced with the overall objectives of the proposed agent framework is outlined in terms of the posed RQs and hypotheses. In addition, further evaluation-related hypotheses are derived that substantiate the assumptions about the agent framework and the provable evaluation objectives. Subsequently, the objectives, corresponding questions and applied metrics of the conducted evaluation are outlined. The evaluation should show that the agent framework fulfils the goals and requirements of the established hypotheses, RQs and use cases across domains.

6.1. RQ1: Cold-Start Problem

Chapter 1 outlined that prior domain knowledge from the beginning of an agent's lifespan is crucial, because the agent should display immediately, in real environments, a well-defined behaviour in order to solve assigned tasks appropriately. Otherwise, a cold-start of the agent leads to undefined and risky behaviour that can cause damage to the environment, the user and the agent itself. Moreover, the agent has usually no unlimited time to adapt its behaviour to the appropriate task and its requirements. Thus, a fast adaptation to upcoming, individual requirements is not only desirable but necessary.

To show that the agent framework prevents cold-start for RL agents, two other independent agents, i.e. a rule-based agent and an untrained RL agent, were evaluated and their performance was compared with the performance of the RL agent that was trained by the simulation component of the proposed agent framework.

The untrained RL agent faces the cold-start problem, since it has no prior knowledge and has to learn the appropriate policies for solving a task from scratch.

The rule-based agent serves as baseline since it has prior knowledge provided by the task description and the contained state rules that allow the agent to recognise the current state and act appropriate to the related actions that are defined within the semantic task specification. In the use case Chapter 5.1 such state rules are provided as example. Equation 6.1 shows a general representation of state rules. An arbitrary number of

observations and their assigned values build terms that are logically concatenated and indicate together the corresponding state.

$$\text{rule} := \{\text{observation}_1 = \text{value}_1, \text{observation}_2 = \text{value}_2, ...\text{observation}_n = \text{value}_n\} \rightarrow \{\text{state}_n\} \tag{6.1}$$

In the task entity description to every state a set of possible actions is assigned and the rule-based agent can select on each execution step, one of the provided actions. Pre-condition is that the domain expert provides in the related task description the state rules and the related actions. However, if multiple actions are related to one state, the rule-based agent has only the opportunity to decide randomly which action to conduct while it stays unclear if the selected action is the best one that can be selected from the set of alternative actions. This restriction indicates the problem that is given in plain rule-based approaches. The rule-based agent cannot evaluate and determine which of the given actions is the one that maximises the utility and contributes for solving the task in an intended manner. The ambiguity that is given in both check-box tasks (see Chapter 5.1.2), demonstrates as well the limitations of the rule-based approach.

Retrospect RQ1:

RQ1 Can the presented agent framework train different kinds of software agents by formal specification and simulation of tasks in such a way that they show a well-defined and beyond state-of-the-art behaviour in solving different tasks from the beginning of their runtime or at least within a short period of time?

Retrospect H1:

H1 Simulating tasks based on a thorough semantic formalisation of Markov decision processes (MDPs) allows, for software agents, a warm-start in terms of a fast adaptation and positive feedback that reflects the performance of the RL agent.

6.1.1. General Goals addressing RQ1

The aim is to enable domain experts to provide formally specified prior domain knowledge in an abstracted and relative simple way to one or multiple agents, even if the domain experts are not technically inclined and have no knowledge of formal and rule-based languages. At the same time, immediately or after a few execution steps, a high performance in terms of maximising the agent's utility function, should be achieved by the agent. Thus, agents of this framework shall be enabled to achieve a high performance (i.e. positive, increasing cumulative reward) in a relatively short timespan, i.e. a few execution steps and increase or keep this performance constantly high. Considering the problem and requirements of RQ1, the following evaluation questions (EQ) were derived and have to be answered by the corresponding evaluation.

EQ1 Does the agent achieve a high performance, i.e. positive, increasing cumulative reward, within a low number of execution steps?

Based on hypothesis H1, the following evaluation hypotheses can be stated:

EH1.1 The RL agent, trained by the agent framework, requires for the most performed web tasks on average less episodes than the untrained RL agent and the rule-based agent, until it achieves a high performance, i.e. a cumulative reward > 0, especially if ambiguous tasks (see check-boxes tasks) are given, whose state representations are not clear during training time.

EH1.2 The RL agent that is trained by the agent framework, achieves on average a higher reward per episode and consequently a higher cumulative reward for all executed web tasks than the untrained RL agent and the rule-based agent.

Based on these two hypotheses, metrics were derived that are outlined in the next section.

6.1.2. Metrics

The utility or performance of an agent can be measured both by the *cumulative reward* (see Sec. 2.3.7), the agent receives for its performed actions as well as by the *number of execution steps*, the agent requires to solve a task *within an episode*. A task is solved and an episode finished, if all specified goal states and a final state of the task are achieved.

Which criterion to evaluate, depends on whether an agent should solve a task quickly (i.e. with a low number of execution steps) or by gaining the maximum expected reward until the task is solved. Since, in the conducted training phase, the number of episodes are defined by domain experts and therefore are fixed, the objective is to demonstrate that an agent, trained by the framework, can achieve compared to other agents, a high performance in a relatively short timespan, i.e. in a few episodes, while keeping this performance high with respect to the received rewards. The assumption is that both observations, i.e. number of episodes until a utility maximising behaviour is achieved and the received cumulative reward along performed episodes, indicate whether an agent faces a cold-start or not.

To assess whether the agent framework prevents a cold-start the following metrics are applied:

M1.1 The **policy adaptation or learning speed** is measured by the **mean number of episodes** until the agent receives increasing positive cumulative rewards (i.e. reward > 0), (see Eq. 6.2).

$$\mathbf{M1.1}: \frac{1}{N} \sum_{\substack{1 \leq i \leq N \\ 1 \leq j \leq T}} \mathsf{k}_{\mathsf{ij}}, \ \mathsf{k}_{\mathsf{ij}} = \begin{cases} 1 & r_{ij} \leq 0 \\ 0 & r_{ij} > 0 \end{cases} \ \mathsf{r}_{\mathsf{ij}} \in \mathbb{R}, \ \mathsf{k}_{\mathsf{ij}} \in \mathbb{N}, \tag{6.2}$$

where r_{ij} is the obtained cumulative reward value per run and episode, k_{ij} is a constant that depends on the ij^{th} episode's cumulative reward, N is the total number of performed evaluation runs and T is the total number of episodes within one run.

M1.2 The performance of the appropriate agent is evaluated by the **mean cumulative reward among all performed episodes**, (see Eq. 6.3).

$$\mathbf{M1.2}: \frac{1}{N}\sum_{i=1}^{N} r_i,\ r_i \in \mathbb{R}, \tag{6.3}$$

where r_i is the cumulative reward per episode and N the total number of episodes required.

In the following sections, the set-up of the experiments as well as the results are presented and discussed in detail.

6.1.3. Set-up

Five of the web automation tasks that were introduced in Chapter 5.1, are evaluated by means of the MiniWob++ benchmark and were presented in the K-CAP 2019 conference paper [155], together with a demo application that is available on Github[1]. For this purpose, it was required to implement a NodeJS based JavaScript program that allows the different agent types to interact via the Selenium Webdriver with the web tasks of the MiniWob++ benchmark. This program, respectively script contains three agent related functions that allow the execution of the agent-specific program.

Each agent (i.e. rule-based, (un-)trained RL agent) utilises the same semantic task entity description, that defines the task representation in terms of states, actions, features, effects, etc. Thus, every agent has the same set of actions available from that it can select actions to solve the given task at hand. Moreover, every agent is restricted in its execution to 1000 episodes per task. These 1000 episodes are executed 10 times for every agent so that in summary every agent performs 10000 episodes per task. Then the mean of the obtained rewards is adopted for each episode. The execution of 10 runs ensures the elimination of noise and the receipt of more reliable data.

It is important to note that the trained RL agent was trained with the simulator component of the proposed agent framework and not with the MiniWob++ benchmark. This is due to the fact that the training and test data have to be different in order to make an unbiased and valid evaluation, see also the description of the use case in Chapter 5.1. Furthermore, the aim is to evaluate whether the proposed agent framework has a positive effect on the performance of the trained RL agent compared to the other two agents evaluated.

For each agent, the reward per episode as well as the cumulative reward are determined in order to evaluate and compare the performance of the different agents. Figure 6.1 depicts the set-up of the previously described evaluation.

[1] `https://github.com/nmerkle/K-Cap-2019-Demo`

To show that the reward scores obtained per episode were significantly different, the *effect size*, i.e. *Cohen's d* [44] (see Eq. 6.4) together with *confidence intervals* [179] (see Eq. 6.5) were applied to the reward value probability distributions of the compared agents with a coverage of 95%. *Confidence intervals*, provided they do not overlap, clearly indicate whether two probability distributions are significantly different, while *effect size* indicates how pronounced the differences between two probability distributions are. According to Cohen, an effect size between 0.2 and 0.5 indicates a small effect, between 0.5 and 0.8 a medium effect, and an effect size greater than 0.8 a strong effect [44].

Eq. 6.4 depicts Cohen's d effect size calculation:

$$\mathrm{d} = \frac{\bar{x}_1 - \bar{x}_2}{\sqrt{(s_1^2 + s_2^2)/2}} \tag{6.4}$$

where $\bar{x}_1$ and $\bar{x}_2$ are the mean values and s_1 and s_2 the standard deviations of the compared distributions.

Eq. 6.5 shows the calculation of confidence intervals assuming that a Gaussian[2]. probability distribution is given.

$$\mathrm{x_l}, \mathrm{x_u} = \bar{\mathrm{x}} \pm \mathrm{z} \cdot \frac{s}{\sqrt{n}} \tag{6.5}$$

where x_l is the lower bound, x_u is the upper bound of the confidence interval, $\bar{x}$ is the mean value, z is the z-score value[3], s is the standard deviation and n is the sample size of the considered probability distribution.

Both RL agents (i.e. trained and untrained) utilise Karpathy's reinforcejs[4] library that implements different RL algorithms, i.e. *SARSA, Q-Learning, Deep-Q-Neural Network (DQNN), Tabular Temporal Difference Learning* and *Stochastic/Deterministic Policy Gradients.* For the evaluation the RL agents apply the DQNN algorithm, since it supports continuous state features and discrete actions, while the other provided RL algorithms are restricted to finite state spaces.

The trained and untrained RL agent are configured with the same hyper-parameters, i.e. **learning rate (alpha): 0.01** and **greedy value (epsilon): 0.1**, **discount factor: 0.75**, **number of hidden units: 100** and **batch size: 5000**. This configuration was determined by hyper-parameter optimisation as the best performing one. The rule-based agent does not require hyper-parameters and acts only based on the corresponding state rules and related actions that are provided by the defined task entity description[5].

The MiniWob++ benchmark serves as the ground-truth, since it assess the performed actions and determines the finalisation of the mini web tasks, while it provides numerical

[2] Due to the sample size of the evaluation, it can be assumed that the distributions are Gaussian.
[3] For covering 95% of the distribution's data, the used z-score value is 1.96.
[4] `https://cs.stanford.edu/people/karpathy/reinforcejs/`
[5] `https://raw.githubusercontent.com/nmerkle/K-Cap-2019-Demo/master/task.json`

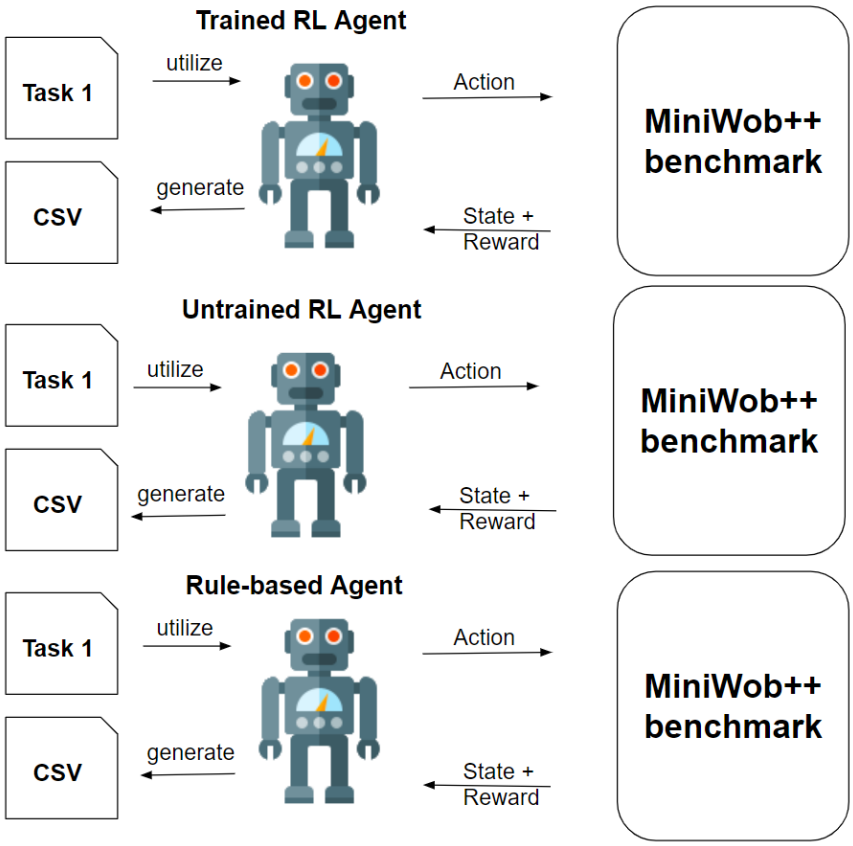


Figure 6.1.: Set-up for the evaluation of automated web tasks. The trained RL agent was previously trained with the simulation component of the proposed agent framework.

feedback to the agent by reward values that range from -1 to +1. If a task is solved correctly and as quick as possible, the agent receives a reward value that is located nearly +1, while not solving the task in time, leads to a punishment of -1. Since the reward value decreases over time, an agent can never receive an exact reward of +1 due to latency when performing an action. In every task episode the agent performs an arbitrary number of actions (i.e. action sequence) in order to solve the task. Ultimately, the agent requires to learn the right sequence of actions in order to solve the web tasks successfully.

To visualise the agent's performance, CSV[6] files are generated that contain for every run and every episode the received mean reward and the cumulative reward. The development of the average reward for all steps completed in an episode and the cumulative reward per episode are visualised by scatter plots. The x-axis of the scatter plots represents the episodes while the y-axis represents the reward values and cumulative reward values. Furthermore, the plotted histograms show the frequency and distribution of positive and negative rewards among all performed tasks.

By means of these plots, it can be determined which agents achieve a high performance within a certain timespan. A steep increase of the curve indicates that the agent learns fast by increasing its utility function with positive reward values. At a certain point the performance stays constantly what shows that the agent has reached its level of optimal experience and stops to learn.

6.1.4. Results and Discussion

This section presents, discusses and interprets the evaluation results, concerning the cold-start problem. Table 6.1 lists for each evaluated agent and web task, the obtained evaluation results.

As previously outlined, the number of episodes until the agent maximises its utility as well as the mean reward per episode and cumulative reward for solving the entire task, were assessed.

Considering the results, it is evident that the RL agent that was trained beforehand by the proposed agent framework achieves for almost every task, besides of the *click button* and *click button sequence* task, the best results regarding the observed and evaluated indicators. It turns out that the trained RL agent shows its strength especially regarding the *click check-boxes* tasks and the *flight booking* task. This can be explained by the fact that both click check-boxes tasks have an ambiguous state representation because the task description provides for both tasks the exact same state representation. Thus, the agent needs to randomly select one of the two possible actions (i.e. clicking check-boxes by label or clicking check-boxes by synonym) that meet the requirements of the task. The distinction between the two tasks is only possible for the agent if the MiniWob++ benchmark gives instructions at runtime as to whether the agent should click check-boxes with the exact

[6] Comma-separated values.

labels or with synonymous labels of the words listed in the instruction text.

The rule-based agent fails most of the time in this task, since it strictly follows the rules that are provided by the task description, and since these rules do not provide a clear distinction between both check-box state representations, the agent can only randomly select one of the click check-box's actions which leads to a 50% probability of either selecting the right or wrong action. The hypothesis and conclusion from the above observations is that the rule-based agent lacks flexibility and thus generalisability because it can only address deterministic decision processes that show no ambiguity and no uncertainty.

In other words, the rule-based agent has the disadvantage that a) it cannot include additional rule-independent information in its reasoning process, since this information is unknown before runtime and b) it adheres strictly to the given rules and learns nothing and therefore cannot adapt itself to the requirements of the task. In order to enable the rule-based agent to perform well, the rules have to be extended and adjusted by the developer, what shows that the rule-based agent is inflexible to changing and previously unknown requirements and states.

In contrast, the trained RL agent has the advantage, that it has previously learned to perform the click check-box actions and by the feedback of the simulation framework, it adjusts its learned policies to the correct actions that are requested by the instruction text. Therefore, the trained RL agent incorporates the words of the instructions text into its state representation which helps to adjust the policies appropriate to the task requirements of the MiniWob++ benchmark.

The RL agent that is untrained and learns its policies from scratch, also manages to perform the task correctly by means of the benchmark's feedback that in turn helps to eliminate the state representation related ambiguity. However, the untrained RL agent requires many more training episodes than the trained RL agent, until it learns the corresponding policies what reveals the cold-start problem.

It can be observed that the results obtained differ significantly in terms of performance (i.e. mean reward scores collected per episode) across agents, as shown by the effect sizes and confidence intervals in Table 6.2 and Table 6.3. For instance, the lower and upper bounds of the different confidence intervals that delimit the distribution of reward scores for each agent do not overlap, indicating that the distributions of the data are outside the 5% mutual significance level and implying that the distributions of reward scores are significantly different. In addition, absolute effect size values greater than 0.8 indicate that strong effects cause the strong differences in the performance of the evaluated agents. Based on these statistics, it can be assumed that the pre-training of RL agents via the simulation component has a significant and strong effect on their performance. Table 6.3 illustrates that between the compared reward value distributions, medium and strong effect sizes prevail depending on the task performed.

Agents	Tasks	**Select List Option**	**Click Button**	**Click Button Sequence**	**Click Check-boxes by Synonym**	**Click Check-boxes by Label**	**Book Flight**
Rule-based Agent	**Episodes until pos. cum. reward**	**1**	**1**	**1**	**2**	**15**	1000
	Cum. Reward	736.822	**967.848**	**767.911**	21.756	15.747	-513.63
	Reward per Episode	0.737	**0.968**	**0.768**	0.022	0.016	-0.514
Trained RL Agent	**Episodes until pos. cum. reward**	**1**	**1**	**1**	37	61	**6**
	Cum. Reward	**823.847**	759.728	654.491	**780.974**	**826.733**	**269.2**
	Reward per Episode	**0.824**	0.760	0.654	**0.781**	**0.827**	**0.269**
Untrained RL Agent	**Episodes until pos. cum. reward**	1000	3	3	852	1000	1000
	Cum. Reward	-72.873	931.947	232.615	92.954	-729.033	-924.13
	Reward per Episode	-0.073	0.932	0.233	0.093	-0.729	-0.924

Table 6.1.: Performance table for every agent and web task with respect to the number of episodes, cumulative reward and mean reward per episode. The bold values show the best results.

Agent type/Task	Confidence interval 95% coverage	Select List Option	Click Button	Click Button Sequence	Click Checkboxes By Synonym	Click Checkboxes By Label	Book Flight
Rule-based Agent	min. value	0.731	0.966	0.756	0.063	-0.002	-0.557
	max. value	0.743	0.970	0.779	0.123	0.034	-0.469
Trained RL Agent	min. value	0.813	0.756	0.641	0.768	0.811	0.239
	max. value	0.835	0.763	0.668	0.794	0.842	0.298
Untrained RL Agent	min. value	-0.098	0.926	0.217	0.003	-0.744	-1
	max. value	-0.048	0.937	0.249	0.04	-0.714	-1

Table 6.2.: The table shows the confidence intervals of the rewards received by the agents per web task. The confidence intervals represent a 95% coverage of the reward values. It can be seen that the confidence intervals do not overlap and thus the results differ significantly.

Compared Agents	Select List Option	Click Button	Click Button Sequence	Click Checkboxes By Synonym	Click Checkboxes By Label	Book Flight
Trained RL Agent / Rule-based Agent	0.62	-4.44	-0.55	2.97	2.95	1.31
Trained RL Agent / Untrained RL Agent	2.91	-2.31	1.74	1.83	6.2	3.81
Untrained RL Agent Rule-based Agent	-2.78	-0.53	-2.37	0.18	-2.74	-0.98

Table 6.3.: The table displays the Cohen'd effect size between the different distributions of reward values obtained by the evaluated agents. The interpretation of effect size values is as follows: +/-0.2 till 0.5 small effect, +/-0.5 till +/-0.8 medium effect, > +/-0.8 strong effect.

Table 6.1 shows in detail that for the *list option, click button* and *click button sequence* task, the rule-based agent as well as the trained RL agent require the same number of episodes until they obtain a positive cumulative reward. However, for solving the *list option* task, the trained agent obtains a higher mean reward per episode and consequently a higher cumulative reward than the rule-based agent, while the rule-based agent outperforms the trained RL agent in both click button tasks regarding both performance indicators.

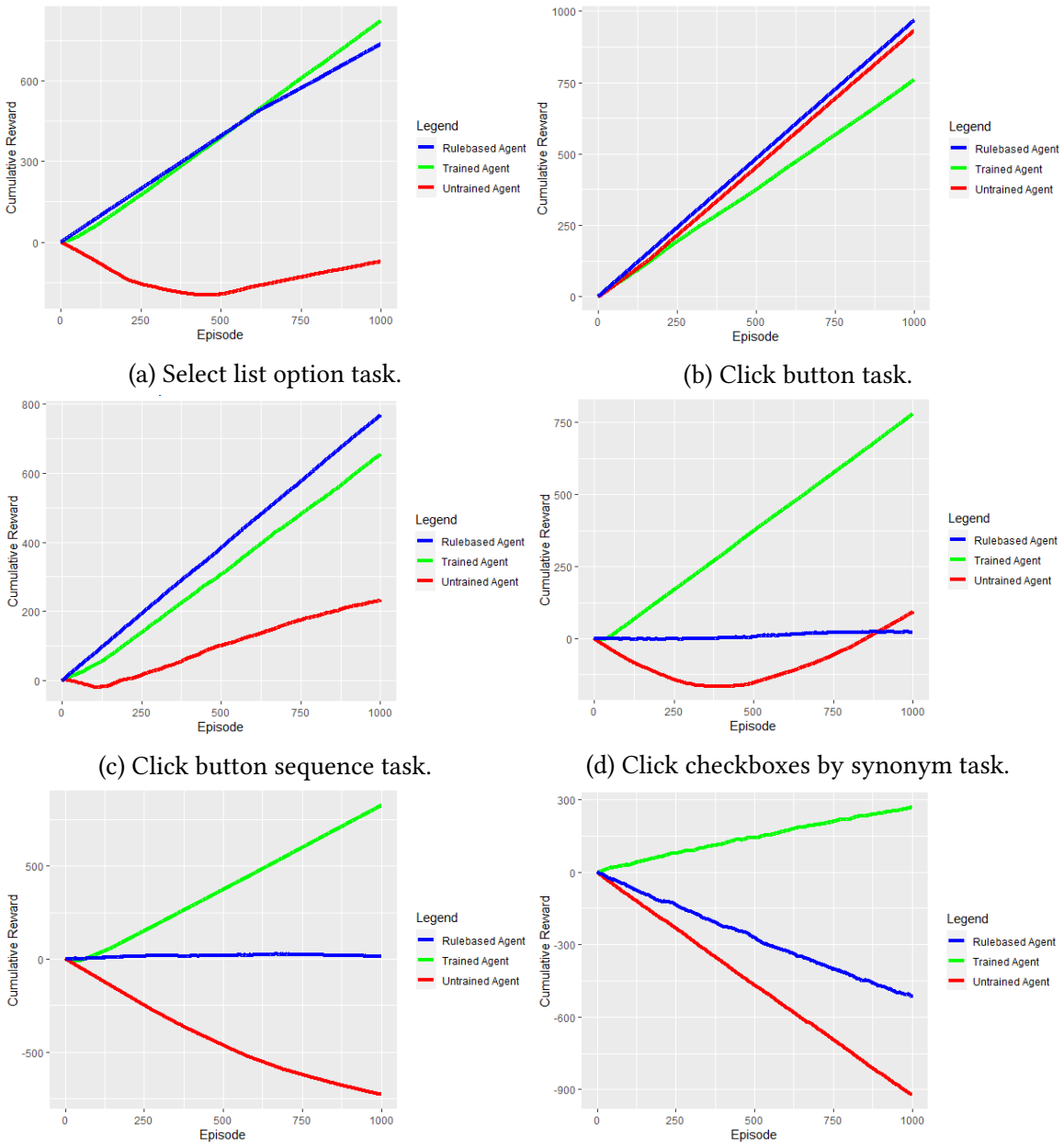
In the list option task, the untrained RL agent fails to obtain a positive reward. It can be assumed that the untrained RL agent requires many more episodes to achieve a comparable performance as the rule-based agent and trained RL agent.

A surprising result is that the untrained RL agent turns out to perform better w.r.t. the obtained cumulative reward than the trained RL agent when it comes to solve the *click button* task. However, the trained RL agent performs better in the *click button sequence* task, which requires more action steps than the *click button* task. However, the trained RL agent requires only 1 episode while the untrained agent requires 3 episodes. Although, at first glance, the trained agent seems to be outperformed by the rule-based agent regarding both click button tasks, it turns out that its performance is nevertheless sufficient and close to the rule-based agent.

For the *click check-boxes* tasks, the trained RL agent requires a few episodes more than the rule-based agent until it receives a positive cumulative reward. However, the trained RL agent outperforms significantly the rule-based agent and untrained RL agent when it comes to assess the obtained mean reward per episode and cumulative reward.

The flight booking task is the most complex task of the evaluated web tasks. Considering this task, the trained agent shows its superiority in terms of performance and adaptation speed, while the rule-based agent and untrained agent show an insufficient performance. The following plots confirm the discussed results visually.

Considering the performance and adaptation time span of the trained agent, the plots show in almost all tasks, the steep increase of obtained rewards in a relatively short timespan (i.e. number of episodes). This observation seems to confirm the assumptions posed in **H1.1** and **H1.2**. Hence, it can be stated that the agent framework helps RL agents to avoid a cold-start and enables a fast adaptation to task requirements.



(a) Select list option task.

(b) Click button task.

(c) Click button sequence task.

(d) Click checkboxes by synonym task.

(e) Click checkboxes by label task.

(f) Flight booking task.

Figure 6.2.: Every agent's cumulative reward for every evaluated web task.

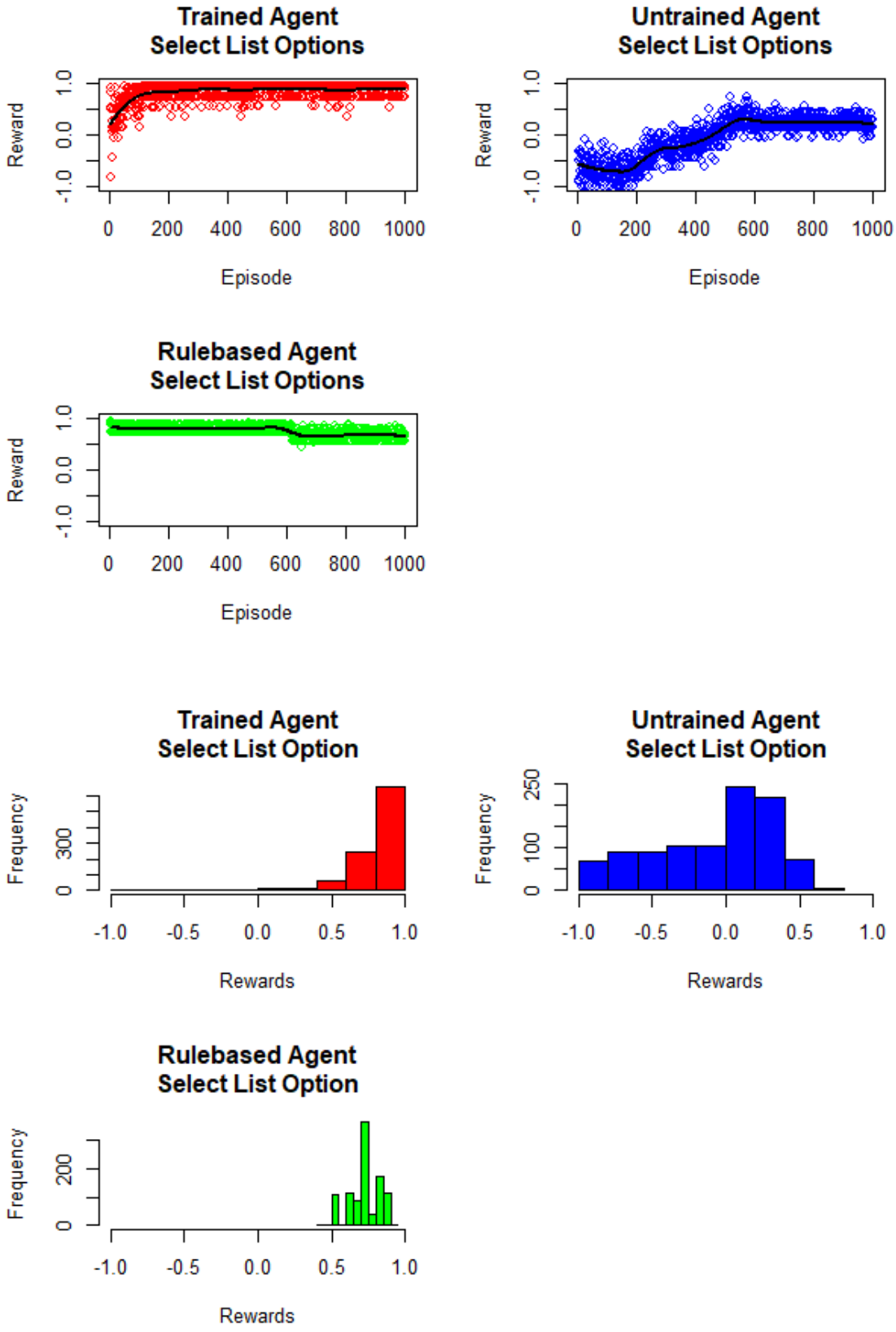


Figure 6.3.: Performance of each agent w.r.t. the select list option task [155].

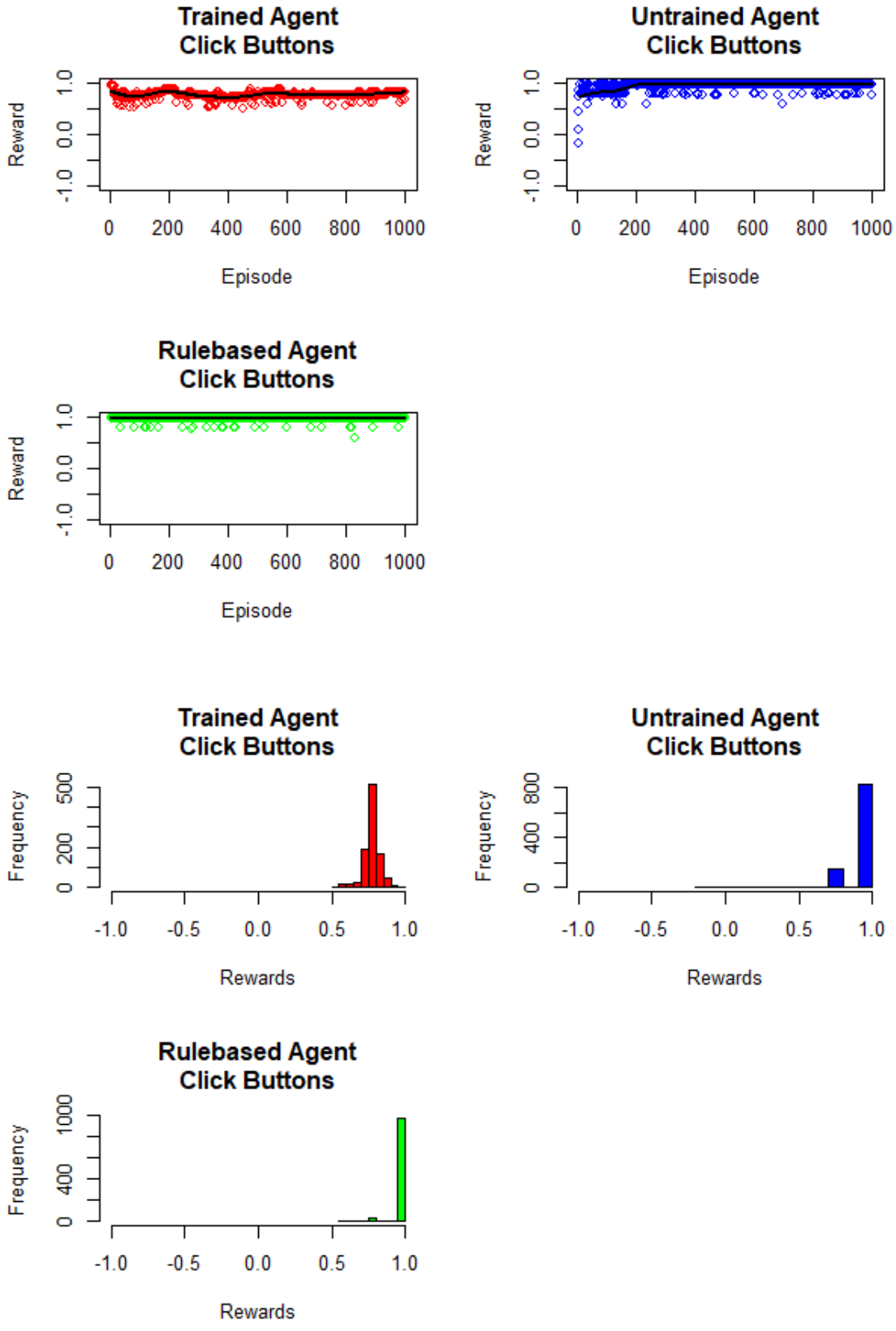


Figure 6.4.: Performance of each agent w.r.t. the select click buttons task [155].

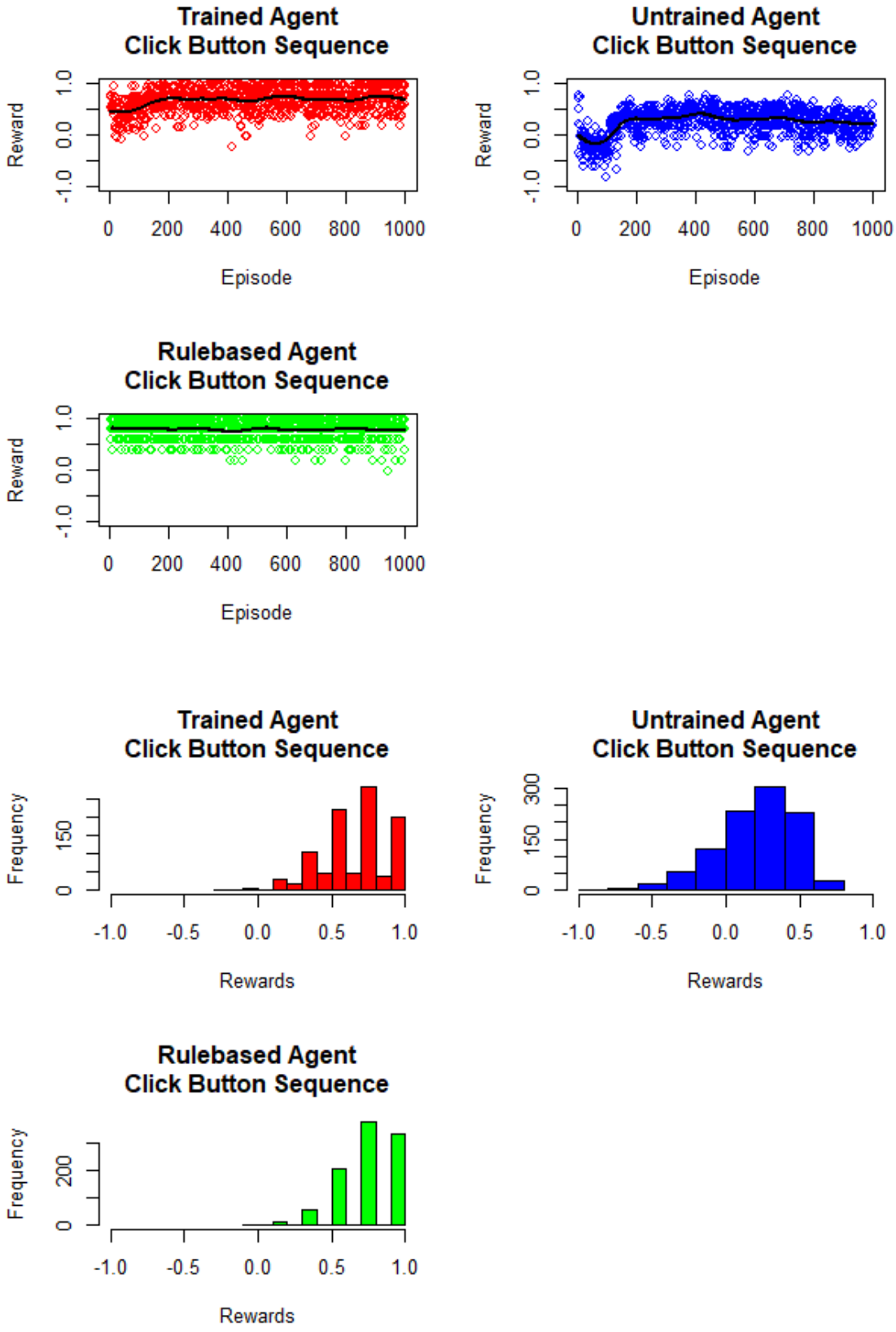


Figure 6.5.: Performance of each agent w.r.t. the select click button sequence task [155].

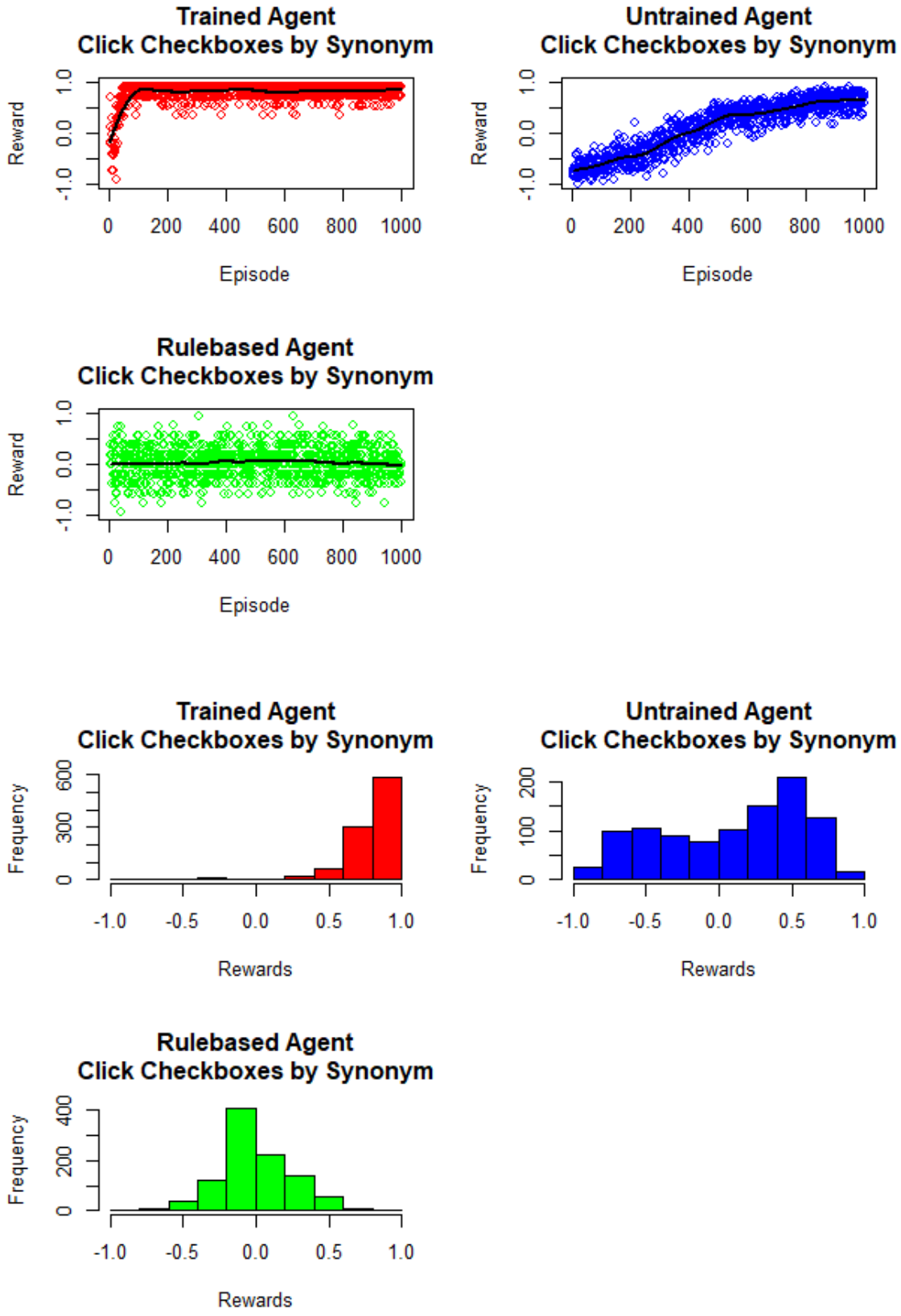


Figure 6.6.: Performance of each agent w.r.t. the click checkboxes by synonym task [155].

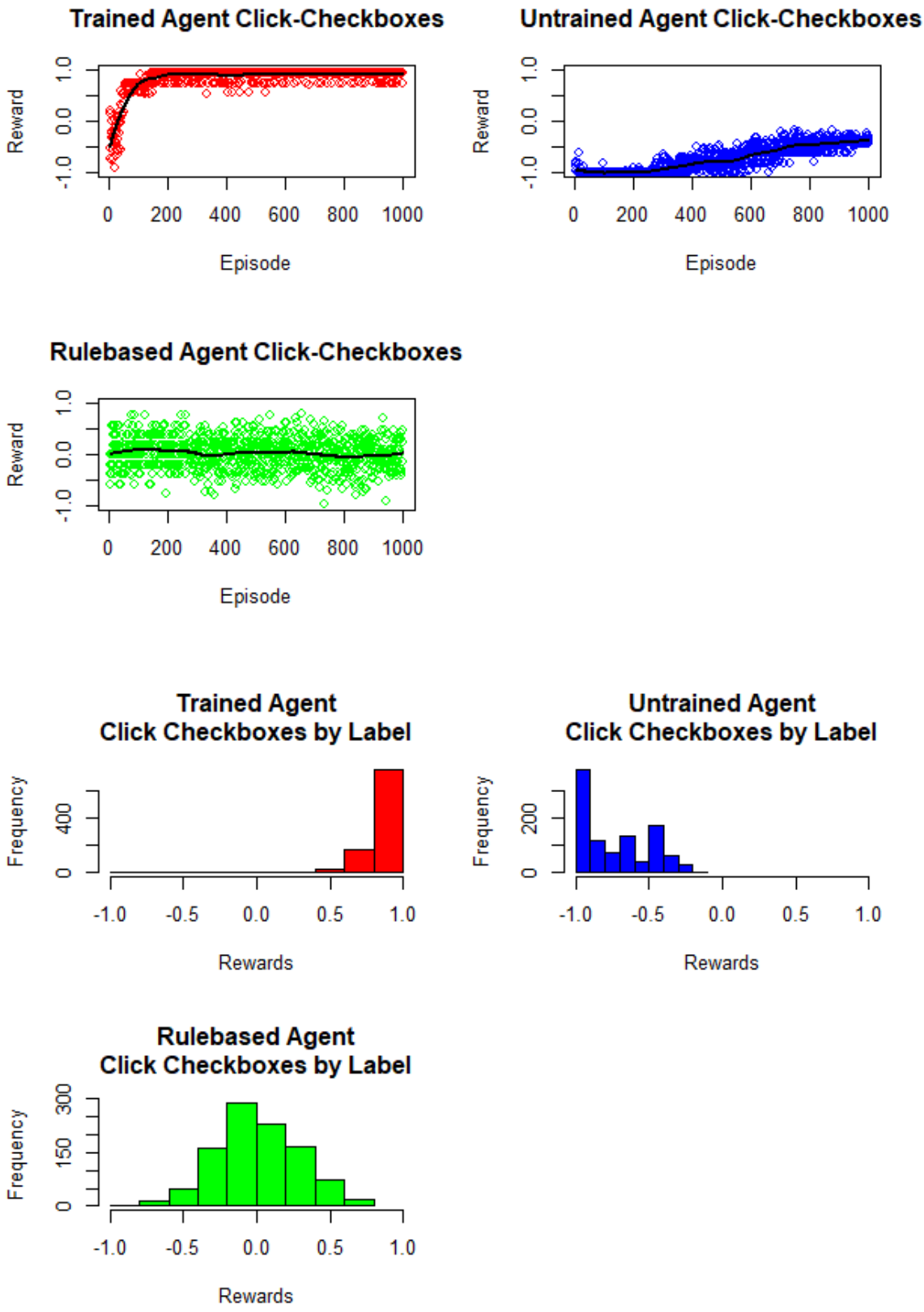


Figure 6.7.: Performance of each agent w.r.t. the click checkboxes by label task [155].

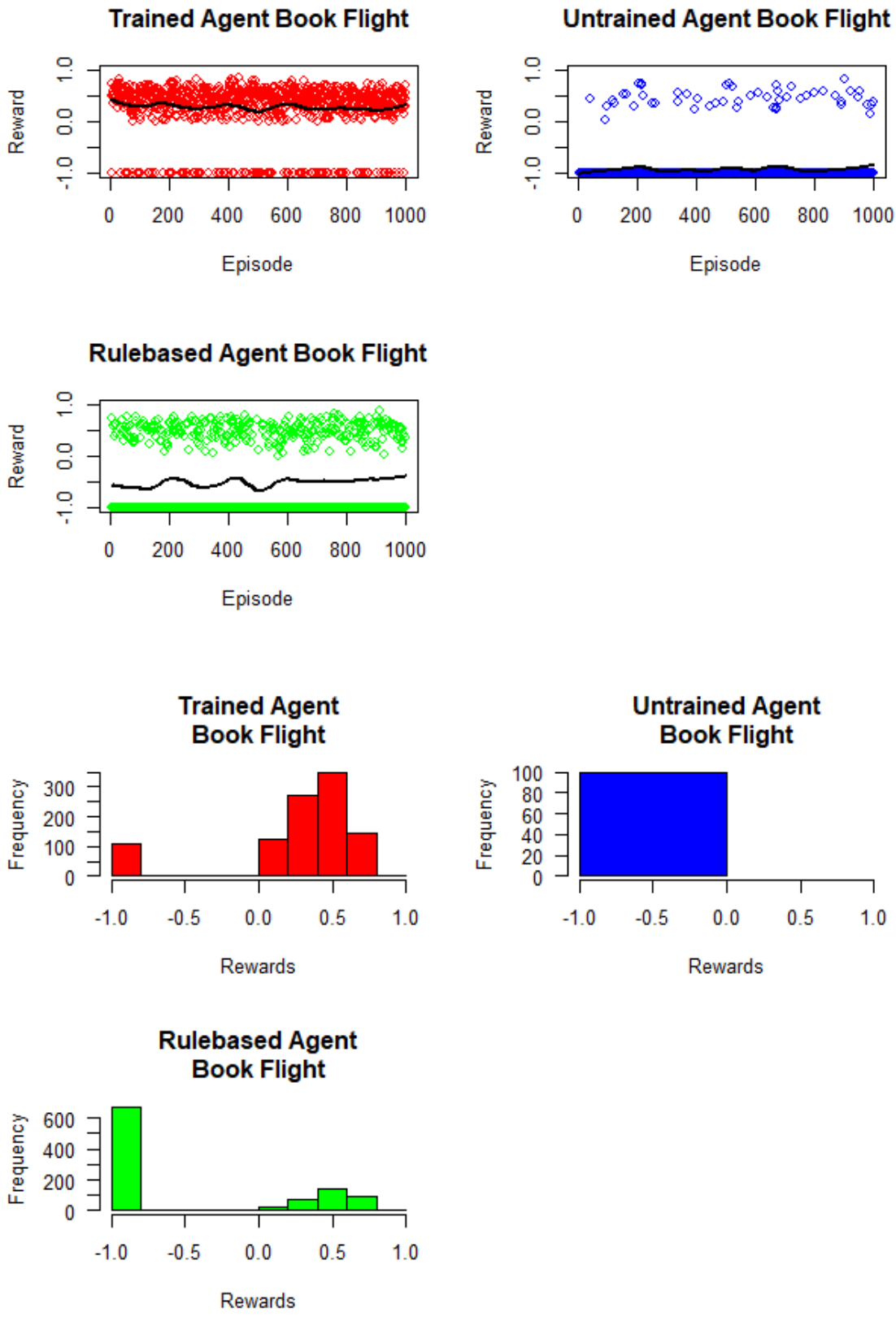


Figure 6.8.: Performance of each agent w.r.t. the flight booking task [155].

6.2. RQ2: Context-Aware Policy Adaption and Disambiguation

Adaptation Domain knowledge is usually hidden in guidelines and domain or application specific data. Thus, an objective is to enable software agents the learning and reproduction of provided knowledge, while eliminating existing ambiguity and biases. This is necessary because the abstraction and formalisation of human knowledge can lead to distorted, incomplete, biased domain knowledge and wrong, ambiguous rules and policies. In addition, generally applicable policies do not necessarily provide the best strategy in specific individual cases where customisation and personalisation of tasks is required. Different factors, such as taking into account the target user characteristics, certain preferences, restrictions, etc. can make it necessary, to adapt general strategies to individual requirements. For this reason, general valid guidelines, which take all eventualities into account, should be adaptable to individual requirements, such as e.g. restrictions, wishes and needs.

As an example, an agent can learn general valid policies for a clinical pathway (CP) that specifies the medical treatment of patients with a heart disease (see use case at Chapter 5.2). The according CP considers all relevant information, concerning the medical history, diagnosis and possible health states of potential patients. Therefore, all relevant and disease specific possibilities and eventualities have to be taken into account. However, every individual patient with a heart disease might differ regarding medical history, health constitution, demographic data, etc. In these individual cases, the CP needs to be adapted to the diagnosis, medical history, context and health constitution of the appropriate patient. A patient with diabetes and overweight might be treated different than a patient who has no diabetes, but is a smoker. Therefore, recommendations in these cases need to focus more on treatments that address the patient's individual impairments, physical constitution, life circumstances and lifestyle, as is mainly required in *precision medicine*.

Another example that makes adaptation necessary, has been outlined as the two *click check-boxes web tasks* were evaluated in Section 6.1. The ambiguity of the corresponding tasks required the ability of the agent to adapt its policies in order to disambiguate the decisions and policies. While a state representation of similar tasks can be identical, in the task entity specification, it may vary during runtime as additional data is available that helps to distinguish the tasks and enhances the state representation. If the agent has learned generally applicable policies for solving a web task, it has to be able to adapt its policies to the current task through the newly given information.

Context-Awareness Agents that have to perform activities in very different contexts, such as in service robotics, have to be able to switch instantly between heterogeneous contexts and adapt their behaviour. In such environments, the state and action space can be very large, making it difficult for agents to find or learn policies that are appropriate to the context and fulfil implicit goals. In particular, when there are many alternatives, i.e. actions, to choose from, an agent has to be able to find and execute the best, i.e. most reward-maximising, sequence of actions without having the necessity to learn all policies in advance [152].

In addition, new contexts can arise at runtime, e.g. when new tasks are created and entered, or devices are added that the agent should be able to use. In such cases, the agent does not have the time to relearn policies, as it has to be able to deal with the new context immediately. For this reason, the agent framework has to be able to provide context-dependent policies by narrowing the search space for policies, i.e. actions and states, especially when there are many solution options from which the agent can choose.

Retrospect RQ2.1 and RQ2.2:

RQ2.1 Can the proposed framework empower a RL agent to correct wrong, ambiguous or unspecific guidelines (i.e. policies) it has learned and adapt them to individual task and user requirements and contextual needs?

RQ2.2 Can the agent framework combine reward-maximising policies across contexts, i.e. across activities, that contribute to the fulfilment of activities and speed up the provision of policies in terms of less training episodes and steps, compared to agents that apply reinforcement learning, i.e. deep-q-learning neural networks (DQNN), for policies learning?

Retrospect H2.1 and H2.2:

H2.1 RL agents can deal with incorrect, ambiguous guidelines and adapt their policies (behaviour) either by changing the task representation or by feedback (e.g. datasets) that provide user and task specific information.

H2.2 The agent framework can numerically represent the context, i.e. the state and action space of tasks or activities, through entity embedding vectors and, based on the spatial distance of the vectors, narrow down the agent's context as well as speed up policy provision. This is done by selecting action candidates that are semantically close to the currently observed state of the agent and running them in parallel in a simulation to determine which action promises the best result for the current state.

6.2.1. General Goals addressing RQ2.1 and RQ2.2

6.2.1.1. Goal Policies Adaptation

Learned ambiguity in domain knowledge and incorrect rules and policies should be adapted or corrected by means of currently observed data and reinforcement learning, so that generally valid guidelines (policies) can be refined and adapted to the specificity of the task, the current context and the preferences and characteristics of the target user. This consideration leads to evaluation question EQ2.1.

EQ2.1 Can agents learn specific actions or strategies via the proposed framework that meet individual requirements and desired behaviour (i.e. high frequency of utility maximising actions) in terms of positive feedback (i.e. cumulative reward)?

The discussed goals and question lead to the following evaluation hypothesis regarding policies adaptation:

EH2.1 RL agents that learn default policies via the agent framework, can adapt their policies according to feedback from the environment and the user, since the feedback reinforces actions that contribute to a positive reward, while it eliminates actions from the data distribution that contribute to negative rewards. Thus, the environment and context determines the performed actions, i.e. future behaviour of the agent.

6.2.1.2. Goal Context-Aware Policies Combination

Agents operating in multi-context environments should be enabled to perform activities without having to learn policies for each context-specific activity. The framework should therefore combine optimal policies that are derived from the agent's context. Even if there are many actions to choose from, an agent should be given the action sequence that is most reward-maximising and helps the agent to obtain the activity that is required and possible in the context. The evaluation questions EQ2.2.1 - EQ2.2.4 as well as the hypotheses H2.2.1 - H2.2.4 outline what aspects of the posed goals are to be evaluated.

EQ2.2.1 Can the presented approach combine reward-maximising policies across contexts, i.e. across activities, that contribute to the fulfilment of activities?

EQ2.2.2 Is this approach able to speed up the provision of policies in terms of less training episodes and steps, compared to agents that apply reinforcement learning, i.e. deep-q-learning neural networks (DQNN), for policies learning?

EQ2.2.3 Can the proposed approach generalise well to unseen tasks, i.e. can it assemble adequate policies that contribute to the successful completion of unseen tasks?

EQ2.2.4 Does the approach show comparable or even better performance than related baseline approaches in terms of executing (unseen and seen) activities consistent with the *Virtual Home* dataset?

H2.2.1 Entity embedding vectors of Markov Decision Process (MDP) entities (i.e. states, actions) help to constrain the state and action space of the agent so that optimal policies for performing activities can be assembled.

H2.2.2 Constraining context through task-related entity embeddings as well as the application of ensembles of agents enable the selection of optimal, i.e. reward-maximising, strategies much faster than applying reinforcement learning, i.e. deep-q learning.

H2.2.3 The proposed approach is able to compose policies even for unseen tasks, provided that the states of the unseen tasks are present in the embedding vectors. Alternative actions can then be combined that lead to the corresponding subsequent states.

H2.2.4 The proposed approach shows comparable performance to considered baseline approaches.

6.2.2. Metrics

6.2.2.1. Metrics Policies Adaptation

By adding or removing states in the task specification, immediately upcoming individual task requirements can be simulated. For example, if a patient has diabetes, a corresponding state indicating diabetes needs to be included in the task entity representation so that the agent is able to recognise the corresponding state and include it in its decision-making. In doing so, each newly arising condition requires a corresponding action to be taken when the condition is met.

M2.1 To evaluate whether an agent has adjusted its policies to task specific requirements, the frequency distribution of all performed actions during task execution has to be considered, (see Eq. 6.6). If the occurrence frequency of an action is 0, this means that the corresponding action has been unlearned, while actions that have a high occurrence frequency have been reinforced by the agent.

$$\mathbf{M2.1} : F(A) = \sum_{i=1}^{N} \mathrm{a_i},\ \mathrm{a_i} = \begin{cases} 1 & \text{action } a_i \text{ was performed} \\ 0 & \text{action } a_i \text{ was not performed} \end{cases} \tag{6.6}$$

where $F(A)$ is the frequency distribution of all possible actions $A = a_i$ that were executed.

In a successful policies adaptation, actions that are no longer required due to the individual task requirements, have to be completely eliminated from the distribution of performed actions or just only occur as single outliers. Vice versa, if new upcoming states are added to the task representation, the agent has to learn and perform the related actions frequently what leads to an increased occurrence of the according actions.

Comparing the distribution of performed actions, resulting from trained and adjusted policies, it can be visualised by bar charts which actions were performed frequently and which actions were eliminated after the trained polices were adjusted.

6.2.2.2. Metrics Context-Aware Policies Combination

Different aspects, i.e. metrics, are relevant to the evaluation of the context-sensitive *policy combination approach*. The evaluation takes into account the **complexity** of an activity by determining the **action sequence length** required to complete an activity. Activities with different complexity, i.e. action sequence length, were tested. Since the *policy combination approach* is compared to deep-q-learning, which is an implementation of reinforcement learning, the speed of policy combination and the number of successfully completed activities are evaluated through different metrics, listed in the following:

M2.2.1 The **speed of policy combination** is defined by the **number of execution steps** required to train or combine policies for each activity.

M2.2.2 The **speed of policy combination** is defined by the **number of episodes** required to train or combine policies for each activity.

M2.2.3 The **activity completion rate** is determined by the **number of successfully completed activities**, (see Eq. 6.7). An activity is considered successfully completed when the action sequence specified in the ground-truth dataset is followed.

$$\mathbf{M2.2.3}: \frac{1}{N}\sum_{i=1}^{N} \mathrm{k_i},\ \mathrm{k_i} = \begin{cases} 1 & \text{activity successfully completed} \\ 0 & \text{activity failed} \end{cases} \tag{6.7}$$

whereby N is the total number of performed activities and k_i is a constant that depending on the activity completion success, evaluates to 1 or 0.

M2.2.4 The development of **cumulative rewards achieved for each activity**, indicates whether the approaches evaluated are performing well, as the reward curve reflects whether the maximum expected reward was obtained or whether more steps were required, due to incorrect actions, before an activity could be completed, (see Eq. 6.8).

$$\mathbf{M2.2.4}: \mathrm{a_t} = \sum_{i=1}^{N} \mathrm{r_{ti}},\ r \in \mathbb{R} \tag{6.8}$$

where a_t is the cumulative reward of the t^{th} activity, N is the total number of performed actions within an activity, r_i is the reward value obtained for the i^{th} action.

M2.2.5 In order to test how error-prone and thus time-consuming an agent was when combining or training policies, the number of incorrect policies was averaged over all activities carried out (see Eq. 6.9).

$$\mathbf{M2.2.5}: \frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{T} \mathrm{k_{ij}},\ \mathrm{k_{ij}} = \begin{cases} 1 & \text{action } a_{ij} \text{ is incorrect, i.e. if } r_{ij} \leq r_{ij-1} \\ 0 & \text{action } a_{ij} \text{ is correct, i.e. if } r_{ij} > r_{ij-1} \end{cases} \tag{6.9}$$

where $r_{ij} \in \mathbb{R}$ is the j^{th} reward value of the i^{th} activity, N is the total number of evaluated activities, T is the total number of actions performed within an activity, k_{ij} is a constant that indicates whether the j^{th} action was performed correctly within the i^{th} activity.

6.2.3. Set-up

The evaluation is divided into two parts. First, the adaptation of ambiguous policies is considered, then the context-sensitive combination of policies, which enables the agent to request policies from the framework according to its current context. The next two sections deal with the appropriate evaluation set-up for each evaluation carried out.

6.2.3.1. Adaptation and Personalisation

For the purpose of evaluation, the clinical pathway (CP) use case (sse Sec. 5.2) for heart disease patients was evaluated. The objective is to demonstrate that the policies can be adjusted to patient specific characteristics.

Figure 6.9 depicts the corresponding evaluation set-up. First, *Agent 1* trains default policies (i.e. a DQNN) by a generic task entity representation, that omits some patient specific states (e.g. *Patient has diabetes*). The training duration is restricted to 1000 episodes. Subsequently, the agent stores a CSV file with the performed actions and received rewards as well as the trained DQNN configuration as JSON file.

A second agent (*Agent 2*) loads and reuses this DQNN configuration for its own task execution. At the same time, the simulator loads an enhanced task specification that consists of additional states (e.g. *Patient is smoker, patient has diabetes*).

Agent 2 performs its actions according to the trained DQNN and adjusts by the feedback from the simulator the DQNN weights. This adaptation takes also 1000 episodes.

Subsequently, *Agent 2* reuses its adapted DQNN in a second run, while it sets the epsilon parameter to zero, in order to omit exploration steps and only execute (i.e. exploit) trained and adapted actions. Afterwards, *Agent 2* saves for every execution step the performed actions and received rewards in a CSV file that later can be inspected for getting insights about the task specific adaptation by plotting the distribution of the actions.

In contrast to the evaluation set-up, in practice agents can either train default policies using the simulator and a corresponding semantic task description and then use them directly for their task, or they can search for already trained default policies that match their task and its requirements. In the latter case, the agents that reuse the policies would need to use their own task description and the framework simulator to adapt the default policies to their own task description through *transfer learning*. This can be achieved, as shown in the evaluation, by changing the composition of states and actions, i.e. adding or removing states and/or actions from the semantic task description.

6.2.3.2. Policies Combination in different Contexts

The evaluation and results presented in this section were adopted from an accepted journal paper[7] [152] published in the Semantic Web Journal (SWJ)[8].

The descriptions of domestic activities provided by the Virtual Home (VH) platform serve as the dataset for training the required entity embedding vectors (see Sec. 5.4). In addition, the VH dataset represents the *ground-truth* against which the success of the combined

[7] https://semantic-web-journal.net/content/context-aware-composition-agent-policies-markov-decision-process-entity-embeddings-and-0

[8] https://www.semantic-web-journal.net

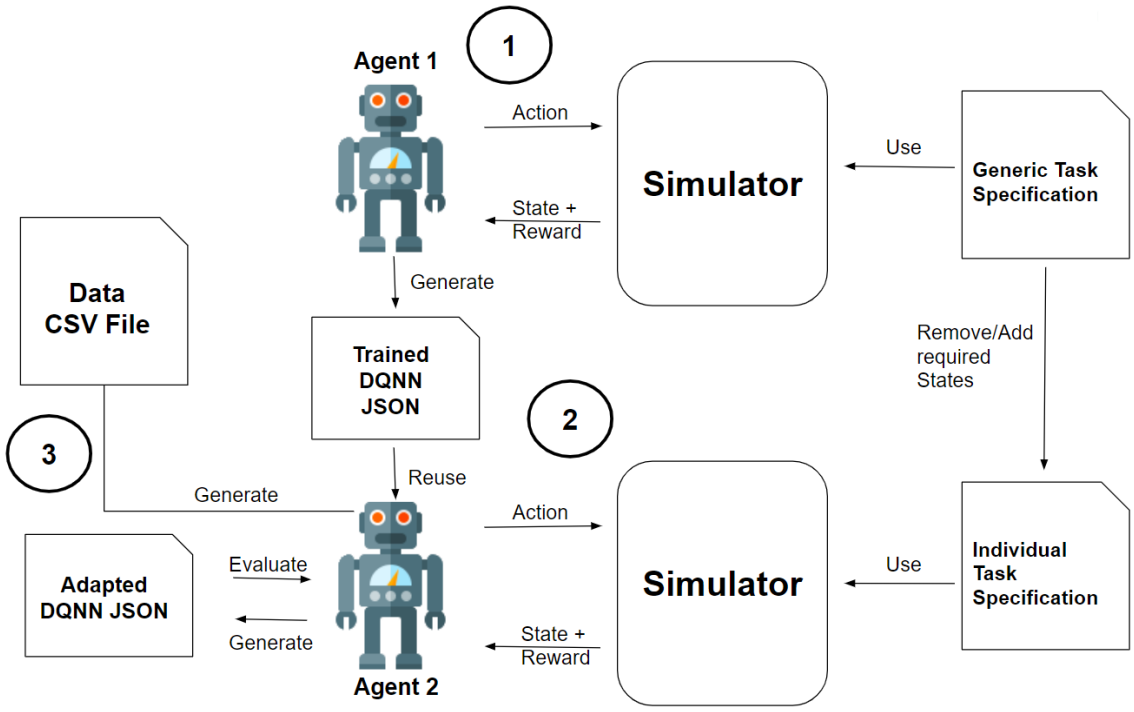


Figure 6.9.: Evaluation set-up regarding the policy adaptation scenario.

policies is tested. Since the VH dataset consists only of activity names and sequential actions, but does not provide named state entities, it was required to construct semantic entity descriptions that also contain states and serve as the basis for the used simulation platform and DNN. The semantic entity descriptions are based on the concepts of the framework's task ontology. Some activities of the VH dataset have redundant names, but differ in some of their actions and in the order in which the actions are performed. To make all activities of the VH dataset unique and unmistakable, the activity names were supplemented with consecutive numbers.

Then, the implemented DNN was trained with entity embedding vectors for all VH activities, states and actions within 1000 iterations. In each iteration, 15 training epochs were executed and a *generator function* generated a batch of 1024 positively and negatively labelled samples for each entity combination.

Fig. 6.10 illustrates the evaluation set-up for the proposed policies combination approach with agent ensembles. The VH dataset is transformed into semantic activity descriptions that are stored in a knowledge graph **(step 1)**, while the VH dataset also serves as input for training entity embeddings that replicate the semantic relationship of states, actions, and activities **(step 2,3)**. The requesting agent randomly selects an activity in the knowledge graph and from this activity a state and sends it to the *Agent Ensemble Manager* **(step 4,5)**, which requests the entity embeddings and the dictionary that maps the states and actions to the corresponding embedding vectors **(step 6)**. The *Agent Ensemble Manager* generates

a predefined number of agent instances corresponding to the number of nearest actions **(step 7)**. The ensemble agents execute their action in parallel threads **(step 8)** and the *Agent Ensemble Manager* receives from the assigned and running simulator instances the corresponding rewards and subsequent states stored in the semantic activity descriptions **(step 9)**. The *Agent Ensemble Manager* stores the action that yielded the highest reward in an ordered list and examines whether it is a target or end state. Step 9 iteratively continues until a final state has occurred. Then, a list with the action sequence and the received rewards and state sequences is reported back to the requesting agent **(step 10,11)**. In this way randomly 52 activities with different sequence length were evaluated.

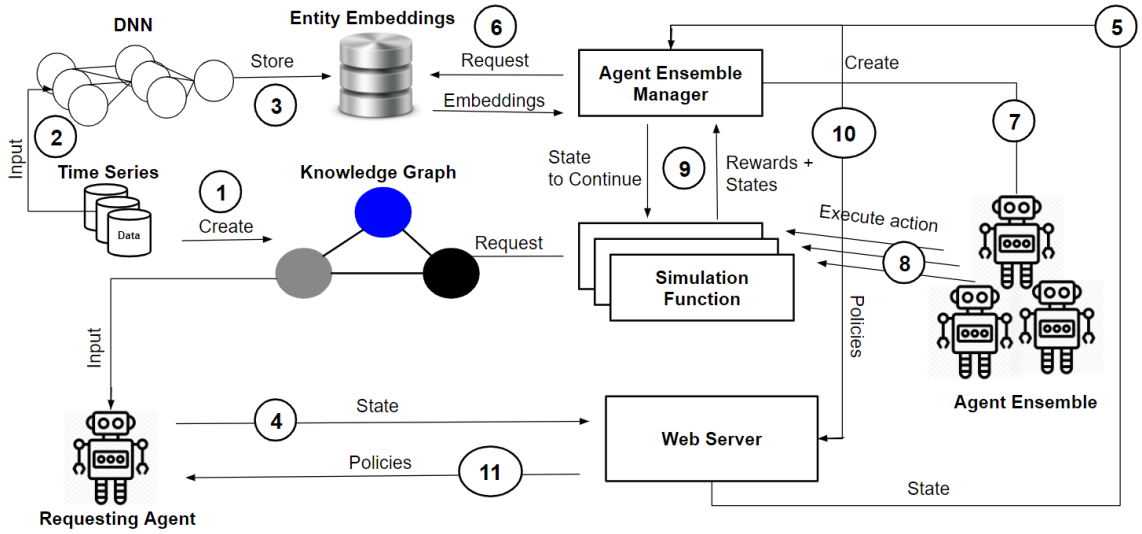


Figure 6.10.: Evaluation set-up of the agent ensemble approach [152].

In order to obtain representative samples that offer activities of varying complexity, it was required to categorise the data samples (i.e. the activities) according to the length of their action sequences and randomly select a certain number of samples from each category. The reason for this categorisation is that the complexity and difficulty of an activity increases as the length of the action sequence increases. Since there were a total of 52 different action sequence lengths and thus categories, one activity from each sequence length category was selected, resulting in a sample size of 52 activities that were assessed. The action sequence lengths of an activity can range from a minimum of 2 actions to a maximum of 80 actions (see Fig. 6.11).

To compare reinforcement learning (RL) with the proposed agent ensemble approach in terms of policy delivery speed, a JavaScript library[9] that implements the Deep-Q-Neural-Network (DQNN) algorithm [170] and trains it for each previously selected activity was used. In parallel, the proposed approach combined policies based on the trained entity embeddings and the feedback of the simulation component for the same activities. For both approaches the required number of *episodes, execution steps* and the *number of incorrectly*

[9] `https://github.com/karpathy/reinforcejs`

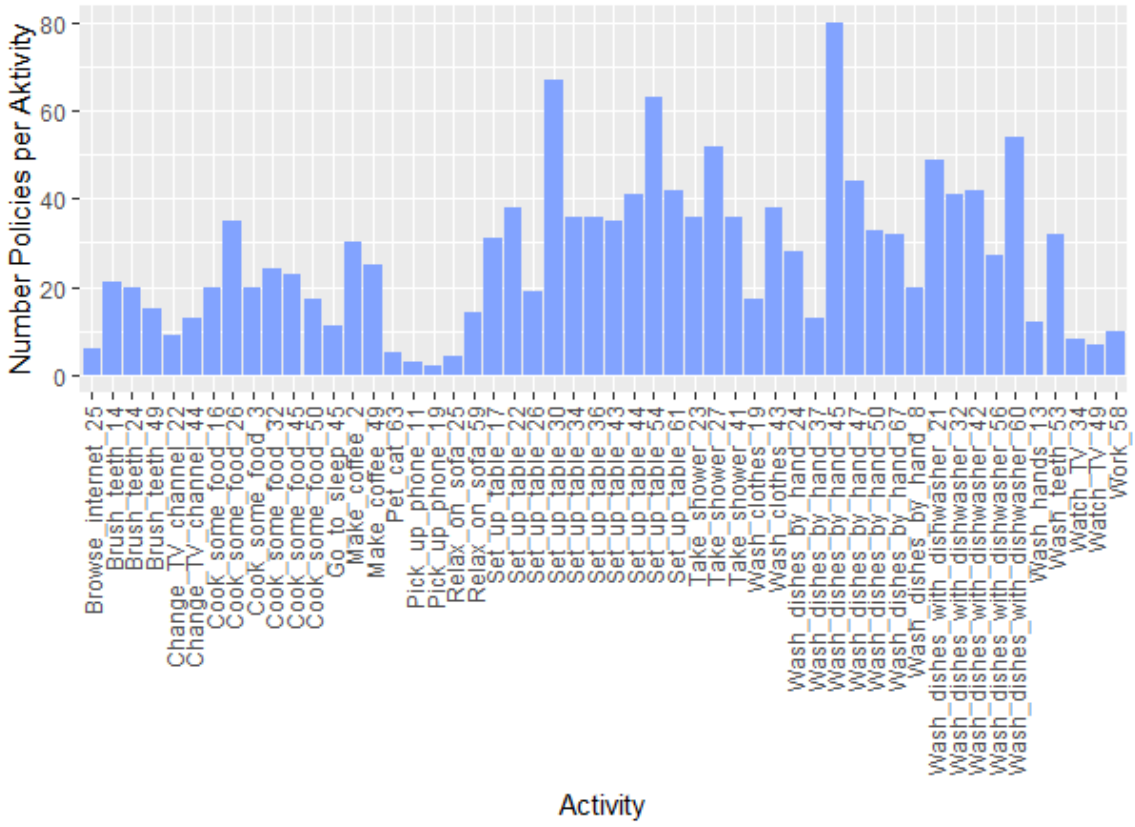


Figure 6.11.: Distribution of the action sequence length (y-axis) among all 52 activities (x-axis) evaluated [152].

executed actions where the reward is < 0 were monitored.

The DQNN was initialised with the following hyper-parameters: **learning rate = 0.05**, **greedy[10] value = 0.9** and **discount factor = 0.9**. To determine whether the corresponding policies were successfully learned by the DQNN, the greedy parameter was set to 0 after each training episode to avoid the agent performing random actions, and then the task was executed again with the trained policies. Then the number of executions performed within an episode and the actual sequence length of the current activity were compared, and if both were equal, the policy training of the DQNN was considered successfully completed.

The maximum number of episodes within which the DQNN had to train policies was limited to 1, 10 and 100 episodes, because without episode limitation it would have taken several days to successfully train the policies for all 52 activities with the DQNN. In addition, the approach was intended to be able to address contextual policy requests immediately or quickly, and therefore the DQNN used for comparison with the proposed approach also had to meet the requirement of delivering policies quickly. For this reason, the DQNN algorithm was challenged to learn policies for each activity within 1, 10 and 100 episodes. Otherwise, if the number of episodes was exceeded, the training for the corresponding activity was terminated and considered incomplete.

The proposed algorithm has one hyper-parameter that represents the search radius, i.e. *cosine distance*, inside which the nearest actions in the embedding space are searched (see Algorithm 9). In the evaluation, the radius parameter was initially set to **0.25** in order to keep the search radius small. Fig. 6.12 shows the obtained distribution of radius values over all evaluated activities.

It is evident that the required mean search radius value lies between a range of 0.6 and 0.8 what suggests that this is the best, i.e. most successful, parametrisation for finding suitable and reward maximising actions. Furthermore, it can be assumed that the mean search radius of about 0.725 is sufficient to narrow down the surrounding action search space of a state, since in most states at least one action found within the radius has proven to be reward-maximising and conducive to activity completion.

The framework's simulation platform used for providing feedback (i.e. state-updates, rewards), allocates rewards that increase by a value of 0.25[11] for each action correctly performed within the sequence. This incremental reward allocation is necessary because the algorithm can repeat found actions infinitely often and thus get stuck in cycles if states within a sequence are not distinguishable by their increasing and thus different rewards.

[10] Specifies the rate of randomly executed actions and thus controls the exploration and exploitation of the policies.

[11] The value 0.25 is an arbitrarily chosen number. It could also have been another number by which the reward value is increased. The important aspect here is that the reward value needs to increase in comparison to the previous state.

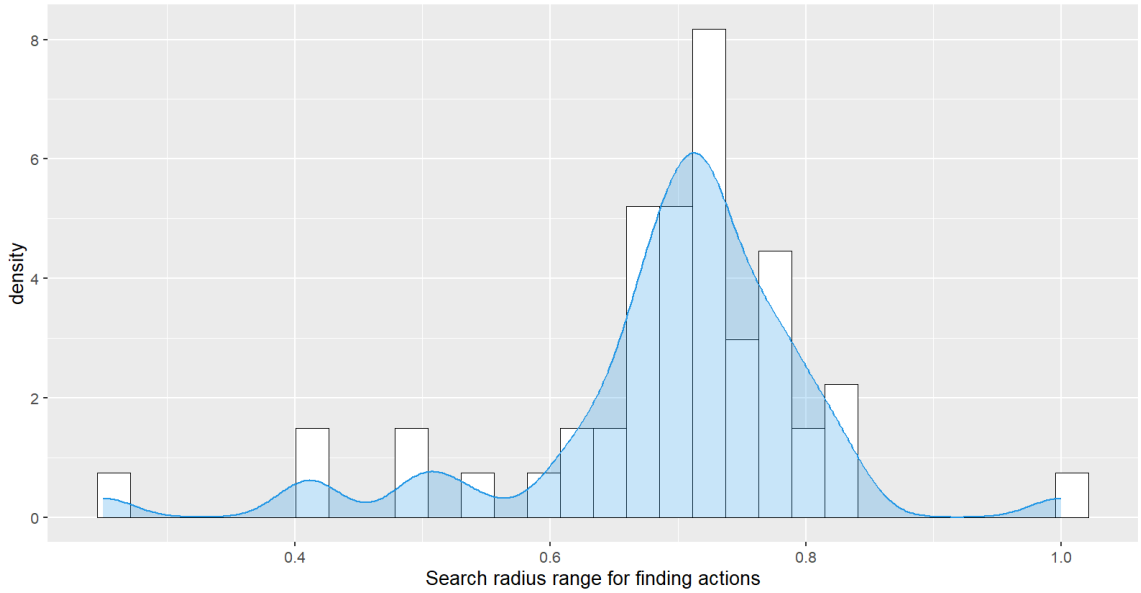


Figure 6.12.: Density plot and histogram of the search radius value ranges (x-axis) in which the most suitable and related action embeddings for submitted, i.e. observed states, were found [152].

In order to evaluate the *generalisability* of the proposed policy combination approach for unknown tasks, a second evaluation run was conducted with a *10-fold cross-validation* [213] in which the *Virtual Home* dataset was shuffled for randomisation and divided into 10 equally sized folds. In each iteration, nine of the 10 folds served as training data, while one fold was omitted from training and served as an evaluation dataset. Since a total of 10 folds were evaluated, 10 different embedding models were trained to evaluate each left-out fold of the data. The performance indicators of the trained models that were evaluated were the *number of steps* required to perform a task, the *number of wrong decisions* made during task execution, the *ratio of successfully completed tasks*, and the *cumulative reward* obtained during task execution. For each measured value of the evaluated folds, the *mean, standard deviation* and *standard error* were calculated, see Table 6.6. In order not to let the tasks run endlessly, the execution of a task was stopped as soon as the search radius, i.e. the *cosine distance*, for the search of actions was greater than or equal to 2, because with a larger search radius it can be assumed that no (close) relationship or semantic similarity to the neighbouring entities is given. This means that no alternative or semantically similar actions to the current state can be found. The interrupted tasks therefore count as unsolved tasks on which the ensemble approach failed.

6.2.4. Results and Discussion

This section discusses the results of two questions. Section 6.2.4.1 deals with the adaptation of ambiguous policies. The evaluation results are intended to answer RQ2.1 and evaluate hypothesis H2.1. Section 6.2.4.2 discusses the results regarding contextual policy combination. The evaluation results are intended to answer RQ2.2 and evaluate hypothesis

H2.2.

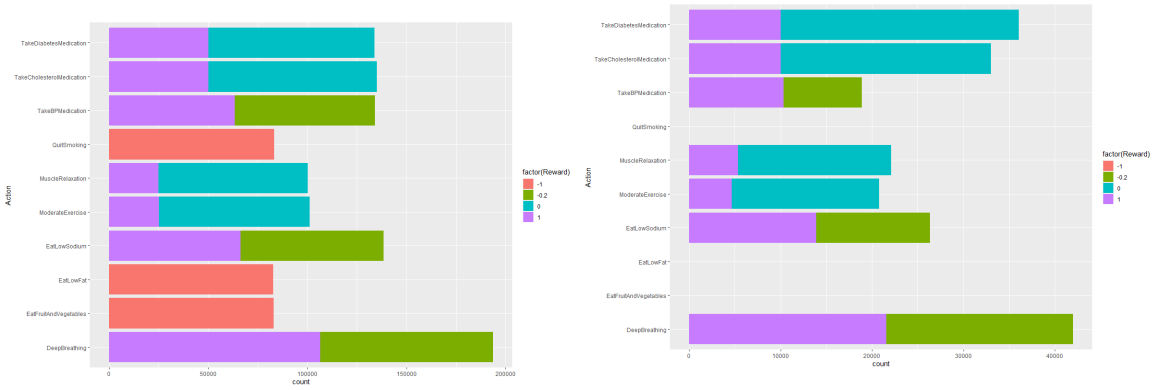
6.2.4.1. Results of Policies Adaptation

As depicted in Fig. 6.13, an agent can perform for the heart disease task 10 different actions that address different disease and patient states.

First, *Agent1* learns default policies for a default task representation with all possible actions. Afterwards, *Agent2* reuses the default policies to adjust them to a reduced task specification that does not require all given actions. In order to achieve this, the reduced task representation does not include all states.

Agent2 performs all actions by means of the provided default policies and is punished by the agent framework for 3 of the actions (i.e. *QuitSmoking, EatLowFat, EatFruitAndVegetables*). Figure 6.13a shows that performing the 3 mentioned actions leads constantly to a punishment by the agent framework with a reward of -1. Consequently, *Agent2* learns to avoid these 3 actions corresponding to the omitted states.

In the second run (see Fig. 6.13b), *Agent2* exploits its adjusted policies by setting its *epsilon* value to zero. The results indicate that it completely omits to perform the 3 considered actions that previously led only to punishments. This observation indicates that RL agents can adjust reused and ambiguous policies through the simulation component of the agent framework that punishes or reinforces actions, depending on the task description. The observed result suggests that H2.1 might apply. To contradict hypothesis H2.1, evidence would have to be provided to demonstrate that the RL agent does not learn new, i.e. required, actions or unlearn redundant, i.e. no longer required, actions due to changes in task descriptions and feedback from the simulation component.



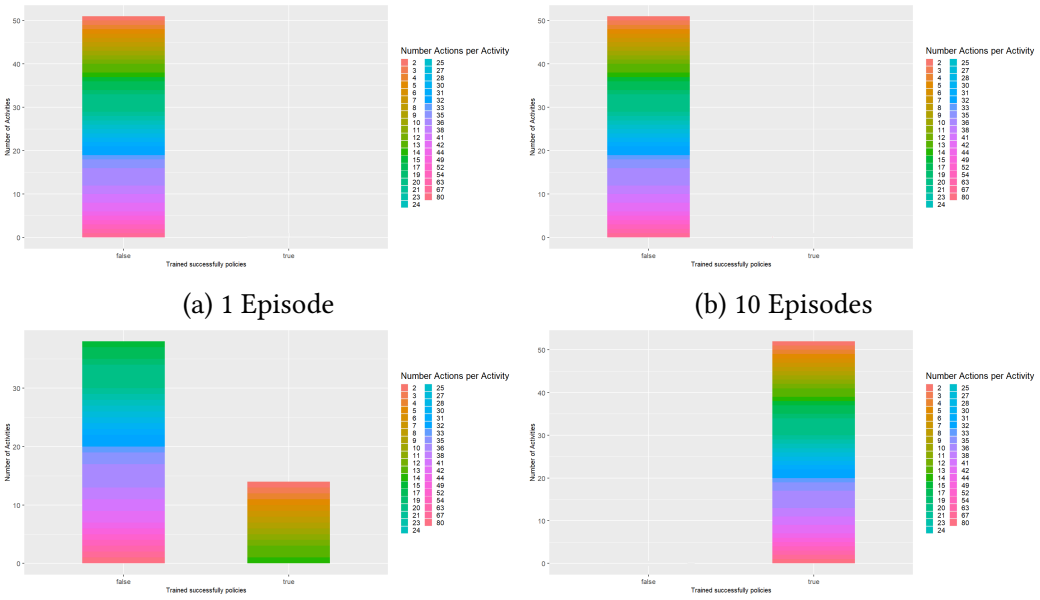
(a) Reinforced actions during training.

(b) Learned actions after training, i.e. adaptation of policies.

Figure 6.13.: Occurrence of actions and assigned rewards during and after training.

6.2.4.2. Results of Context-Aware Policies Combination

EQ2.2.1: Activity Completion The evaluation results suggest that the proposed approach was able to successfully generate policies for all 52 activities within one episode each, while the DQNN agent needed at least 100 episodes to learn policies for 14 out of 52 activities (see Fig. 6.14 and **metric M2.2.3**). For the remaining 38 activities, the DQNN agent would have needed many more episodes before learning reward-maximising and correct sequential behaviour, because as the action sequence length increases, the state space and action choices also increase, making it more difficult and time-consuming for the agent to learn correct action sequences. Fig. 6.14 illustrates that the DQNN agent could not successfully learn any of the activities within 1 and 10 episodes, while after 100 episodes it had learned at least 14 activities with minimum and maximum action sequence lengths of 2 and 14 (see **metric M2.2.2**). For activities with a sequence length greater than 14, the DQNN agent failed to learn the correct action sequence within the 100 episodes (see **metrics M2.2.2** and **M2.2.3**).



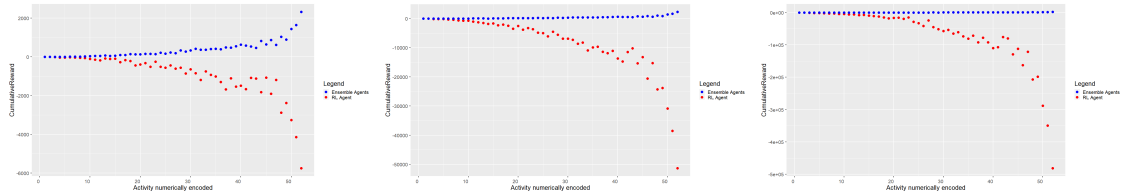
(a) 1 Episode (b) 10 Episodes

(c) 100 Episodes (d) Ensemble agent approach with 1 Episode

Figure 6.14.: Distribution of successfully trained (right bar) and unsuccessfully learned (left bar) activities by the ensemble agent approach after 1 episode and the RL, i.e. DQNN, agent after 1, 10 and 100 training episodes. The colour shades indicate the action sequence length of the respective activities. In contrast to the RL agent, every task performed could be successfully completed by the proposed agent ensemble approach [152].

Fig. 6.15 shows for the proposed approach (blue dots) and for the DQNN agent (red dots) the cumulative rewards (see **metric M2.2.4**) obtained during policy training and combination. The scatter plots illustrate that the proposed ensemble agent approach shows reward maximising behaviour and thus composes the right policies, while the DQNN agent tends to show a reward minimising behaviour. This is reasonable because the principle

of reinforcement learning is to learn policies by *trial and error* and therefore the DQNN agent performs a certain number of exploration steps depending on the greedy value set, which can lead to wrong decisions and negative rewards, and since the DQNN agent needs many more episodes than the ensemble agents to learn policies, the penalties add up (see **metric M2.2.1**).



(a) 1 Episode (b) 10 Episodes (c) 100 Episodes

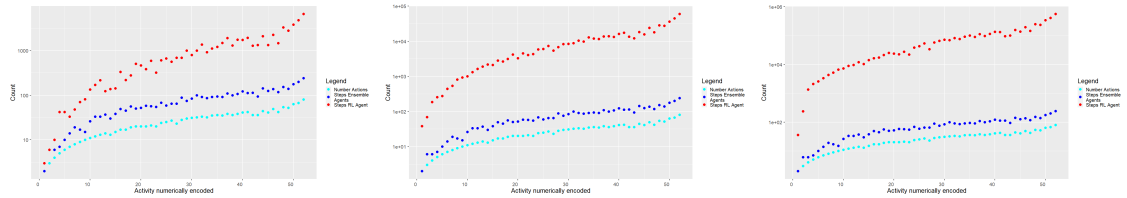
Figure 6.15.: Cumulative reward (y-axis) for each activity (x-axis) obtained during combination by ensemble agents (blue dots) and during training by RL agent (red dots) after 1, 10 and 100 Episodes [152].

As described before, 52 different activities with different action sequence lengths and thus difficulty levels were evaluated and for all 52 activities the presented approach was able to compose VH dataset, i.e. ground-truth, compliant policies. Therefore, it can be concluded for **EQ2.2.1** that the proposed *policies combination approach* is able to compose reward-maximising policies across contexts, i.e. across activities, that contribute to the fulfilment of activities without having to train policies in advance.

EQ2.2.2: Velocity of Policies Provision To evaluate whether the proposed policy combination approach outperforms the DQNN agent in terms of velocity, the steps executed were counted until correct, i.e. activity description compliant, policies could be provided for each activity (see **metric M.2.2.1**). Fig. 6.16 depicts that the presented policies combination approach needs significantly fewer steps than the DQNN agent to provide correct policies for each tested activity.

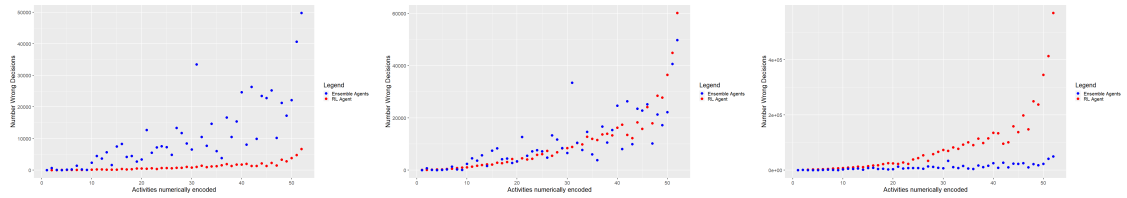
The speed required for policy combination can also be measured by the number of wrong decisions an agent makes, because the more wrong decisions an agent makes, the more steps it has to go through until it finds the right action sequence. For this reason, the number of wrong, i.e. reward-minimising, actions performed by both approaches for each evaluated activity was also counted to show how error-prone and slow the compared agents are when composing or training policies, (see Table 6.4 and **metric M2.2.5**).

Figure 6.17a and Table 6.4 show that the ensemble agents perform significantly more wrong actions (see **metric M2.2.5**). However, Figure 6.17b and 6.17c illustrate that the more episodes the RL agent has to perform, the more wrong actions it shows. Furthermore, it has to be taken into account that the ensemble agents considered all potentially possible actions of all activities in the VH dataset, whereas the RL agent only knew the possible actions of the corresponding activity. Thus, the ensemble agents could also make



(a) 1 Episode (b) 10 Episodes (c) 100 Episodes

Figure 6.16.: Required steps (y-axis) of ensemble agents (blue dots) and RL agent (red dots) per activity (x-axis) until policies are composed or trained. The turquoise dots show the number of required policies among each activity. The plots are scaled and displayed in logarithmic scale to make overlapping values visible in the plots [152].



(a) 1 Episode (b) 10 Episodes (c) 100 Episodes

Figure 6.17.: Number of incorrect decisions (y-axis) made for each activity (x-axis) during a) policy combination by ensemble agents (blue dots), and b) policy training by the RL agent (red dots) [152].

Approach	1 Episode	10 Episodes	100 Episodes
RL Agent	1079.615	10203.92	84521.58
Agent Ensemble	10690.48	-	-

Table 6.4.: Average number of incorrect actions performed for each activity within 1, 10 and 100 episodes (see **metric M.2.2.5**). A total of 52 activities were evaluated for each of the approaches considered. The agent ensemble approach required only 1 episode for each task, thus values for 10 and 100 episodes are not given.

significantly more mistakes, as they could choose actions from a much larger action space encompassing all activities. In addition, the simulation platform also penalised actions of the ensemble agents that were not listed in the current activity description, although they had the same purpose as the intended actions. Furthermore, the RL agent did not provide reliable policies after 1 training episode. As explained earlier, the RL agent required at least 100 episodes to provide correct policies for at least 14 activities with a maximum action sequence length of 14, while the proposed agent ensemble approach could provide correct policies for all tested 52 activities within only 1 episode.

Considering the evaluation results of both approaches, it can be concluded for **EQ2.2.2** that the proposed approach is indeed able to speed up policy delivery in terms of fewer training episodes and steps compared to agents using reinforcement learning (i.e. DQNN).

EQ2.2.3: Generalisability of the Policies Combination Approach The 10-fold-cross validation shows that the policy combination approach is generalisable to a certain extent. On average, **73.42%** of all evaluated tasks could be solved with the approach (see **metric M2.2.3**), while the **standard error** is **0.058378**. In the evaluation on the generalisability of the approach, not all states were known, i.e. were given in the vocabulary, so it becomes explainable why a certain part of the tasks could not be solved. Since a strict sequential processing of the states would have been necessary to successfully complete an activity, even just one unknown state could lead to a task not being solved. A higher success rate of even 100% could be achieved if all possible states, i.e. also the states from unknown tasks or activities, are known in advance in the trained vocabulary and thus embedding model. In such cases, alternative actions or strategies leading to the corresponding states could presumably be found and assembled.

The **average cumulative reward** (see **metric M2.2.4**) achieved is **70.74531** for all successfully completed tasks, with the approach taking an average of **102.2408** steps (see **metric M2.2.1**) to combine policies per task, and a total of **74175.76 false decisions** (see **metric M2.2.5**) being made in the ensemble until policies can be assembled. The evaluation results show that the performance of the policy combination approach tends to reproduce the successful results of EQ2.2.1, even if the embedding models were not trained on all datasets. Figure 6.18 - Figure 6.21 show that the agent using the policy combination approach significantly outperforms the RL agent by requiring on average significantly fewer execution steps (see **metric M.2.2.1**) to solve tasks and is therefore faster and can solve significantly more tasks within one episode (see **metric M2.2.2**). For instance, the bar charts in Figure 6.18 illustrate that the RL agent requires at least 100 episodes in order to complete around 50% of the tasks and have an almost comparable success rate to the policy combination agent, while the policy combination agent is able to solve the majority, i.e. on average 73.42%, of the evaluated tasks in 1 episode (**metric M2.2.2**). However, Fig. 6.21 depicts that the policy combination approach causes more incorrect execution steps (see **metric M2.2.5**) than the RL approach in the first run with one episode, but later with 10 and 100 episodes the RL agent performs significantly more incorrect actions than the agent using the proposed approach. This makes sense, as the framework's policies combination

approach only needs one episode and has a limited search area where it has to seek for semantically related actions to solve appropriate tasks, whereas the RL agent performs more incorrect actions, which result in a negative reward value, to explore strategies with more training episodes. Thus, it can be assumed that the proposed approach generalises to unseen tasks, provided that all states and actions are known in advance and are encoded in the embedding model. For this reason, a knowledge graph (KG) is necessary that provides all possible state and action entities to ensure that newly deployed tasks consisting of an unknown sequence of policies can also be solved using the policy combination approach.

It may sound like a contradiction that despite the generalisability of the approach, all states and actions have to be known in advance. However, the algorithm does not know at runtime in which order and combination of actions the activities should be executed by an agent. The proposed approach computes context-dependent, i.e. state-dependent, the appropriate sequence of actions that leads to the execution of an activity matching the context.

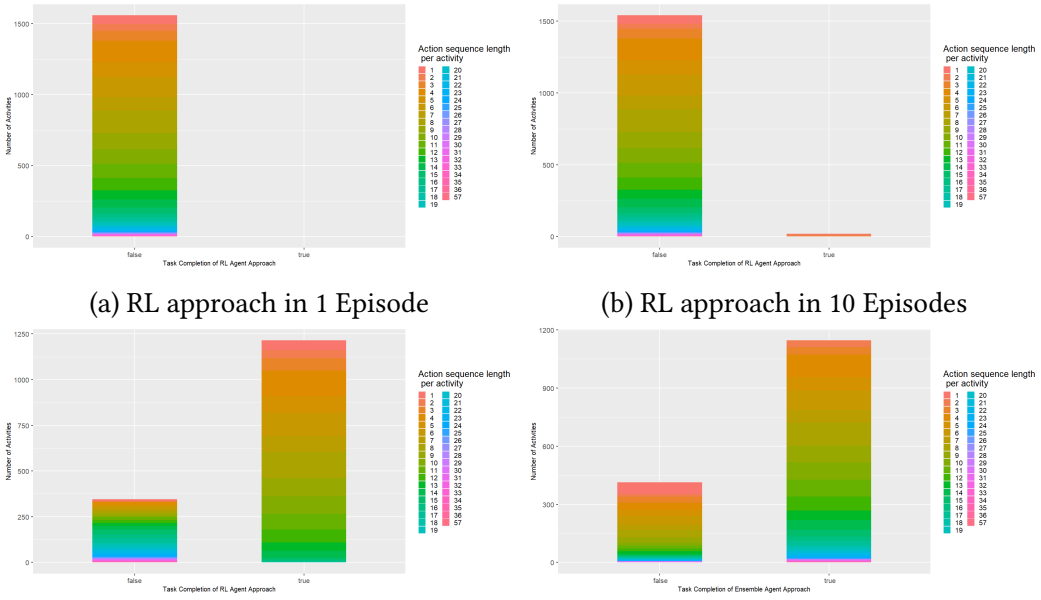
Fold/Model	Mean Steps	Standard Deviation Steps	Mean Wrong Decisions	Standard Deviation Wrong Decisions	Mean Cumulative Rewards	Standard Deviation Cumulative Rewards	Ratio of Successful Completed Tasks
1	48.30128	31.79215	6749.506	9202.326	72.76122	96.16414	0.75
2	118.2692	86.36616	96655.24	69026.88	81.89423	140.3923	0.7628
3	122.8974	93.64512	100309.7	72714.3	88.86218	169.9565	0.7179
4	52.64103	35.07043	5750.878	8079.803	82.35577	109.651	0.8333
5	113.8013	79.68733	95561.93	62987.27	68.69231	108.4526	0.7307
6	107.7821	77.4701	87136.28	58276.37	67.0625	114.2231	0.6859
7	124.3205	81.52498	101042.5	61886.1	79.35256	119.4853	0.6795
8	106.0128	82.70882	89122.44	67571.99	65.14423	127.8612	0.7756
9	73.95513	61.45771	30134.42	20998.12	46.11859	138.7159	0.7756
10	115.5962	77.63278	94667.5	61356.65	69.19391	109.2988	0.6306
Total	98.35769	Standard Error: 28.95776	70713.04	Standard Error: 39758.7	72.14375	Standard Error: 12.03673	Mean: 0.7342 Standard Error: 0.058378

Table 6.5.: Evaluation results of the 10-fold cross validation for the proposed ensemble approach. Here the results of all tasks were considered, i.e. also the ones that were aborted after a certain time because no solution could be found within a predefined radius.

Fold/Model	Mean Steps	Standard Deviation Steps	Mean Wrong Decisions	Standard Deviation Wrong Decisions	Mean Cumulative Rewards	Standard Deviation Cumulative Rewards	Ratio of Successful Completed Tasks
1	45.47863	32.16943	2080.957	3594.237	70.44231	96.46974	0.75
2	123.8487	83.96434	101512.2	67464.97	82.02941	132.7949	0.7628
3	121.0268	87.37487	99281.79	69028.09	75.08929	144.0891	0.7179
4	53.18462	35.1649	2910.638	4000.664	86.575	110.9935	0.8333
5	122.4211	79.91233	103169.6	63230.7	71.36184	111.7522	0.7307
6	117.3084	77.95408	96250.3	59737.22	69.63084	112.6236	0.6859
7	129.9717	84.65127	108104	64376.4	78.47642	127.7595	0.6795
8	115.8512	87.60334	98003.36	72001.17	73.13017	142.1648	0.7756
9	68.07438	43.43466	26076.76	16718.89	29.46281	37.1089	0.7756
10	125.2424	78.31453	104367.9	62006.22	71.25505	109.7531	0.6306
Total	102.2408	Standard Error: 32.88568	74175.76	Standard Error: 44629.17	70.74531	Standard Error: 15.52223	Mean: 0.7342 Standard Error: 0.058378

Table 6.6.: Evaluation results of the 10-fold cross validation for the proposed ensemble approach. Here only the results of completed tasks were considered.

EQ2.2.4: Comparison to Baseline Approaches In Sec. 3.2.2, related works [202, 138, 136] were discussed that propose to use language models to combine policies for *Virtual Home*



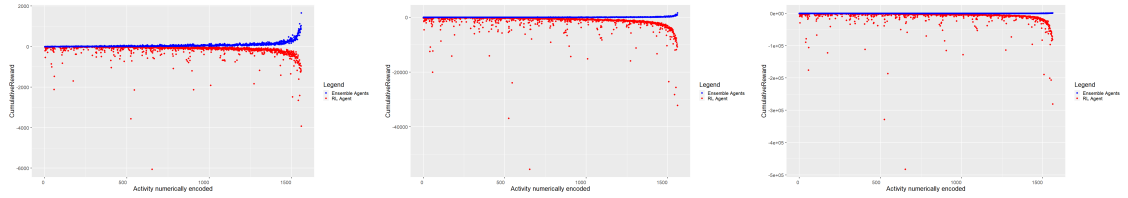
(a) RL approach in 1 Episode

(b) RL approach in 10 Episodes

(c) RL approach in 100 Episodes

(d) Ensemble approach in 1 episode.

Figure 6.18.: Distribution of successfully finished (true value at x-axis, right bar) and unfinished (false value at x-axis, left bar) tasks by the RL and ensemble approach. The RL agent had 3 different training times, i.e. 1, 10 and 100 episodes while the proposed ensemble approach required only 1 episode.

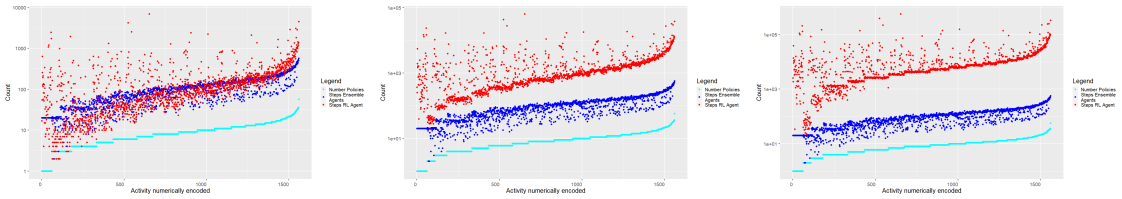


(a) 1 Episode

(b) 10 Episodes.

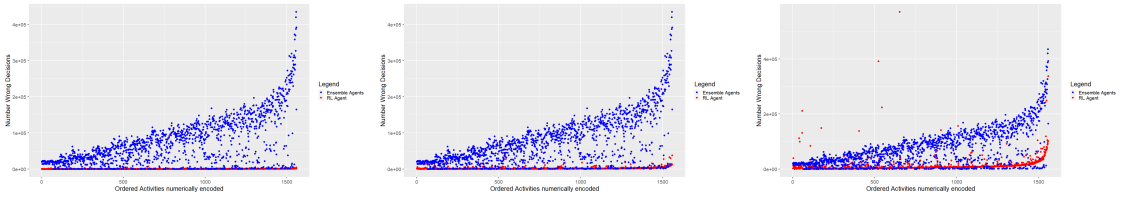
(c) 100 Episodes.

Figure 6.19.: Development of cumulative reward (y-axis) for each performed and numerically encoded activity (x-axis) during training for the ensemble approach (blue dots) in 1 episode and the RL approach (red dots) in 1, 10 and 100 episodes.



(a) 1 Episode (b) 10 Episodes. (c) 100 Episodes.

Figure 6.20.: Required steps (y-axis) for each performed and numerically encoded activity (x-axis) during training for the ensemble approach (blue dots) in 1 episode and the RL approach (red dots) in 1, 10 and 100 episodes. The turquoise dots show how many actions an activity requires in order to be solved. The plots are scaled and displayed in logarithmic scale to make overlapping values visible in the plots.



(a) 1 Episode (b) 10 Episodes. (c) 100 Episodes.

Figure 6.21.: Number of wrongly executed actions (y-axis) for each performed and numerically encoded activity (x-axis) during training for the ensemble approach (blue dots) in 1 episode and the RL approach (red dots) in 1, 10 and 100 episodes.

activities. These related works serve as baseline for evaluating the activity completion rate **(see metric M2.2.3)** in seen and unseen *Virtual Home* activities and comparing the performance in this regard with the above baseline approaches. Table 6.7 shows the results extracted from the related works and compared with the obtained evaluation result. The results suggest that the proposed approach to combine policies gives superior results for both seen and unseen *Virtual Home* activities. However, it is worth mentioning that the evaluation set-ups and goals differ to some extent, and the considered approaches have an additional layer of complexity as they transform natural language descriptions of activities into word embeddings using RNNs, LSTMs, and Transformer architectures before using reinforcement learning to combine policies. In contrast, the proposed approach uses semantic task entity representations and trains from these entity embeddings to search for possible actions to perform. For this reason, the results are only comparable to a certain extent. Nevertheless, the baseline approaches were used for comparison, as it was intended to investigate whether the proposed approach could keep up with related works in terms of policy combination and successful activity execution.

Approach	Executability of novel, unseen activities in (%) i.e. Compliance to Virtual Home dataset	Executability of known, i.e. trained, activities in (%) i.e. Compliance to Virtual Home dataset
ResActGraph [138]	49.3 %	unknown
PG(LCS) [202]	35.5 %	unknown
PG(LCS+Sim) [202]	39.8 %	unknown
LID-Text [136]	58.2 %	87.6 %
LID-ADG [136]	25.5 %	46.7 %
Ensemble Policy Combination (proposed approach)	**73.42 %**	**100 %**

Table 6.7.: Baseline approaches compared to the proposed policy combination approach in terms of their feasibility and conformance to the Virtual Home dataset activities. It is important to note that the experimental set-ups are different. For this reason, the results are only comparable to a certain extent.

6.3. RQ3: Share and Reuse of Policies

At runtime in real operation, agents also produce and observe data that can be used to derive corresponding rules, which in turn can be made available to agents on the World Wide Web (WWW) that have to solve similar tasks. In this way, expert knowledge can be shared, retrieved, reused and adapted as needed, without the requirement of passing long and costly learning procedures. Furthermore, not every agent implements ML algorithms but only applies a rule- and ontology-based reasoning approach. This type of agents needs also to be supported through the exchange of knowledge.

The proposed agent framework provides a methodology to derive rules from the data produced by the agents that can be made available to other agents in a formal representation, via provenance data and task embedding vectors.

Provenance data is seen in this work as meta information that enables other agents to retrace the authorship and application area of provided policies in order to assess their utility and applicability for their own task requirements while task embeddings allow the detection of similar tasks and thus the detection of reusable policies.

As the provision of provenance data is only a question of formal definition, it does not need to be evaluated. Therefore, only the application of derived rules is evaluated together with the retrieval of reusable policies by task embeddings.

Retrospect RQ3:

RQ3 Are rules and policies obtained via mining techniques from the collected data reflecting agent behaviour validly applicable and as accurate and effective as the originally trained behaviour inherent to the data, and do the rules derived from the data actually mirror the behaviour that enables the providing agent to achieve a reward maximising behaviour?

Retrospect H3:

H3 Rules derived and formalised from recorded demonstration data can enable agents in a multi-agent environment to solve given tasks on demand without the requirement and effort of training policies from scratch.

6.3.1. General Goals addressing RQ3

Provenance data mainly contains information about the authorship and extraction of derived rules. On the web, this information is to be made available together with the mined rules and task descriptions in a machine-readable representation, enabling other web agents to reuse the provided expert knowledge, i.e. rules.

In addition, it is assumed in this thesis that similar tasks also require similar policies. Therefore, a methodology is required that allows agents to compare task entities with

respect to their similarity in order to find the corresponding policies. Consequently, if policies can be reused, web agents do not have to go through long learning processes and can directly apply the provided expert knowledge.

In addition, precise and significant rules are also comprehensible to humans, which is conducive to the *explainability* of decisions. In the scope of the evaluation, the following evaluation questions, EQ3.1 - EQ3.2, are posed.

EQ3.1 Are the obtained rules derived from collected data using *frequent item sets* and/or *decision rule mining* techniques validly applicable and as powerful as the originally trained policies of the agent that provided the dataset?

EQ3.2 Is the application or reuse of task policies, which are determined by the similarity of entity embeddings representing semantic task entities, as efficient in terms of reward development, as their use for the originally intended task?

The evaluation hypotheses EH3.1 and EH3.2 summarise the made assumptions and expected outcomes.

EH3.1 Data produced by RL agents in real environments can be exploited respectively mined in order to derive significant rules that allow other (rule-based) agents to solve the corresponding task on demand and without the necessity of going through long-running and costly ML procedures. The assumption is that the found rules are sufficiently meaningful to enable the agent to engage in utility-maximising behaviour as measured by an increase in the obtained positive cumulative reward.

EH3.2 Task representations that have similar or partly identical properties and requirements, can require policies that are mutually usable by agents. In order to determine the similarity of tasks with respect to their properties, task entity embeddings have to be trained that allow the numerical representation of tasks in a n-dimensional vector space. Policies that are found for one task can then be reused for other similar tasks.

6.3.2. Metrics

M3.1 For evaluating the derived rules and retrieval of task policies by task embeddings, the **reward distribution** of a finite set of reward values is considered. This metric is used to assess the coverage of rewards, as it indicates whether all possible rewards could be achieved in the course of each task, because a task can only be completed if the correct sequence of actions and thus the corresponding rewards have been obtained.

$$\mathbf{M3.1} : \mathrm{P(x)} = \sum_{i=1}^{N} \mathrm{r_i},\ \mathrm{r_i} = \begin{cases} 1 & \text{reward value } r_i = \mathrm{x} \\ 0 & \text{otherwise} \end{cases} \tag{6.10}$$

whereby $P(x)$ is the frequency of reward value $x \in \mathbb{X}$ while X is a finite set of reward values, r_i is a constant indicating whether each possible reward value x was (not) obtained and N is the total number of rewards that can be obtained.

M3.2 The **cumulative rewards along episodes and action steps** are considered (see Eq. 6.11).

$$\mathbf{M3.2}: \sum_{t=1}^{T} \mathsf{r_t},\ \mathsf{r_t} \in \mathbb{R} \tag{6.11}$$

whereby t is either an execution step or an episode and r_t is either an obtained reward value for an execution step t or a cumulative reward for an episode t.

M3.3 For the evaluation of the COVID-19 use case (see Sec. 5.3), the **development of the SEIR categories and available hospital beds** among execution steps, i.e. days, are considered. Thus, the evaluation assesses the **number of days until the number of infections stops increasing**, i.e. $f(t) \leq 1$ and the **number of hospital beds starts increasing**, i.e. $g(t) > 1$, (for corresponding equations see Sec. 5.3.2 and Eq. 6.12). The reasoning for this metric is: The more steps, i.e. days, are required to stop the increase in infections and decrease in hospital beds, the worse the actions taken by the agent to contain the COVID-19 pandemic, are.

$$\mathbf{M3.3}: \sum_{t=1}^{T} \mathsf{d_t}\ \mathsf{d_t} = \begin{cases} 1 & 1 < \mathsf{g(t)} \wedge \mathsf{f(t)} \leq 1 \\ 0 & \text{otherwise} \end{cases} \tag{6.12}$$

whereby T is a time variable and represents the total number of considered steps, i.e. days, d_t is a constant with value 1 representing one day, $\mathsf{g(t)} = \frac{\Delta \mathsf{I_t}}{\Delta \mathsf{I_{t-1}}}$ is the *growth factor* of infections and $\mathsf{f(t)} = \frac{\Delta \mathsf{B_t}}{\Delta \mathsf{B_{t-1}}}$ is the *growth factor* of available hospital beds. When the *inflection point* of the *growth factor* is reached at a *growth rate* of 1, it means that the growth is flattening and the *carrying capacity* is saturated, which means that the number of infections stops increasing and, in the case of hospital beds, the number of available beds starts increasing because infected people no longer occupy beds.

M3.4 The **distribution of performed actions** and the rewards obtained for them are considered in Eq. 6.13.

$$\mathbf{M3.4}: \mathsf{P(a)} = \sum_{t=1}^{T} \mathsf{k_t},\ \mathsf{k_t} = \begin{cases} 1 & \text{performed action } a \text{ in time step } t \\ 0 & \text{otherwise} \end{cases} \tag{6.13}$$

whereby T is the total number of execution steps, $P(a)$ is the frequency of action $a \in \mathbb{A}$, $\mathbb{A} = \{\mathsf{a_1}, \mathsf{a_2}, ..., \mathsf{a_n}\}$ is a finite set of actions, k_t is a conditional constant indicating whether action a was (not) executed.

The performance of executing the mined rules (policies) as well as the retrieved similar policies are compared to the performance of executing the policies for the originally intended task. The objective is to evaluate whether by the application of mined rules and similar policies at least the same as or even a higher performance than by the trained policies can be achieved.

6.3.3. Set-up

Two different evaluations were conducted regarding the share and reuse of trained policies within the agent framework. The first evaluation demonstrates how datasets, generated by RL agents during real-time execution, can be mined for deriving significant rules that allow rule-based agents to perform tasks on demand without a performance loss and without the necessity of passing long preprocessing and learning procedures. The second evaluation addresses the utilisation of task embeddings within the agent framework. In particular, the focus is on the retrieval of matching policies, based on the similarity of related task embedding vectors. The objective is to evaluate the feasibility and effectiveness of the presented approach.

The tasks that were evaluated are the web automation (see Sec. 5.1) and the COVID-19 measures in Sec. 5.3. For the web automation task it was an objective to mine rules that adhere the learned order of actions. Therefore, frequent item sets were applied while the adopted and devised algorithms, i.e. *sequential covering, OneR, Best Fit Sequence*, observed and preserved the order of performed actions within the frequent item sets. The most frequent action sequences were then selected by the approach as potential rules. The obtained results that will be discussed in Sec. 6.3.4, show that this approach was successful. It is worth mentioning that the developed *Best Fit Sequence* algorithm was applied to the *Flight Booking* task because the task at hand is divided into stages and the said algorithm addresses tasks with stages and sub-goals.

Furthermore, the public Virtual Home (VH) dataset[12] that consists of instructions (i.e. agent programs) for household agents, was adopted and transferred into semantic task descriptions (JSON-LD) in order to enable the agent framework to process the task specification and demonstrate the ability of the agent framework to find similar tasks and reusable policies. The corresponding dataset has the advantage of providing numerous alternative agent programs for one activity and supporting a broad range of household activities.

It can be assumed that alternative programs that relate to one activity, will have a high similarity in their presentation and application what proves to be useful for the evaluation of the task embeddings approach. Similar programs however can vary regarding the required actions or the order of action sequences, while solving the same household activity. The VH dataset is briefly described in Sec. 6.3.3.2.

[12] `http://virtual-home.org/tools/explore.html`

Since both evaluations have different objectives and requirements, two different set-ups were implemented that are described in the next two subsections.

6.3.3.1. Rule Mining Evaluation

Figure 6.22 shows the structure and processing of the evaluation with regard to the proposed rule mining approaches.

First, a RL agent trains the policies for the appropriate task *Task1* and then executes the task with the trained policies. During task execution, the epsilon parameter of the RL algorithm is set to zero, because the agent shall only execute its learned polices without performing exploration steps, since the made assumption is that the agent has already achieved an optimal performance.

The agent tracks its perceived states and performed actions. After a predefined number of episodes (i.e. 1000), the agent stores a CSV file with the tracked states and performed actions. Second, this CSV file is preprocessed and mined by means of different approaches (i.e. frequent item sets, sequential covering and OneR algorithm) by a *rule mining agent*. The rule mining agent stores for every approach the corresponding rules as a JSON file.

Third, subsequently, a rule-based agent requests and reuses these rules in order to solve the same task and store its performance via a CSV file. By comparing the performance of both agents, the evaluation tests whether the previously posed hypotheses H3.1 and H3.2 are true.

The same evaluation is also conducted with an agent that implements a genetic algorithm (GA), since the proposed framework also supports a GA implementation that can be selected for the configuration of an agent instance. The hyper-parameters of the GA algorithm are the following ones: **population size:** 200, **gene size:** 200, **mutation probabilities:** 0.3, 0.2, 0.1, 0.01, 0.001, **crossover probabilities:** 0.8, 0.75, 0.7, 0.65, 0.6. To terminate and thus detect convergence of the GA algorithm, a maximum number of iterations in which the best fitness value does not change, need to be defined. The maximum number of iterations to detect convergence was set to 200. It is evident that some hyper-parameters were evaluated with a list of different values to test different configurations. Each gene within one chromosome represents a COVID-19 measure per day. The containment measures are optimised for 200 days, since 200 genes build a chromosome in the experiment. The order of genes plays an important role, since the sequence of genes influences the success of the measures to contain the pandemic and keep the economy stable. Hence, in the algorithm's mutation or crossover operations, permutation strategies of genes are performed.

For every task specification, an initial configuration respectively instantiation is required. Exemplary, Table 6.8 and Table 6.9 show the initial configuration of the COVID-19 measures task description. For the evaluation set-up, a population size of 100000 is assumed. Furthermore, the configuration defines that 1000 persons of the population are exposed

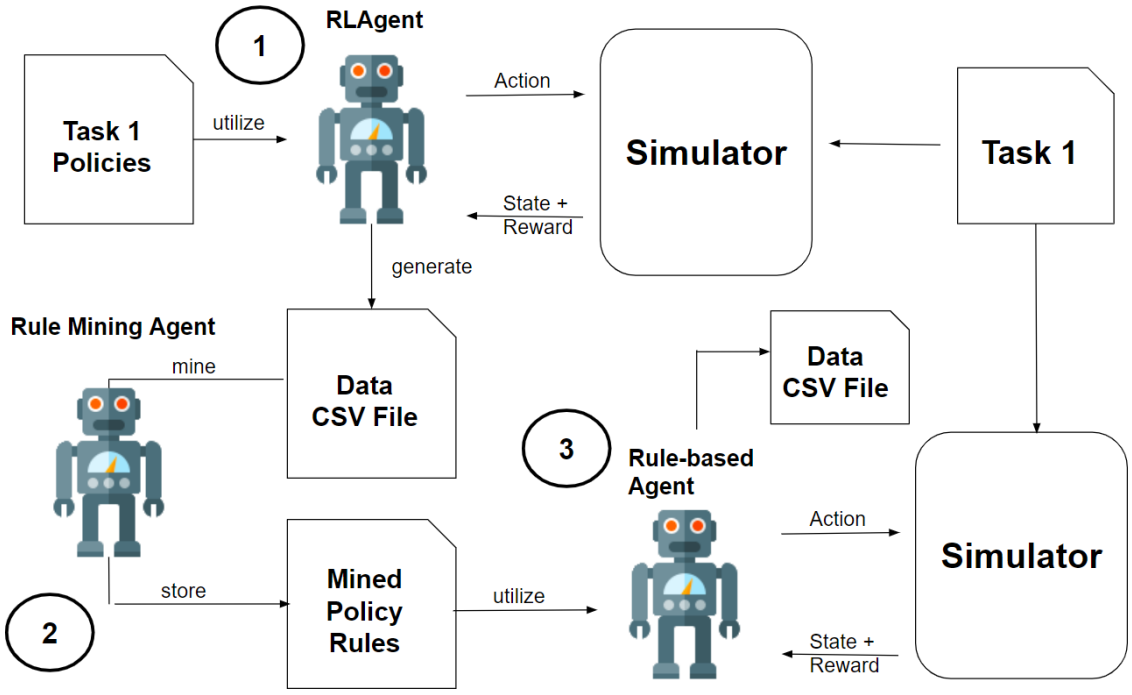


Figure 6.22.: Set-up for rule mining evaluation

and infected, while recovered people are initially set to 0.

Population Size	Susceptible	Exposed	Infected	Recovered	Available Beds
100000	98000	1000	1000	0	250

Table 6.8.: Configuration of initial conditions for the COVID-19 measures task.

As outlined in Chapter 5.3, the SEIR model equations require model parameters that are defined in Table 6.9. The mentioned parameters impact the rate of change and are described in detail, in the referenced chapter about the considered COVID-19 use case.

Recovery Factor	Risk Patient Factor	Avg. Incubation Time (Alpha)	Contact Rate (Beta)	Infection Duration Rate (Gamma)
0.01	0.05	0.1923	0.0000049	0.1724

Table 6.9.: Initial model parameters for the COVID-19 measures task.

6.3.3.2. Task Embeddings Evaluation

In order to evaluate whether policies of similar tasks can be reused, the *Virtual Home (VH)* dataset has been utilised. The VH dataset belongs to an activity knowledge base[13] on the web and provides action sequence specifications (i.e. programs), related to interactive household objects. The dataset consists of instructions that enable software agents to perform different activities within a household. Moreover, the dataset serves for the simulation and visualisation of household activities within the corresponding multi-agent platform[14]. The provided dataset is separated in 12 categories (e.g. leisure, work, social interaction, etc.) and consists of 1563 activities (e.g. *Prepare coffee, Clean room, etc.*), respectively agent programs and 1307 actions.

For the purpose of evaluation, each activity however, had to be transformed into a processable semantic task entity specification, since the agent framework requires these specifications in order to simulate tasks and train agents. An activity, respectively agent program of the VH dataset is structured as in Listing 6.1 and starts with the name of the activity (line 1):

Listing 6.1: A Turn on light activity from the Virtual Home dataset.

```
1 Turn on light
2 Walk into living room. Find wall. Find light button. Switch on light button.
3
4 [Walk] <living_room> (1)
5 [Walk] <light_switch> (1)
6 [Find] <light_switch> (1)
7 [SwitchOn] <light_switch> (1)
```

Subsequently, the activity is described by a short natural language text (line 2). Finally, an ordered sequence of actions, related to interaction objects (e.g. light switch, living_room), follows (lines 4-7). Every household object has an assigned object id number that indicates whether an action should be performed on the same object or on a different object instance.

In this example the activities are performed always on the same object (i.e. light switch), since the ids of the object are identical. More detailed information regarding the dataset can be found at the project's Github[15] page.

Since for evaluation purposes, task representations were generated based on the agent programs of the *Virtual Home* dataset, states and observation features had to be artificially derived that correspond to single actions. The observation features that indicate a state, were defined as categorical features that either take the value 0 for false or 1 for true. The observation feature labels were generated by adding an "*Is*" in front of each action label. For instance, the corresponding feature related to the action *Walk_living_room*

[13] http://www.virtual-home.org/tools/explore.html
[14] http://virtual-home.org
[15] https://github.com/xavierpuigf/virtualhome/blob/master/dataset/README.md

would be transformed into *IsWalk_living_room*. Every related state was created by adding the expression "*_Done*" at the end of every action label. Analogously, according action effects were created by concatenating the word "*Set*" in front of the action label, so that *SetWalk_living_room* is obtained.

Since the VH dataset sometimes contains identical task names, it was required to enumerate tasks that have identical names in order to make the tasks separable and unique. This fact implies that tasks in the dataset can exist in numerous variations so that these tasks are comparable regarding their characteristics (i.e. number of actions, order of actions) and objectives.

Figure 6.23 represents an example MDP of a task entity representation, derived from the task *Turn on light 15* (see Table 6.10). The states of the MPD are derived by the corresponding actions that lead to the states. The edges also represent what reward is given when the agent performs an action in the corresponding state. The state (i.e. *SwitchOn_lightswitch_Done*, following the last action, represents the final target state, since it can be assumed that the last action of the activity leads to a target state, indicating the successful completion of an activity. Therefore a reward of 1.0 is given for the last action of the sequence while the interim actions yield in a reward of 0.5.

It is important to note that the sequence of actions is fixed defined for each agent task and needs to be learned and followed by the respective agent. However, is this not the case, the agent is not able to solve the task successfully.

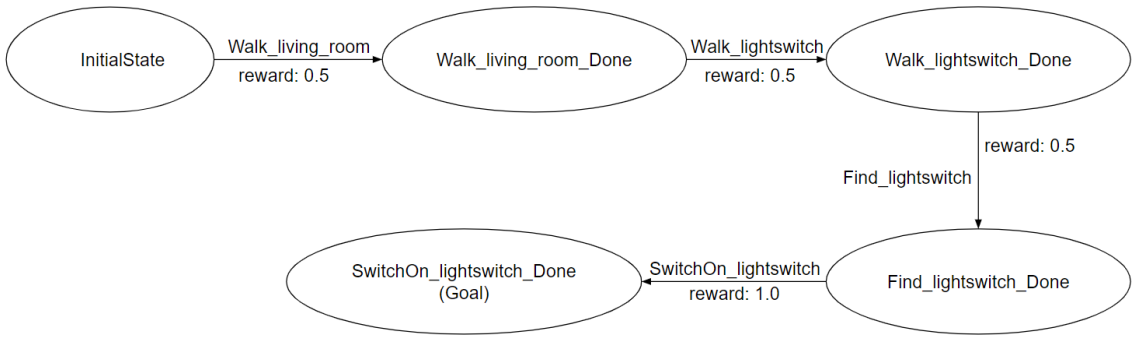


Figure 6.23.: A generated MDP for the *Turn on light 15* task.

The JSON-LD excerpt in Listing 6.2 shows exemplary how for one action (i.e. Walk living room) of the agent program, an appropriate task specification with a state (lines 3-41), an observation feature (lines 43-70), an action (lines 72-93) and action effect (lines 95-110), are derived and represented. For every action the task description was enhanced by these instances (state, observation feature, action, effect). Moreover, an artificial state (i.e.*InitialState*) was specified that represents the initial state of the MDP.

Listing 6.2: An exemplary task specification as JSON-LD serialisation.

```
[
# An exemplary state
{
   "@id": "http://example.org/Entity/Walk_living_room_Done",
   "@type": [
     "http://example.org/Concept/Category-3AState"
   ],
   "http://example.org/Property/Property-3AHasObservationFeature": [
     {
       "@id": "http://example.org/Entity/IsWalk_living_room"
     }
   ],
   "http://example.org/Property/Property-3AHasAction": [
     {
       "@id": "http://example.org/Entity/Walk_living_room"
     }
   ],
   "http://example.org/Property/Property-3AIsGoal": [
     {
       "@type": "http://www.w3.org/2001/XMLSchema#boolean",
       "@value": false
     }
   ],
   "http://example.org/Property/Property-3AIsInitialState": [
     {
       "@type": "http://www.w3.org/2001/XMLSchema#boolean",
       "@value": false
     }
   ],
   "http://example.org/Property/Property-3AHasExpression": [
     {
       "@value": "IsWalk_living_room == 1"
     }
   ],
   "http://example.org/Property/Property-3AHasReward": [
     {
       "@type": "http://www.w3.org/2001/XMLSchema#double",
       "@value": 0.5
     }
   ]
 },
 # The according observation feature
 {
   "@id": "http://example.org/Entity/IsWalk_living_room",
   "@type": [
     "http://example.org/Concept/Category-3AObservationFeature"
```

```
    ],
    "http://example.org/Property/Property-3AHasRangeStart": [
     {
       "@type": "http://www.w3.org/2001/XMLSchema#double",
       "@value": 0
     }
    ],
    "http://example.org/Property/Property-3AHasRangeEnd": [
     {
       "@type": "http://www.w3.org/2001/XMLSchema#double",
       "@value": 1
     }
    ],
    "http://example.org/Property/Property-3AHasFeatureType": [
     {
       "@value": "NOMINAL"
     }
    ],
    "http://example.org/Property/Property-3AHasUnit": [
     {
       "@value": ""
     }
    ]
  },
  # The according action
   {
    "@id": "http://example.org/Entity/Walk_living_room",
    "@type": [
     "http://example.org/Concept/Category-3AAction"
    ],
    "http://example.org/Property/Property-3AHasText": [
     {
       "@value": ""
     }
    ],
    "http://example.org/Property/Property-3AIsNegation": [
     {
       "@type": "http://www.w3.org/2001/XMLSchema#boolean",
       "@value": false
     }
    ],
    "http://example.org/Property/Property-3AHasEffect": [
     {
       "@id": "http://example.org/Entity/SetWalk_living_room"
     }
    ]
  },
```

```
94  # The effect of the according action
95  {
96    "@id": "http://example.org/Entity/SetWalk_living_room",
97    "@type": [
98      "http://example.org/Concept/Category-3AEffect"
99    ],
100   "http://example.org/Property/Property-3AHasImpactType": [
101     {
102       "@value": "ON"
103     }
104   ],
105   "http://example.org/Property/Property-3AHasObservationFeature": [
106     {
107       "@id": "http://example.org/Entity/IsWalk_living_room"
108     }
109   ]
110 }]
```

The implementation of the task embeddings evaluation is performed in 2 subsequent steps. Figure 6.24 illustrates the appropriate set-up.

First, for every given task specification an embedding vector is trained within the framework. Then, an arbitrary number of tasks (see Table 6.10) are randomly selected. Subsequently, for every selected task (i.e. *Task1, Task2*), the framework determines the most similar tasks by computing the cosine distance between the task embedding vectors. The task that is most similar to the randomly selected one is then chosen by the framework to assess the re-usability of task strategies for similar tasks. To do so, the framework generates, as previously described, a task specification for each of the selected tasks, i.e. the randomly selected task and its most similar task, and consolidates both tasks in one specification.

Agent1 trains policies for *Task1* via the agent framework and the appropriate task specification, and *Agent2* adopts and reuses subsequently the policies that were trained by *Agent1* for *Task1* and applies these policies to *Task2*. In this way, the evaluation can demonstrate by assessing the agents' performance (i.e. cumulative reward and order of performed action sequences) that policies that were trained for slightly different tasks, can be reused and adapted for similar tasks successfully.

6.3.4. Results and Discussion

This section presents and discusses the obtained results of the conducted evaluation. First, the evaluation results of the rule mining approach (Sec. 6.3.4.1) and afterwards the results of the task embeddings approach (Sec. 6.3.4.2) are outlined in terms of feasibility and performance.

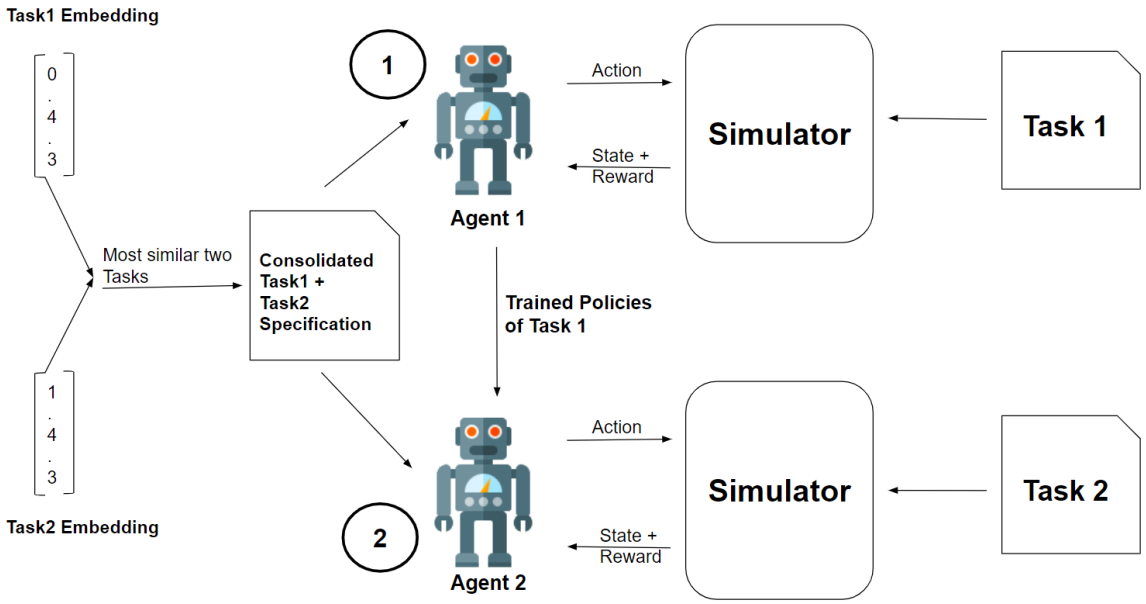


Figure 6.24.: Setup for task embeddings evaluation

6.3.4.1. Results of Rule Mining Evaluation

Web Automation The plots of Figure 6.25 suggest that the derived rules for the web automation tasks, have been successfully reused for performing the corresponding web tasks. The plots for each web task illustrate that, on average, an agent that applied the mined rules, received an almost maximum positive reward for each episode, see also **metric M3.2**.

Only for the flight booking task, which was more difficult and complex to solve, the agent received some negative rewards[16]. However, this is explainable by the fact that searching for destination and departure airports from a long list of airports, took some time, what led to the situation that sometimes the task could not be solved in time (i.e. within 30 seconds). However, this is acceptable, as the aim of this evaluation was not necessarily to complete the task quickly, but to perform it correctly. Furthermore, the agent had no influence on the duration of the airport search since the MiniWob++ benchmark conducted the database search.

The histograms of Figure 6.26 confirm and reproduce as well the results of the scatter plots in Figure 6.25. It turns out that almost all web automation tasks result only in positive rewards while in the flight booking task the number of positive rewards exceeds the number of negative rewards per episode, see **metric M3.2**.

Virtual Home Dataset In order to reproduce the results also for heterogeneous application areas, the VH dataset was evaluated in the same manner as the web automation tasks, because the VH dataset provides also sequential tasks. The only difference between the web tasks and the domestic tasks is that the web tasks additionally include stages that

[16] The rules that were applied for the *Flight Booking* task, were derived by the *Best Fit Sequence* algorithm.

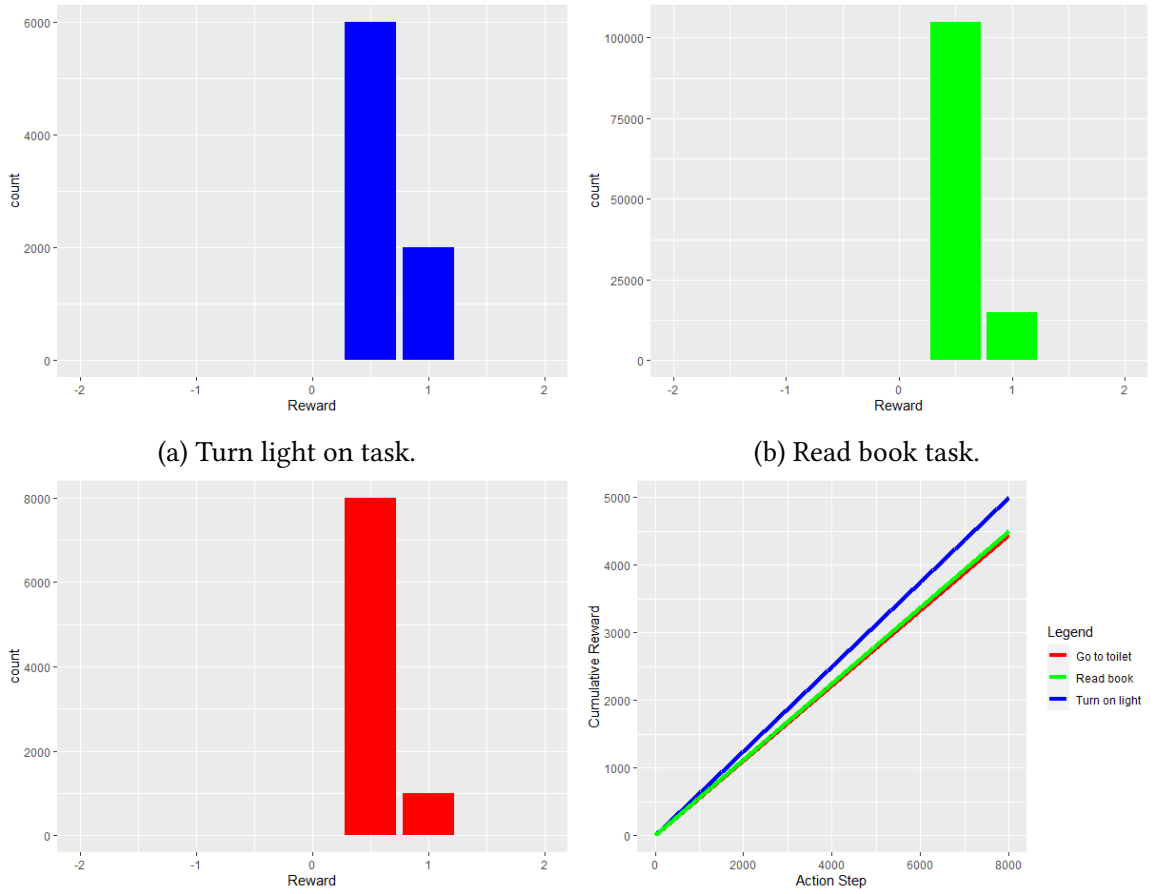
Figure 6.25.: Performance of agents that reused for the web tasks the mined rules.

Figure 6.26.: Histograms of rewards that were achieved by utilising the mined rules.

summarise sequences of actions to fulfil a sub-goal. However, in both tasks, the agent has to comply again with the order of actions.

It is important to note that each action of a VH task, performed in the correct order, is rewarded with a 0.5 reward. Only the final action that leads to the final state results in a reward of 1, which is illustrated by the histograms in Figure 6.27, see also **metric M3.1**.

The histograms imply that the derived rules are consistent with the correct sequence of actions. The cumulative reward also confirms that the performance and compliance of the derived and applied rules is comparable to the original guidelines, see **metric M3.2**.



(a) Turn light on task.

(b) Read book task.

(c) Go to toilet task.

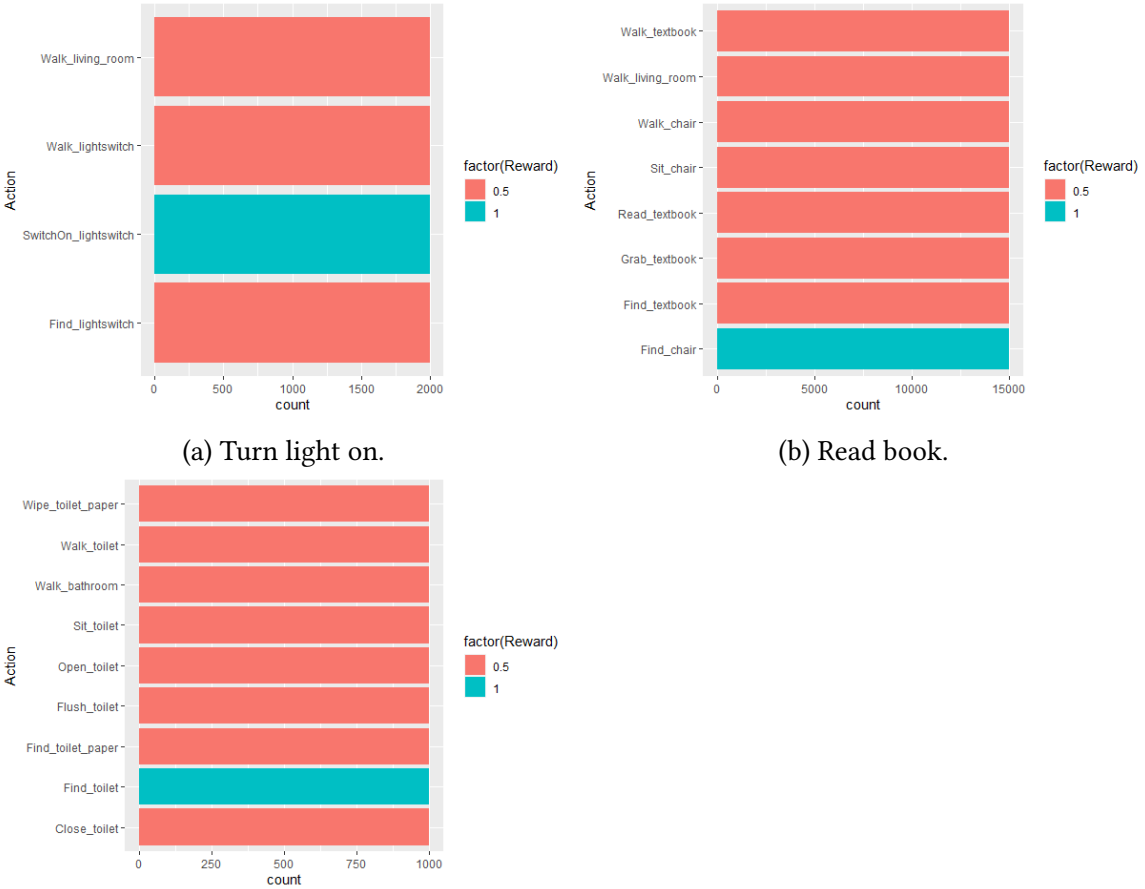
(d) Cumulative Reward development of all 3 tasks.

Figure 6.27.: Virtual Home dataset tasks, executed with mined rules.

The coloured bar charts of Figure 6.28 give insights about the distribution of performed actions during the evaluation run. The different colours (i.e. red, blue) represent the reward values that were assigned for every performed action, see **metrics M3.1** and **M3.4**.

The blue coloured bar charts depict the actions that led to the completion (i.e. target state) of the corresponding task and resulted in a reward value of 1. Since every listed

action has to be performed in order to complete the task at hand, every bar chart exhibits consequently the same frequency of performed actions, see **metric M3.1**.



(a) Turn light on.

(b) Read book.

(c) Go to toilet.

Figure 6.28.: Performed actions and rewards of Virtual Home dataset, executed with mined rules.

COVID-19 Measures The agent framework also supports the processing and simulation of unordered tasks that do not require a fixed order of actions. For this reason, the rule mining approach has to be evaluated in terms of applicability with respect to unordered tasks.

The COVID-19 measures is a use case and task that complies with the requirement of being unordered because the action sequences are not predetermined and each possible action can be executed in an occurring state. However, this also implies that different rule mining algorithms are required than for sequential tasks. In the proposed approach, the algorithms that address rule mining of unordered tasks, are *sequential covering* and *OneR*. Therefore, both algorithms were implemented, executed and compared, in order to

determine differences regarding performance and applicability.

Figure 6.29 depicts the cumulative reward of both algorithms and allows the comparison to the RL algorithm that previously provided the data for mining the rules, see **metric M3.2**. The comparison with the RL algorithm is necessary because it serves as a ground-truth and the aim of the evaluation is to show that the rules obtained do not lead to a loss of performance.

Figure 6.29a illustrates that sequential covering and OneR outperform the RL algorithm in terms of improvement speed. Initially, a descent in reward scores is obtained for all three agents. However, the descending cumulative reward value turns into an increasing cumulative reward value after some time. The RL agent needs 250 time steps[17], i.e. days, for its cumulative reward value to turn to an increase, while sequential covering and OneR only need half the time (i.e. 125 days) to achieve a positive effect, see **metrics M3.2** and **M3.4**.

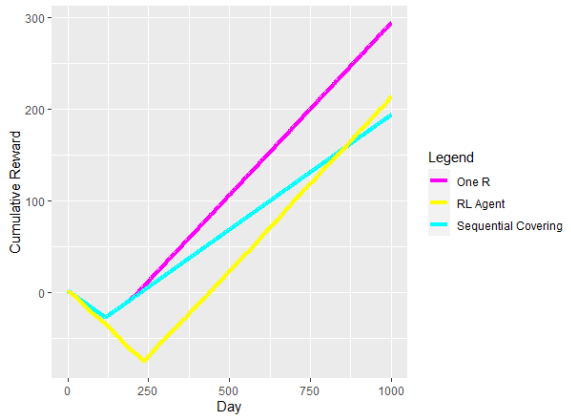
The curve of the RL agent is lower than the curves of the mined rules. However, the OneR algorithm best approximates the behaviour of the RL agent, as both have the same but time-shifted curve shape. The sequential covering curve has a flatter increasing cumulative reward, since its measures are more restrictive and less economy friendly than at the other two agents. The action steps for each evaluation run are restricted to 1000 steps and therefore no conclusion can be made about how the cumulative reward of each agent will evolve after more execution steps. Nevertheless, it is important to note that especially for the COVID-19 measures task, a fast increase of high performing measures is desirable because the faster the course of the disease can be contained, the more risks can be avoided and the more helpful are the imposed measures, see **metric M3.3**.

Considering the results, it can be concluded that mined rules exploit in particular useful actions from the data that is provided by the RL agent. This can be explained by the fact that the approach conducts a data preprocessing step in which data samples with low valued respectively rewarded actions are filtered out. The corresponding histograms as well reproduce the outcome of the cumulative reward plot, see also **metric M3.2**. Furthermore, the rule mining algorithms extract significant patterns from the positively rewarded data samples. These patterns reproduce the successful actions of the RL agent.

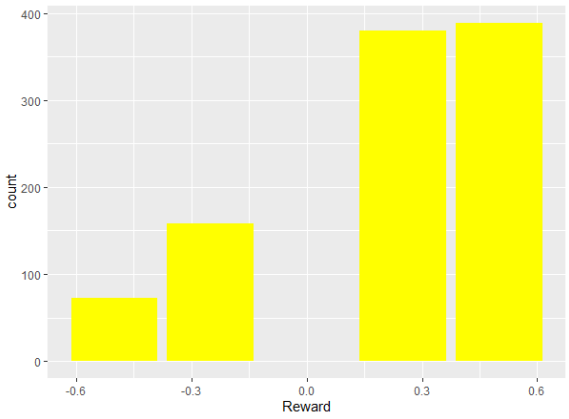
The sequential covering algorithm is less powerful than the OneR algorithm because its cumulative reward curve is flatter and rises more slowly because the OneR algorithm receives higher reward values (e.g. 0.5 and 0.25) for its actions, while the sequential covering algorithm only receives a reward value of 0.25 for its restrictive measure (i.e. TotalConfinement). However, in the first 125 days, both rule mining algorithms are equal, while they outperform the RL agent in this period. This in turn suggests that both rule mining algorithms allow the derivation of significant rules that lead to an increase in the

[17] One time step represents one day

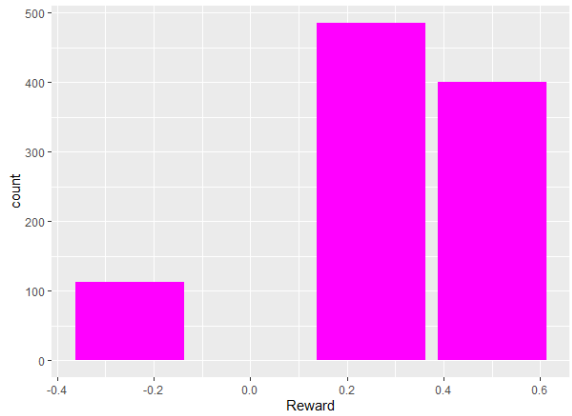
agent's performance.



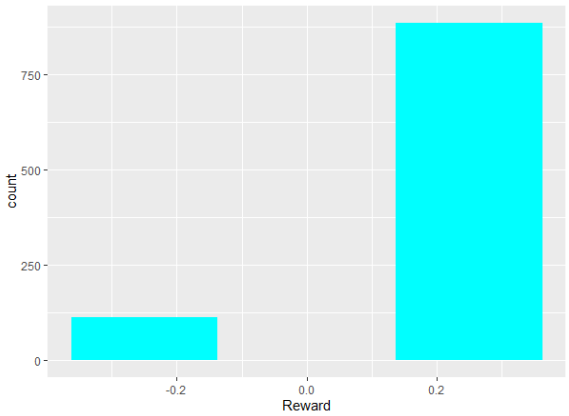
(a) Cumulative reward of sequential covering, OneR and RL algorithm over time, i.e. days.



(b) Reward histogram of RL agent.



(c) Reward histogram of OneR algorithm.

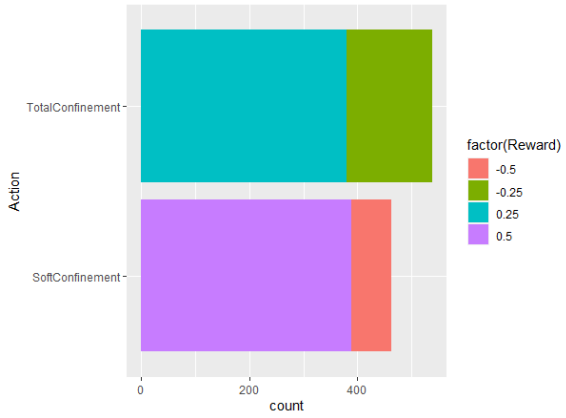


(d) Reward histogram of Sequential Coverage algorithm.

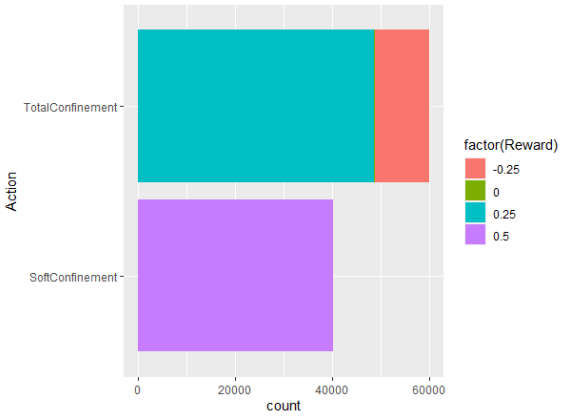
Figure 6.29.: COVID-19 measures: RL agent performance versus mined rules performance.

Figure 6.30 depicts for every evaluated algorithm, the distribution of performed actions, see **metric M3.4**. It might be interesting to see that the number of performed actions differs between the algorithms. For the task concerning COVID-19 measures, it was possible to execute 4 actions, however the RL algorithm and the OneR algorithm performed in summary only 2 of 4 actions (i.e. *TotalConfinement, SoftConfinement*) that were for both identical, while the sequential covering algorithm managed to solve the task with only 1 action (i.e. *TotalConfinement*). The actions *SomeRestriction, NoRestriction* were omitted by all considered algorithms. This seems to make sense, because the trade-off between public health and economy had to be addressed by the simulation model. While one objective was to preserve the public health in a large part by keeping the number of infections low and the number of hospital beds sufficient, the other objective was to keep the economy up and stable. The reward assignment configuration in Table 5.8 tried to address both

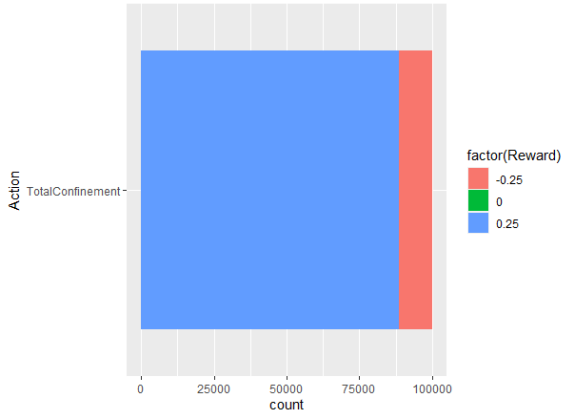
objectives. However, only the plots of Figure 6.31 provide insights about whether the two diverging objectives could be achieved effectively by the performed measures.



(a) RL agent.



(b) OneR algorithm.



(c) Sequential Covering algorithm.

Figure 6.30.: COVID-19 measures: Performed actions of each algorithm and received rewards per action.

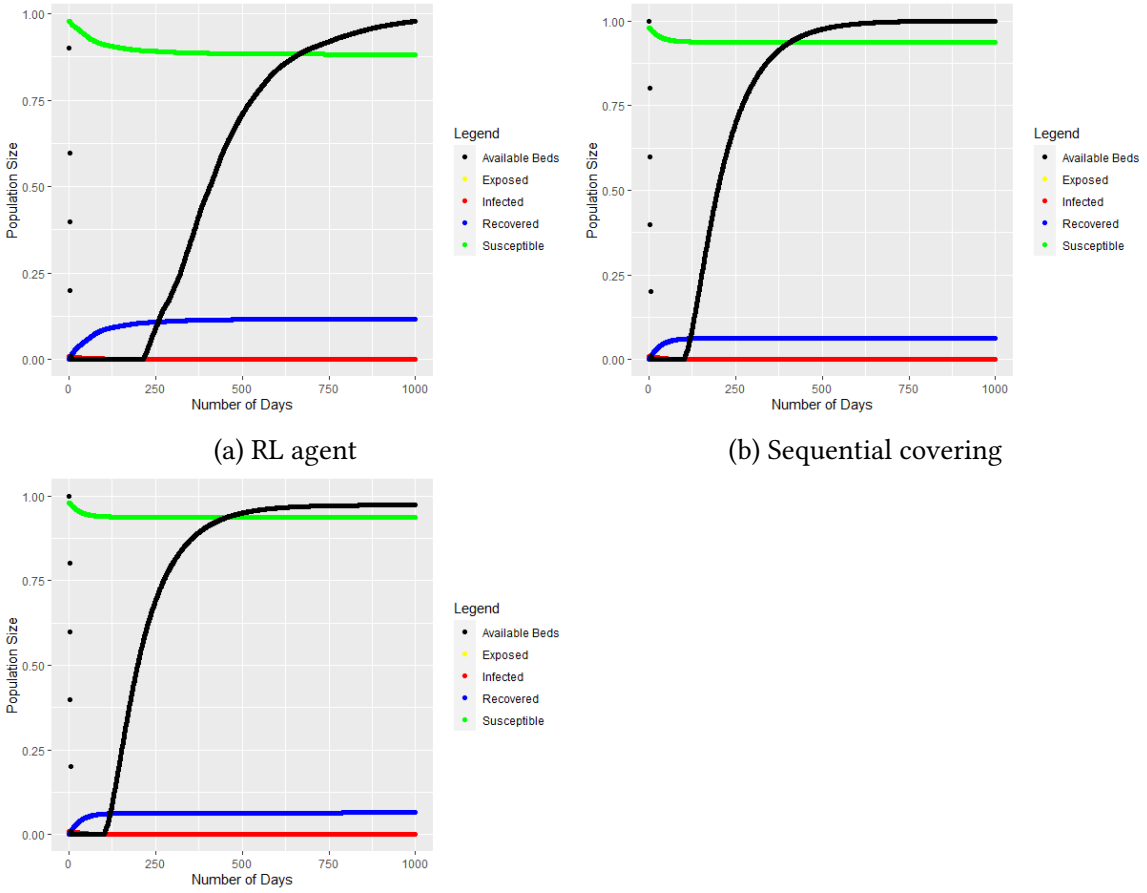
The plots of Figure 6.31 illustrate the development of the SEIR categories and number of hospital beds, see **metric M3.3**. Agents that applied the mined rules, performed comparable or even better than the RL agent. The proportion of infections could be kept significantly low by each agent, while in turn the proportion of susceptible people kept high. However, in all plots it can also be determined that for some time no (sufficient) hospital beds were available until the infections were contained. This might be due to the fact that the number of hospital beds (i.e. 250) was very low compared to the population size (i.e. 100000) and initial infections and exposures (i.e. 1000). Moreover, the configuration of rewards within the task simulation model can be adjusted so that even better results or different objectives can be achieved.

In both rule mining algorithms, infections remain very low in the first 125 days, while there are not enough hospital beds available for infected people. The number of susceptible people decreases by approximately 10%. The proportion of recovered people increases

within the first 125 days and keeps consequently equally low as the proportion of susceptible people, i.e. also approximately 10%.

The RL agent achieves for susceptible people a decrease of approximately 25% while the proportion of recovered people increases up to approximately 25%. However, the RL agent requires approximately 250 days until the number of available hospital beds increases.

At the beginning of the pandemic, for all 3 algorithms, the proportion of available hospital beds is below the infection curve, suggesting that the derived sequence of measures is still suboptimal, see **metric M3.3**. However, in terms of infections and containment time, the results of both rule mining agents are significantly better than the result of the RL agent.



(a) RL agent

(b) Sequential covering

(c) OneR

Figure 6.31.: Development of the SEIR categories based on the imposed measures. Since the exposures overlap with the infections, they cannot be seen in the plots.

The plots of Figure 6.32 depict that the genetic algorithm (GA) agent performs worse than the tested rule mining agents because its received punishments are significantly higher. OneR and sequential covering result in a quite similar curve shape and both require approximately 125 days until their cumulative reward (see **metric M3.2**) turns into a positive, i.e.

increasing, direction. However, OneR reaches after 172 days again an inflection point (see **metric M3.3**) that leads to a decreasing cumulative reward development, while sequential covering positively keeps its performance. It has to be acknowledged, though, that in the case of the *COVID-19 measures* task, the cumulative reward development is not particularly meaningful because the development of the reward curve depends strongly on the task configuration and reward assignment and this in turn depends on the determined goals of the task. In the COVID-19 task, a trade-off between diverging goals are prevalent and this trade-off is reflected to a certain extent in the cumulative reward development. However, it is apparent that the rule mining agents perform better than the GA agent in terms of rewards received.

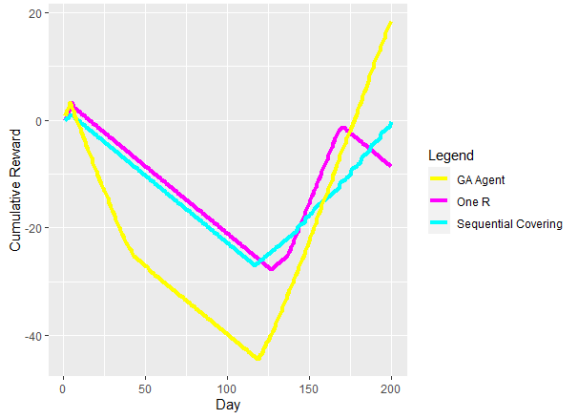
The reward histograms of the Figures 6.32b - 6.32d show that the GA agent receives more rewards with higher negative values than the rule mining agents. However, it also achieves a high number of positive rewards compared to the rule mining agents and covers therefore the entire spectrum of rewards that are receivable. This is due to the fact that the GA agent executes all possible measures while the rule mining agents perform only certain measures high frequently while they omit others. Figure 6.33 illustrates this situation. The most frequently imposed measure of the rule mining agents is *TotalConfinement* while the least one is *NoRestriction*. This indicates that the rule mining agents derived rules from the GA agent's dataset that are highly restrictive and thus contribute to the objective of keeping the infections low and the availability of hospital beds high.

The development of each rule mining agent's SEIR curve in Figure 6.34 confirms the previously outlined observation. The goal to keep the infections low and the proportion of susceptible persons high as well as the hospital beds sufficient, has been accomplished by the rule mining agents, while the GA agent shows for more than 75 days a lack of hospital beds and a significantly high curve of infected and exposed persons. Only after more than 100 days does the number of hospital beds increase again.

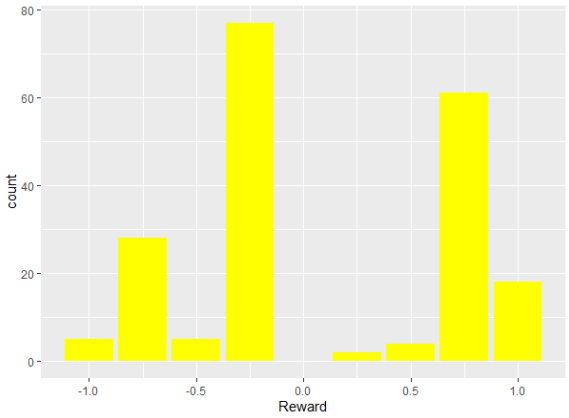
Some conclusions can be drawn from these results. The evaluated rule mining agents show satisfactory performance compared to the other evaluated approaches, e.g. GA, if the data they use for their rules replicate the goals and desired behaviour. Thus, rule mining approaches can be a fast alternative to learning algorithms, provided that good quality data about the intended processes are available. However, it still depends on the simulation model and the assumptions and goals of the domain experts which measures are reinforced by the model and the agent framework during the data generation process.

It can be stated that the made assumptions have to correctly reflect the objectives in the simulation model so that RL agents can be reinforced by the framework to take the optimal measures that contribute to the intended task objectives.

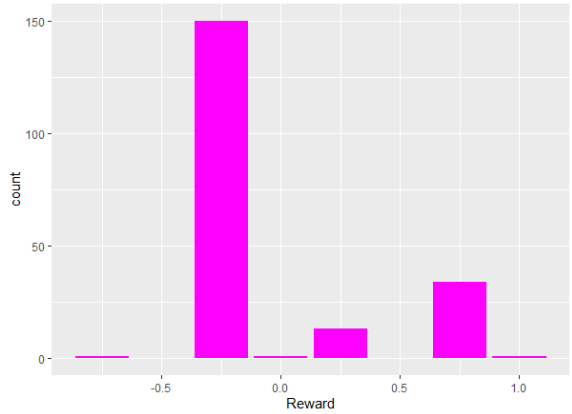
The obtained performance results of the agents suggest that hypothesis EH3.1 is true since the derived rules enable agents to achieve an equal or even better performance during task execution.



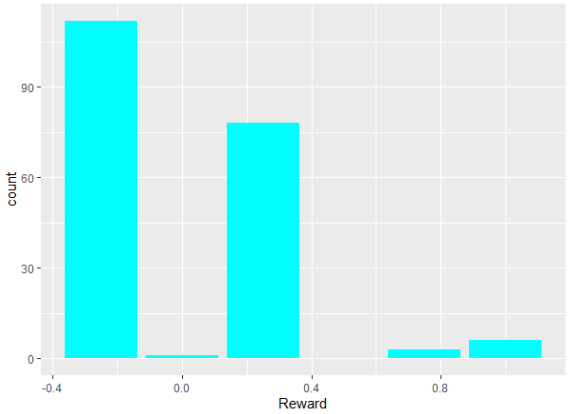
(a) Cumulative reward of sequential coverage, OneR and RL algorithm.



(b) Reward histogram of RL agent.

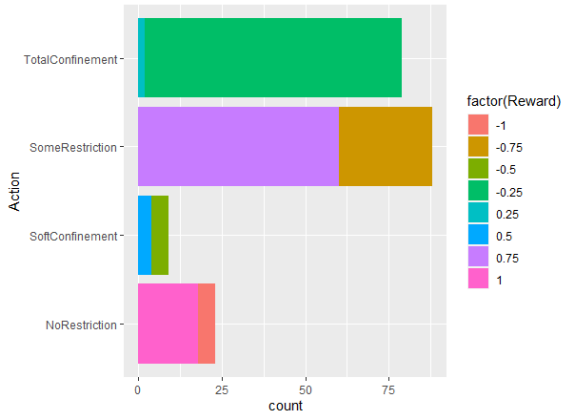


(c) Reward histogram of OneR algorithm.

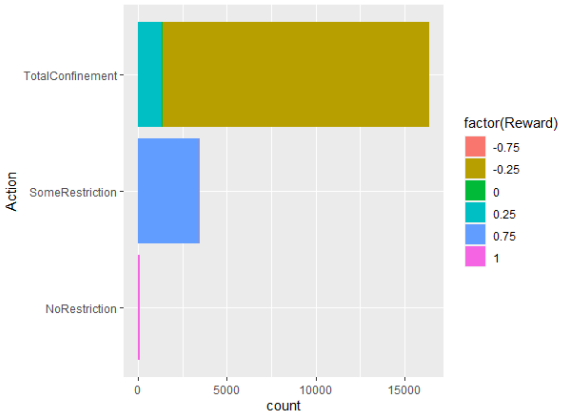


(d) Reward histogram of Sequential Covering algorithm.

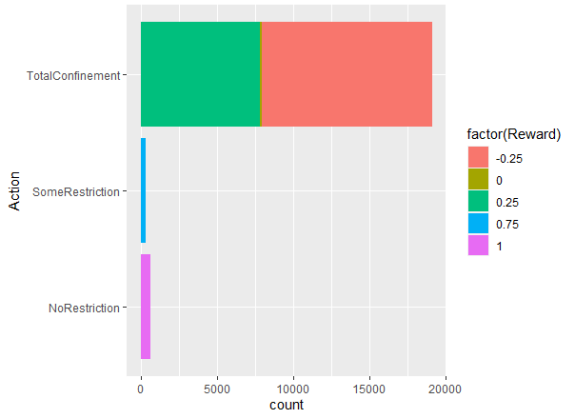
Figure 6.32.: COVID-19 measures: GA agent performance versus mined rules performance.



(a) GA agent.

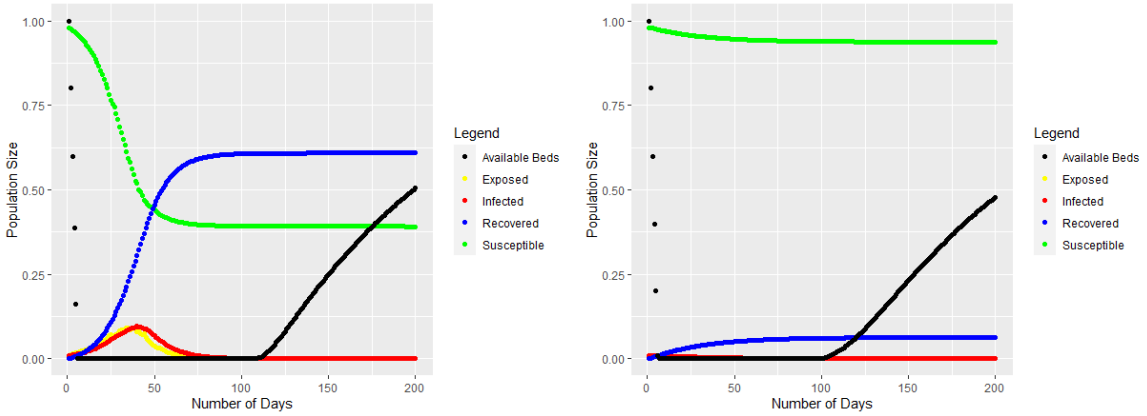


(b) OneR algorithm.



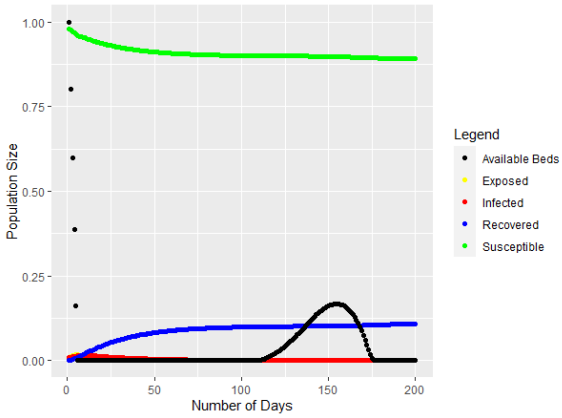
(c) Sequential Covering algorithm.

Figure 6.33.: COVID-19 measures: Performed actions of each algorithm and received rewards per action.



(a) RL agent

(b) Sequential covering



(c) OneR

Figure 6.34.: Development of the SEIR categories based on the imposed measures. The exposures (yellow) in Fig. 6.34b and Fig. 6.34c overlap with the infections and therefore cannot be seen in the plots.

6.3.4.2. Results of Task Embeddings Evaluation

Table 6.10 lists VH tasks that were randomly selected during the evaluation run. The table also depicts for every selected task the corresponding most similar task, the sequence of actions required to conduct the task, and the number of actions that are common and not common to both tasks.

Moreover, the table provides the determined cosine distance as well as the Jaccard similarity between the original task and the most similar task. The lower the cosine distance is the closer two task vectors to each other are. It is evident that for almost every task the cosine distance is quite low while the Jaccard similarity is rather high, what is a consistent and desired outcome.

Only between the *Read book* tasks, a moderate Jaccard similarity and cosine distance is measured. This is explainable by the lower number of actions (i.e. 6) that match for both tasks, while each task consists of 9 actions that are required to conduct both tasks. This in turn means, that both tasks vary regarding 3 actions. However, the plots in Figure 6.38 illustrate that nevertheless the policies of *Read book 7* can be successfully reused by *Read book 25* and vice versa because the agent adjusts its policies based on the framework's feedback to the actual task that it conducts. It is important to not that the *Read book 7* task also shows that the agent does not require many steps until it adjusts its policies to the task at hand.

Another remarkable result is illustrated by the bar charts reflecting the distribution of actions, see **metric M3.4**. It can be observed that actions that do not correspond to the actual task are refrained from by the agent after the adjustment of the given policies.

The findings discussed thus suggest that task embeddings are an efficient means of retrieving similar task entities and related policies because agents do not have to learn the policies from scratch and can thus avoid lengthy ML procedures such as data pre-processing and policy training.

Figure 6.35 depicts screen shots of the spatial distribution of the considered task embeddings and their neighbours, visualised by Tensorflow's *Embedding Projector*[18].

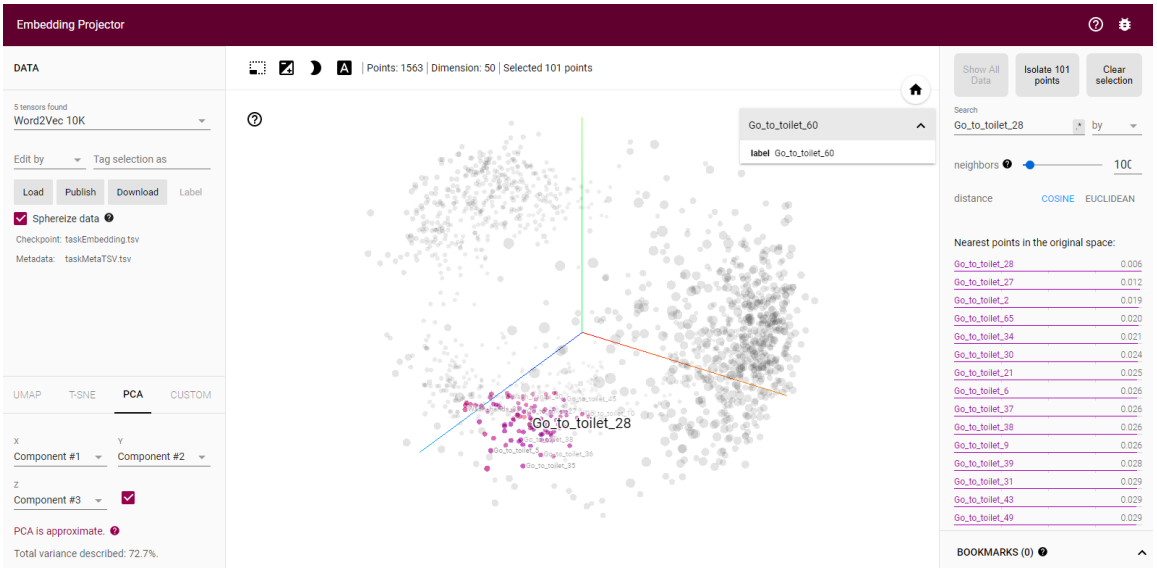
To visualise and compare the learned embedding vectors, two separate TSV[19] files, containing the weights of the embedding vectors and corresponding task labels, were generated and uploaded to the Projector page. The Projector page also lists on the right side, the nearest neighbours to a selected embedding vector and the computed Euclidean or cosine distance between both vectors.

In order to assess the re-usability of policies for the most similar tasks, the cumulative reward (see **metric M3.2**), the distribution of reward values and the distribution of per-

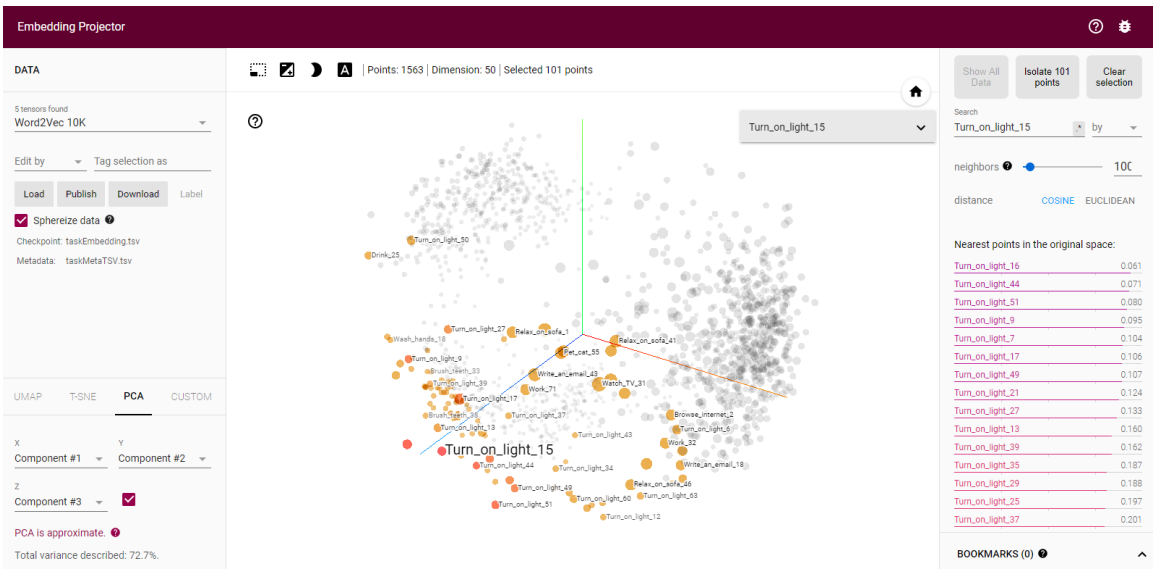
[18] `https://projector.tensorflow.org/`
[19] Tabular-separated values.

Task	Action Sequence	Most similar task	Conjunct Actions	Disjunct Actions	Cosine Distance	Jaccard Similarity
Go to toilet 28	Walk bathroom Walk toilet Open toilet Sit toilet Find toilet_paper Wipe toilet_paper StandUp Flush toilet Close toilet	Go to toilet 60	8	2	0.006	0.8
Turn on light 15	Walk living_room Walk lightswitch Find lightswitch SwitchOn lightswitch	Turn on light 16	4	0	0.061	1.0
Read book 7	Walk living_room Walk textbook Find textbook Grab textbook Walk living_room Walk chair Find chair Sit chair Read textbook	Read book 25	6	5	0.046	0.54

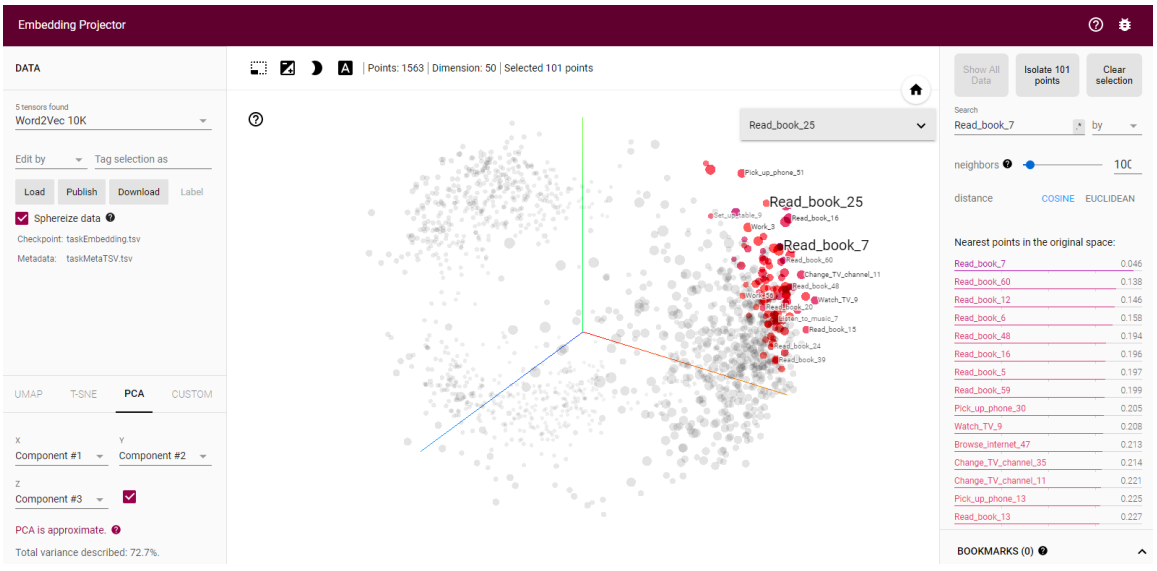
Table 6.10.: Comparison of most similar tasks by their embedding vectors.



(a) Go to toilet task embeddings.



(b) Turn on light task embeddings.



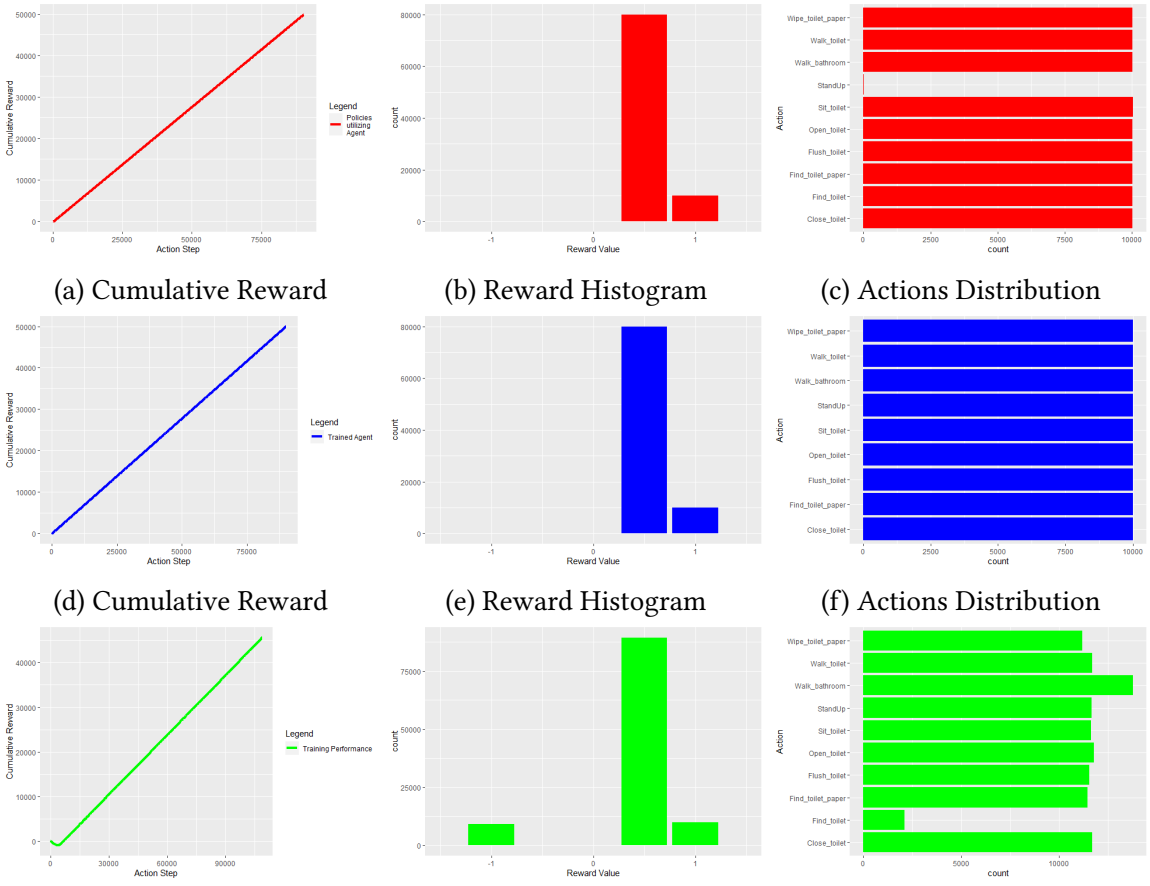
(c) Read book task embeddings.

Figure 6.35.: Embedding Projector showing trained task embeddings.

formed actions (see **metric M3.4**) were considered. Therefore, policies that were trained per task, were reused for the most similar corresponding task. The Figures 6.36 - 6.38 illustrate the corresponding results for each VH task.

The red coloured diagrams show the results for the reuse of policies for the actual task. The blue coloured diagrams depict the application of trained policies applied to the originally trained task and lastly the green coloured diagrams represent the performance and action distribution during the training of policies. The purpose of this three-fold evaluation is to compare and keep track of changes, concerning the performance and selection of actions.

The diagrams illustrate that in all three cases, the reused policies allow the agents to solve the assigned task and lead to an increasing positive cumulative reward, see **metric M3.2**. Moreover, hardly any negative rewards are achieved besides during the training phase. However, this is expectable, because during the training of policies, the agent requires to conduct exploration steps that consequently can lead also to wrong decisions and actions. The agent learns as well by negative feedback to avoid actions that lead to a negative outcome. The principle of *trial and error* applies here.

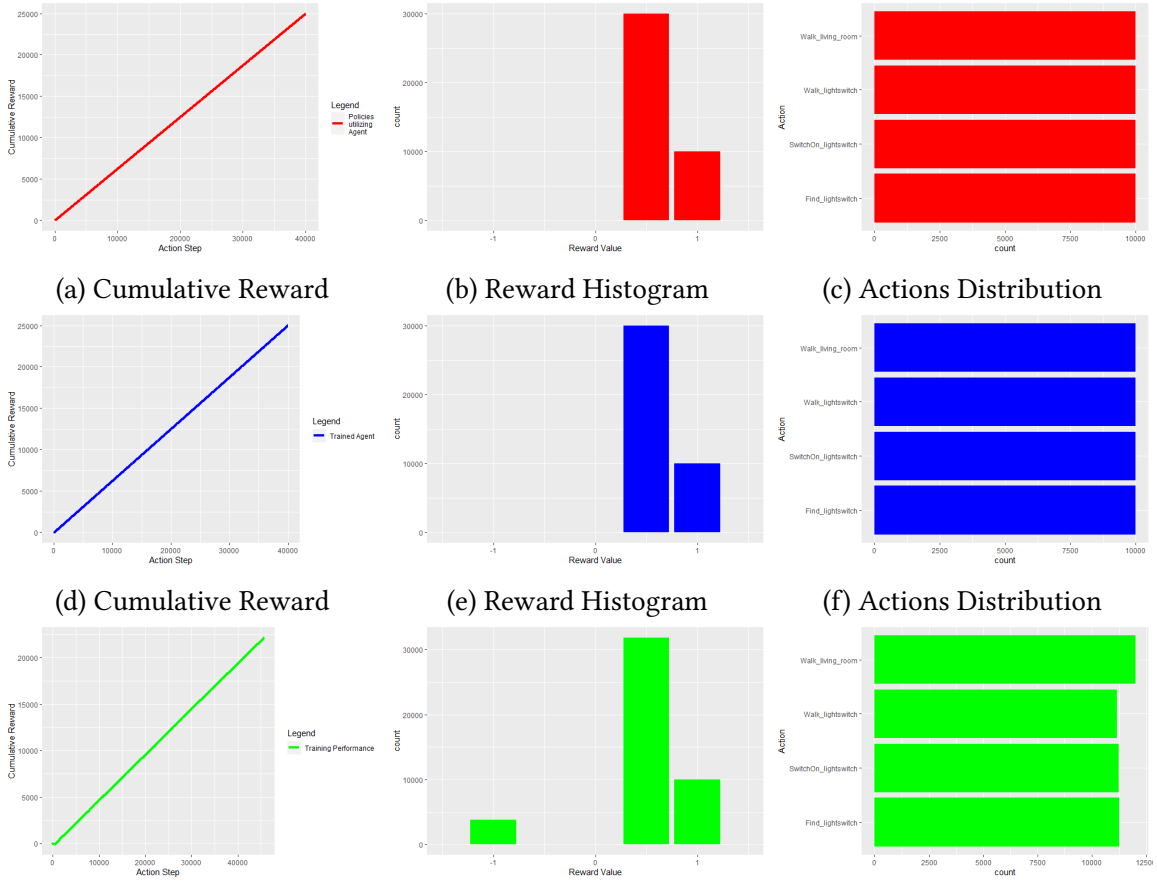


(a) Cumulative Reward (b) Reward Histogram (c) Actions Distribution

(d) Cumulative Reward (e) Reward Histogram (f) Actions Distribution

(g) Cumulative Reward (h) Reward Histogram (i) Actions Distribution

Figure 6.36.: Go toilet 28 policies trained and applied to Go toilet 60 task.



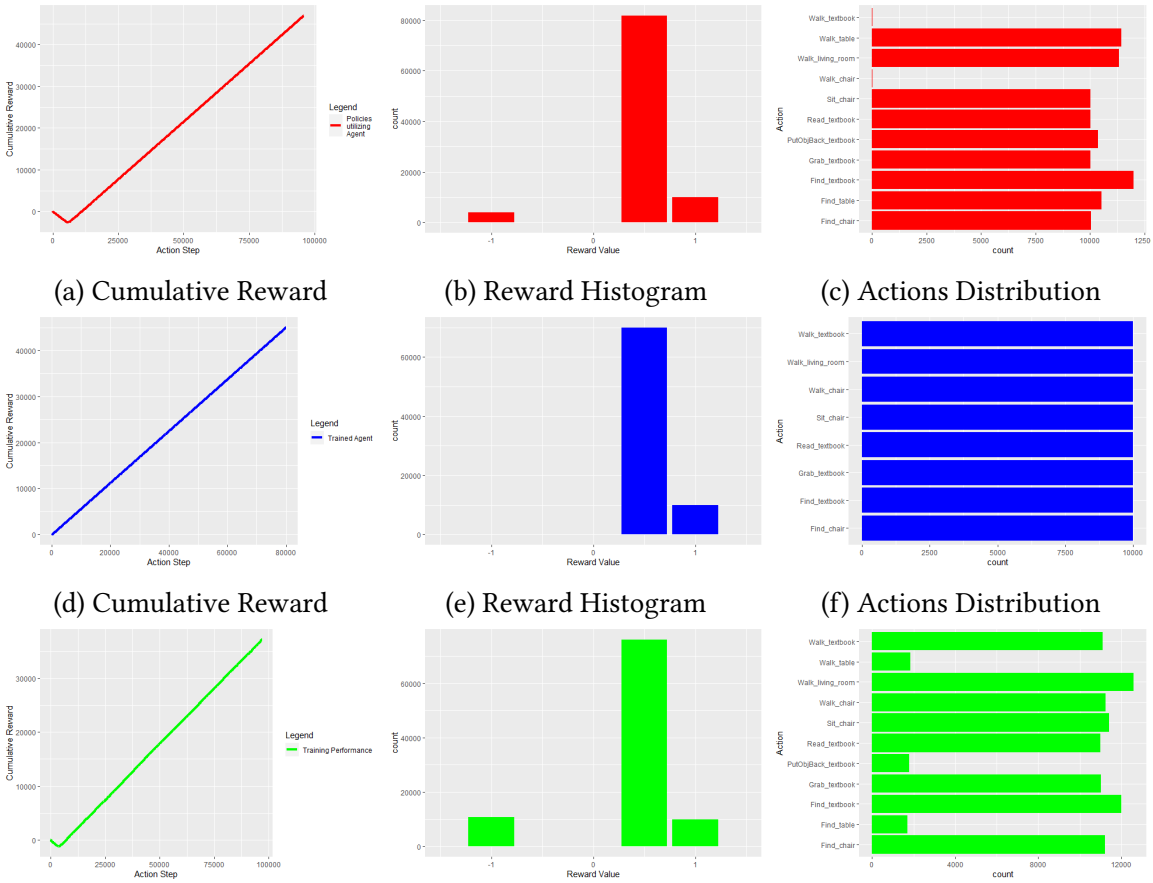
(a) Cumulative Reward (b) Reward Histogram (c) Actions Distribution

(d) Cumulative Reward (e) Reward Histogram (f) Actions Distribution

(g) Cumulative Reward (h) Reward Distribution (i) Actions Distribution

Figure 6.37.: Turn on light 15 policies trained and applied to Turn on light 16 task.

The evaluation results suggest that the use of task embeddings, as outlined in hypothesis H3.2, allows agents to find similar tasks and reusable policies so that they do not have to learn the policies from scratch, but can simply use task embeddings to find the most similar and consistent task that already provides pre-trained guidance.



(a) Cumulative Reward (b) Reward Histogram (c) Actions Distribution

(d) Cumulative Reward (e) Reward Histogram (f) Actions Distribution

(g) Cumulative Reward (h) Reward Histogram (i) Actions Distribution

Figure 6.38.: Read book 7 policies trained and applied to Read book 25 task.

6.4. Threats to Validity

This section introduces *validity threats* [48] based on Cook's and Campbell's definition and discusses them in terms of the evaluation conducted [279]. The knowledge about the *validity threats* presented in this section, is adopted from the book [279] of Wohlin et al. and can be used as a tool to assess during the experimental design whether the validity of the evaluation results is affected by different aspects. This is done by assessing the design and implementation of the evaluation, the subjects selected and measurements made. Validity threats are sometimes unavoidable, and knowledge of them helps to uncover specific validity problems related to the assumptions and results of an experiment. The aim is to avoid these threats as much as possible by reconsidering experimental design decisions, treatments and the (statistical) tools used, such as e.g. questionnaires, metrics and the selection or combination of the subjects evaluated.

Since in this work different experiments are performed to evaluate each approach with respect to the posed RQs, in this section the validity threats that could challenge the validity of the obtained results are considered and outlined. It is important to note that not all validity threats are applicable to all conducted experiments and sometimes a *trade-off* between different validity threats has to be taken into account and the removal of one of these threats may lead to the strengthening of another validity threat [279]. This fact implies that it is sometimes unavoidable to accept some validity threats and to interpret the obtained results in the light of these threats.

Regardless of the threats to validity, it is worth mentioning that the related work, discussed in Sec. 3, except for the baseline approaches in Sec. 3.2.2, do not serve as a baseline for the experiments, i.e. case studies, conducted. This is because related work similar to the agent framework presented here, or following its application goals and processes, does not yet seem to exist and thus only individual approaches and contributions of the proposed agent framework can be evaluated in the context of the use cases defined in Sec. 5 and the considered related work in Sec. 3.2.2. The majority of the related work considered was either purely domain-specific approaches that were hardly applicable to the defined case studies or they were not reproducible and implementable for the experiments. This circumstance made it necessary to define a wide range of use cases from different domains in order to evaluate the applicability, performance and generalisability of the agent framework in the scope of the evaluation criteria, i.e. the metrics specified here. However, the approach to combine policies could be evaluated to a limited extent compared to the approaches discussed in Sec. 3.2.2 that served as baselines.

According to Cook and Campbell, there are four different types of *validity threats* that can impact the validity of experimental outcomes [279]:

i. **Conclusion validity** – Assesses whether the outcomes of an experiment are reproducible and whether the drawn conclusions are correct and reliable.

ii. **Internal validity** – Assesses the (unknown) causal effects to independent variables of a test. The possible internal threats are categorised as *single group, multi group* and *social threats* and concern the test subjects.

iii. **Construct validity** – Assesses whether the design, i.e. the *construction*, of the experiments poses a threat to the validity of the results.

iv. **External validity** – Assesses whether the results of an experiment are generalisable to "industrial practise" [279].

Each threat to validity can be evaluated through different criteria that will be outlined in the appropriate sections with respect to the evaluation in this thesis.

6.4.1. Conclusion Validity

Low statistical power The statistical power of obtained data can be usually figured out by a *power analysis* and *hypothesis testing* where a *null hypothesis* H_0 is to be rejected, given that a certain significance level (p-value) is exceeded or fallen below. In a power analysis some parameters have to be known or assumed in advance, e.g. the *significance level*, the desired probability of having true positives and the *effect size* between different tested data. However, in the performed evaluation no power analysis as well as hypothesis testing, with p-values, has been conducted since a) it is not a common and useful approach for evaluating software agents[20], b) it was difficult to apply it to the given evaluation data since prior statistical assumptions about the data distribution (e.g. Gaussian) would have been required that introduce further sources of error, and c) Hubbard et al. argued and showed that p-values are no reliable and informative statistical measure for assessing the significance and effect of the measured observations, because *"p values exaggerate the evidence against* H_0*"* [110]. According to their review, *confidence intervals* and *effect sizes*, e.g. *Cohen's d*, are more expressive than p-values, since p-values only indicate depending on the sample size and overlap of data distributions whether a null hypothesis can be rejected in favour of a posed alternative hypothesis [110]. However, a p-value does not indicate how big the differences between compared evidences, i.e. probability distributions are. As earlier mentioned, another commonly used statistical test are confidence intervals. Confidence intervals are used in *frequentist statistics* and have the disadvantage that if there is an overlap of confidence intervals between different probability distributions, another statistical test is still required to assess whether there is a statistically significant difference between the two distributions. Furthermore, the sample size influences the spread of confidence intervals. For instance, Morey et al. argue that it should be avoided to apply confidence intervals since they allow *"only the shallowest of interpretations, and are not well-suited to the needs of scientists"* [176].

Since the commonly known statistical tests have the outlined drawbacks, and the evaluated metrics, e.g. cumulative reward or required steps for solving a task, have no known probability distribution in advance, statistical tests for comparing the performance of

[20] The common approach for evaluating agents has been introduced in Sec. 2.3.7

agents, except for *effect size* and *confidence intervals*, were omitted in the evaluation. It can be argued that the combination of effect size and confidence intervals already provides sufficient information about the significance of the results when used together. For this reason, *effect size* and *confidence intervals* were used as statistical measures in the evaluation.

In terms of the conducted evaluation, one problem was that the *population mean* of the observed performance indicators, such as e.g. the agent's obtained *cumulative reward* and its *learning velocity* were unknown in advance because there exist no common reference values or ground-truth for both assessed performance indicators in scientific literature that could have been used to evaluate against. For this reason informed rule-based agents and agent performance testing benchmarks such as the MiniWob++ platform were utilised in order to evaluate the framework agent against. However, the obtained results from the different approaches were so significantly different with respect to their mean values, standard deviations and effect sizes and thus, showed a clear, separable pattern that allowed for the conclusions in the evaluation results sections. The significance of the results have been ensured by performing several runs of the experiments, so that random chance data, i.e. noise, could be eliminated. Furthermore, for the evaluation of the experiments in Sec. 6.1, the *effect size*, i.e. *Cohen's d*, and *confidence intervals* with 95% coverage were calculated to assess the *significance* of the evaluation results. The presented plots illustrate the clear differences in the results of the compared agents.

Violated assumptions of statistical tests As outlined earlier, no statistical tests were conducted in the scope of the evaluation, thus, no assumptions in terms of statistical tests could be violated.

Fishing and the error rate To prevent "fishing" for desired outcomes, experimental constraints and conditions, e.g. number of runs, hyper-parameters, metrics and definition of the evaluation goals, design and procedure, were clearly defined in advance before the experiments were conducted. In addition, all results of the experiments, including the poor ones, were reported and aggregated by statistical measures such as *mean value, standard deviation, standard error, variance*, etc. Since no hypothesis testing was performed, no significance level and therefore no error rate was relevant in the evaluation of the results.

Reliability of measures To ensure reliability and reproducibility of the measured evaluation results, the evaluation set-up was pre-defined with the applied algorithms, evaluation benchmarks (e.g. MiniWob++), hyper-parameters, configuration settings and performance indicators. In addition, some practical implementations, datasets and experiments were documented and made available on Github[21] and Figshare[22] [154] so that the assessment can be reproduced and evaluated under the same or different conditions.

[21] see repositories: `https://github.com/nmerkle/K-Cap-2019-Demo`, `https://github.com/nmerkle/K-Cap2019`, `https://github.com/nmerkle/SW_Journal`

[22] `https://doi.org/10.6084/m9.figshare.22215079.v1`

Reliability of treatment implementation The implementation of the evaluation was under the same conditions for each evaluated software agent. This was ensured through the same computation and simulation environment, the same tested tasks, the same number of runs, the same hyper-parameter and configuration settings.

Random irrelevancies in experimental setting The simulation of tasks as well as the evaluation of the corresponding agents were executed under controlled conditions, so that no disturbance variables and external factors could influence the evaluation outcomes. Furthermore, in the experiments no persons were involved who could have introduced random irrelevancies.

Random heterogeneity of subjects This criterion was not relevant for the evaluation since no heterogeneous study groups were involved in the experiments. However, it was intended that heterogeneous agent types were evaluated since the objective was to compare the feasibility of the evaluated approaches against different software agents. The heterogeneity between agents was to some extent intentional and therefore not random, in order to show the advantages of the proposed approaches and agents trained with the proposed agent framework.

6.4.2. Internal Validity

The criteria for assessing internal validity threats can be sub-divided into *single group threats, multi group threats* and *social threats* although the last criterion may not be relevant to the experiments envisioned in this work. The mentioned criteria and their relation to the experiments in this work are explained in more detail in the following sections.

6.4.2.1. Single group threats

In the experiments, there were no *single-group threats* because the agent that used the framework was compared with other agents, e.g. a pure rule-based agent and an untrained RL agent, which served as the control group and the selected experimental groups, so that the different experimental results could be attributed to the effects of the different approaches.

History A threat to validity from history did not play a role in the experiments because the experiments were always conducted under the same circumstances, which could not be influenced by historical changes, i.e. time and space.

Maturation The maturation of the evaluated agents was intended to some degree, as RL agents were expected to learn policies and improve their strategies to solve the assigned tasks. However, it could be controlled by parameters and learned ML models to either drive this maturation or let the agents start their task performance at zero experience.

Testing This criterion is only relevant for test persons who could learn from feedback loops and thus, was not relevant for the evaluation because the tested software agents required feedback loops, i.e. repeated iterations, of the experiments and feedback from the agent framework, in order to improve their strategies.

Instrumentation The *instrumentation* validity threat might concern experiments that were performed with just one or a few evaluated hyper-parameter configurations. It might be that maybe another hyper-parameter configuration would have yield in different results than the obtained ones. However, this threat can be resolved by hyper-parameter tuning.

Also, the execution steps of the experiments were partly limited to 1000 runs and maybe a longer execution would have shown different results, however one demand to agents was to show a high utility, i.e. learning performance, in a short time span what explains the limitation of training steps and episodes.

Statistical regression The performed experiments and the selected agents were independent of previous experiments. For this reason, random variations could not occur in the subsequent experiments and their results. This means that statistical regression in the given context is not a criterion for evaluating the validity of the planned experiments.

Selection The selection of agents, i.e. algorithms and implementation of agents, was based on algorithms commonly used for software agents. In particular, rule-based agents, RL agents, statistical, i.e. Bayesian agents, and evolutionary agents were the subject of the conducted experiments, thus giving a wide choice of agents. Certainly, many more algorithms and agent types could have been tested, but in the end, for the intended agent framework, it was not important which algorithm an agent implements, but whether the agent can improve its strategies using the proposed approaches implemented in the agent framework. Thus, it can be stated that the selected agents or algorithms are representative of the agents that are the target users of the agent framework.

Mortality The mortality of subjects is not a criterion that can be applied to the designed experiments because human subjects are not intended for the experiments and thus cannot drop out of the experiments and thus affect the validity of the results.

Ambiguity about direction of causal influence In the experiments, the causation of environmental effects is defined by the formal representation of tasks and it is assumed that agents can influence the state of the environment with their executed actions. Since the simulation component simulates the environment, it is possible to control during the experiments whether and how state changes are caused by the executed actions of the agents. For instance, another assumption in one of the experiments in Sec. 6.2.3.2 is that task complexity can be measured by the sequence of actions that an agent requires to perform to solve and complete the assigned task. To clearly show the causal impact of sequence length on task complexity, different types of agents, i.e. RL agent and another agent implementing the proposed approach, are compared. The experiment in Sec. 6.2.3.2

shows that the longer the action sequence of a task, the more time both agents take and the more erroneous actions they perform. By comparing the agents and their achieved outcomes, the differences in the approaches become evident and comparing different agents and approaches ensures that the direction of causal influences can be recognised.

6.4.2.2. Multi group threats

Multiple group threats are common when studying different groups of subjects. There is a trade-off between single-group threats and multiple-group threats because multiple groups can be influenced by single group threats differently, which can affect or even threaten the validity of the results obtained in the experiment.

Interactions with selection In the planned experiments, interaction with selection is not a threat because there are no human subjects involved in the experiments who might exhibit different behaviours due to different levels of maturity and influences of prior history.

6.4.2.3. Social threats

Social threats can occur in both single- and multi-group experiments. Since these types of threats involve human subjects in particular, they are not relevant to the experiments conducted in this thesis.

6.4.3. Construct Validity

Threats to construct validity can be caused by two different types of threats, namely design threats and social threats. *Design threats* involve the definition and design of experiments, whereas *social threats* involve social factors that might affect the generalisability of experimental results. That means that the observed results of the experiments may not generalise to the hypothesis that was established and evaluated [279].

6.4.3.1. Design Threats

Design threats can be avoided if the planned experiments successfully reproduce the approach or theory being evaluated. The design of the experiments can be evaluated using the following criteria.

Inadequate pre-operational explication of constructs In the context of the experiments, the experimental set-up, evaluated use cases, tasks, goals, configuration parameters, algorithms, metrics, and observable performance indicators, such as learning speed, agent performance or utility, cumulative rewards, task complexity, were specified in detail and unambiguously to avoid the risk of an *inadequate pre-operational explication of constructs*. The defined and also adopted metrics ensure the reproducibility of the collected data and also make it comprehensible what specifically is being measured. Furthermore, the specification of what is to be measured and evaluated is given by the research questions that ask about the performance and learning speed of the agents being evaluated. For instance, the

performance of agents is measured by a utility or fitness function and feedback from the environment, while the successful completion of a task can also be measured by defining target states in tasks that need to be achieved by the respective agent. It can therefore be assumed that what is to be measured is measured in the evaluation of the framework and agents.

Mono-operation bias The measured data were collected using various experimental procedures. For example, learning speed was measured by the execution steps and the episodes performed. The performance of the different agents was measured by the average reward over several iterations of the experiments and the cumulative reward. Different agents performing tasks with different complexity were also evaluated. In addition, balanced datasets with a high number of samples were used or generated to make the observations meaningful. However, for one of the experiments, only the Virtual Home dataset was used to show the performance of the proposed approach.

Mono-method bias In the experiments, different measures were established to evaluate different observations from different perspectives. Also, the obtained measures were compared with each other to find inconsistencies in the observations. The goal was to obtain reliable results that agree with each other and support the findings.

Confounding constructs and levels of constructs This validity threat concerns both the different levels of evaluation constructs and the presence or absence of these constructs. If experimental subjects are at different levels of maturity, this could bias the validity of the results. With respect to the experimental constructs designed in this thesis, the agent that is trained in advance by the agent framework is strategically more advanced than the agents that have yet to learn the policies. For this reason, it could be argued that the agent in question has some advantage and a higher level than the agents being compared to it because of the pre-training. However, this warm-start is exactly what the proposed agent framework is all about. Namely, the experiments aim to show that the agent is able to overcome the cold-start by using the simulation component of the framework. The assumption previous to the evaluation was that an agent can be trained off-line using the framework in order to gain some prior knowledge.

Interaction of different treatments The evaluated agents are not simultaneously involved in different experiments, but the experiments are conducted independently (one after the other), so that the effects of the experiments cannot affect the agents in the subsequent experiments and cannot distort the evaluation results.

Interaction of testing and treatment Compared to human subjects, software agents have no awareness of the tests performed and therefore cannot be sensitised by using the information and learned knowledge about the methods performed to falsify the obtained test results.

Restricted generalisability across constructs Generalisability of the proposed approaches was an important objective and target within this work. However, generalisability can only be evaluated to a certain extent. For instance, during the evaluation of RL agents only one implementation, i.e. deep-q-learning, has been evaluated, however, there are many more reinforcement learning algorithms in *model-free* and *model-based* RL, e.g. policy gradient, A2C/A3C, etc., that could have been implemented and evaluated. Testing all possible algorithms would have been gone beyond the scope of this work. Thus, the author had to decide due to pragmatical reasons, for one commonly used RL algorithm, i.e. Deep-Q-Learning that supports huge state spaces.

Regarding the used datasets, there were also some limitations, since e.g. for one of the experiments only one dataset, i.e. Virtual Home dataset, that specifies agent tasks in a domestic environment, has been evaluated. This dataset provided task descriptions with ordered action sequences and related objects. The dataset was considered sufficient and representative for demonstrating that entity embeddings facilitate and accelerate agents in obtaining policies. However, it has been not evaluated what happens, if continuous states are given that cannot simply be embedded into the vector space of the actions.

Another possible validity risk due to the *restricted generalisability across constructs* is that not all possible domains and tasks could be evaluated during the evaluation. Thus, a selection of domains and tasks had to be made. Therefore, it can only be stated that the results are valid for the evaluated tasks and it is assumed that the approaches can be generalised and transferred to similar tasks as far as the tasks can be formally defined and represented in the proposed agent framework.

6.4.3.2. Social Threats

Social threats such as *hypothesis guessing, evaluation apprehension* and *experimenter expectancies* are irrelevant in the context of the experiments of this thesis, since the test subjects are software agents and not humans that could be influenced by social factors. Therefore, these threats will not be discussed in this thesis.

6.4.4. External Validity

External validity threats can occur when external factors such as people involved, location, and time of experiments play a role. The external validity of tested approaches is given when the evaluated approaches are "generalisable" and applicable "in industrial practice" [279]. The goal of this work is to create an agent framework with generalisable approaches that cover heterogeneous use cases and domains. Therefore, considering this threat to validity is relevant and important to evaluate and, if necessary, question the validity of the planned experiments.

Interaction of selection and treatment The agent framework under evaluation also addresses specific target users, such as domain experts, agent developers, and end-users who use the services provided by the agents. However, evaluating the usability of the

framework was beyond the scope of this work. Considering the evaluated agent types, they might be representative for the intended use cases and domains, since they apply commonly used rule-based and machine learning approaches as well as (semantic) web technologies that are utilised in different domains and application fields, such as e.g. *Internet of Things*. For this reason, the agents and algorithms are considered representative for the intended use and thus, this observation indicates that the evaluated approaches seem to be also applicable in industrial practice.

Interaction of setting and treatment The programming languages (e.g. JavaScript) used in the experiments to implement the framework and the agents as well as the test platforms (e.g. MiniWob++), use cases, and environments are representative and commonly used in industrial and scientific practice. Moreover, the APIs of the proposed agent framework allow open access by other platforms and tools used in practice, provided that these platforms follow the specified APIs of the agent framework. Furthermore, the framework ontology enables a formal representation of decision-making processes and is therefore generally suitable for describing processes of different domains. It contains the common properties of tasks and abstracts the interaction of agents with their environment. For instance, states of the ontology represent and aggregate the measured states of the environment. The concepts of the used ontology are generic enough that they can be applied to different tasks. This also implies that they are scalable to industrial tasks. Domain experts have the possibility to describe tasks and their properties in an abstract way based on the ontology without having to go into low-level implementation details.

The evaluated use cases come from heterogeneous domains and can be transferred to industrial tasks where software agents make decisions based on their observations. Furthermore, different task types, i.e. sequential vs. non-sequential, single agent vs. multi-agent, etc., were elaborated in the doctoral thesis, which are also present in the evaluated tasks. This also ensures the scalability and transferability of the evaluated use cases to industrial practice.

Interaction of history and treatment The planned experiments are insensitive to historical influences. This means that the experiments cannot be influenced by different times. Thus, this criterion is not relevant for assessing the validity of the experiments.

6.5. Summary of this chapter

This chapter outlined the different evaluation aspects of the proposed agent framework.

First, the cold-start problem, subject of RQ1, has been examined. The conducted evaluation demonstrates that the framework enables agents in heterogeneous tasks to have a warm-start in terms of adaptation speed and performance. The objective was to evaluate whether a RL agent that is trained by the proposed agent framework, can show a reward maximising behaviour in a relatively short timespan. Therefore, different types of agents (i.e. untrained RL agent, rule-based agent) were evaluated and compared with the RL agent that was previously trained by the proposed agent framework. The results indicate that the trained RL agent outperformed the other agents in most of the cases regarding the evaluated aspects, especially if ambiguous state representations were given and an adaptation of policies was required.

Second, the evaluation investigated in RQ2.1, that is concerned with the elimination of ambiguity and adaptation of trained policies. It was discussed that context changes and personalisation can require the ability of RL agents to personalise and adapt their behaviour to upcoming situations, users and requirements. The corresponding evaluation of RQ2.1 suggests that RL agents that have trained certain ambiguous or unspecific policies can adapt their policies through new and modified feedback that may be given by varying task entity specifications. Therefore, RL agents are highly flexible regarding context changes and personalisation. Thus, the agent framework simulated and reinforced these changing or newly upcoming task requirements, so that agents changed their behaviour in an intended way. The evaluation demonstrated that agents refrained from less reinforced actions and performed actions that promised a positive feedback and reward maximising utility function.

Third, as an alternative to reinforcement learning, a policy combination approach was compared in terms of a) success rate in completing a task and b) speed, i.e. the number of steps and episodes required by the framework to deliver context-related policies for answering RQ2.2. Domestic activities of different complexity, i.e. action sequence length, were evaluated and compared with a DQNN agent. The evaluation indicates that the presented policy combination approach significantly outperforms the DQNN agent in terms of speed and activity completion rate, as the proposed approach required only one episode for each activity to provide the correct sequence of actions for a corresponding state, while the DQNN agent required at least 100 training episodes to correctly perform a few activities. In contrast, the policy combination approach was able to provide the correct sequence of actions for each evaluated activity, regardless of the complexity, i.e. action sequence length, of the activity. The *Virtual Home* dataset served as a ground-truth for assessing the correctness of the action sequences.

Fourth, two different aspects (i.e. the rule mining of data and task embeddings for the retrieval of policies) that concern RQ3 and contribute to the *share and reuse* of policies were considered. For the rule mining approach different algorithms (i.e. frequent item sets,

sequential covering, OneR, Best Fit Sequence) were implemented and tested on simulated data. Then, the performance of the agents that used the mined rules were compared to the performance of the RL agents that trained previously the policies for the intended task. The mined rules were derived from the data that was previously generated by the RL agents. The results of the evaluation show that the agent that applies the mined rules is as powerful as the RL agent that generates the data or even outperforms the RL agent.

For evaluating the task embedding approach, the VH dataset was adopted and transformed into framework-specific task representations. Subsequently, the *Embeddings Trainer* of the framework trained for every VH task the corresponding task embeddings vector, in order to find the most similar task with its corresponding policies. The policies of the most similar task were then adopted and applied to the intended task. It was shown that similar guidelines (policies) that were found by computing the similarity of task embedding vectors, could be successfully applied to similar tasks and that this is a reliable method to find suitable guidelines for solving own tasks.

Fifth, the threats to the validity of the conducted evaluation were discussed in order to identify the limitations and weaknesses of the evaluation. It became apparent that some threats to validity are unavoidable, as an evaluation also has certain constraints or lacks information and resources to avoid all threats that could affect the validity of the evaluation results. However, if one is aware of the threats, the results obtained can be re-interpreted with an awareness of these threats.

Considering all evaluated aspects, it can be stated that the framework fulfils the previously determined and intended objectives. To summarise its functionalities: It allows a) the avoidance of the cold-start, b) the adaptation of agent policies to new context and requirements, c) the mining of rules from data, generated by RL agents, d) the retrieval of suitable task policies by task similarities, e) the integration of mathematical models by means of the framework ontology in order to simulate phenomena, f) a simplified and programming-free integration of tasks and agents and the reproduction of the agent life-cycle by its process flow. As side-effect, the agent framework even allows by the proposed approaches, the generation and enrichment of knowledge graphs, consisting of task instances, what however could not be evaluated in the scope of this work.

7. Summary and Outlook

The first part of this chapter, i.e. Chapter 7.1, discusses and summarises the contributions of this thesis with respect to the posed research questions and evaluation results. In the second part, i.e. in Chapter 7.2, an outlook on potential future work is given, as every scientific work might have its limitations and not all possible problems can be considered and solved exhaustively within one doctoral thesis.

In the motivation chapter, the corresponding problems were addressed and research questions (i.e. RQ1-RQ3) as well as hypotheses on these questions were discussed. The background chapter dealt with the concepts and types of software agents, semantic technologies, data mining and machine learning approaches and paradigms used within the proposed agent framework. In the related work chapter, related work was considered and the differences to the approaches of the agent framework were elaborated, while the subsequent approach chapter presented the methods, concepts and approaches developed for the agent framework. The use cases chapter presented different use case scenarios and example tasks with different requirements from different domains, e.g. the medical domain, that were used to evaluate the framework. In the chapter on the evaluation, the design of the evaluation, the metrics and the scenarios carried out were then presented, and the results of the evaluation were analysed and interpreted.

7.1. Discussion about Contributions

As already outlined in the introduction, this thesis aims for answering three research questions, which include a) the cold-start problem, b) the adaptation of standard policies to real environments and finally, c) the mining and combination of rules and context-based policies.

A main goal of the present work, especially for experts of heterogeneous domains, agent developers and software agents, was to provide a framework that supports all mentioned roles during the integration and runtime of software agents. The agent framework should therefore map and support all process steps that an agent requires to go through until it is executed. The simplified integration of new tasks, algorithms and agent instances should be implemented by the agent framework.

Another aim was to create the possibility for agents to autonomously provide semantically enriched knowledge in the form of policies and meta-information in order to promote the exchange of task knowledge and cooperation among software agents. The claim was that knowledge should be openly accessible in a machine interpretable way to diverse types of

software agents for heterogeneous types of domains and tasks. The following subsections discuss in detail for each research questions the elaborated contributions of this work.

7.1.1. RQ1: The cold-start problem of agents

To solve the outlined cold-start problem that agents face when they are newly deployed in a software system and infrastructure, the framework provides a simulation component and a lightweight framework ontology that enables the semantic formalisation and shared understanding of Markov decision processes (MDPs). To this end, the framework assumes that tasks can be described as MDPs that an agent can go through by its actions to learn the most useful solution strategy.

As the World Wide Web (WWW) and the Semantic Web increasingly connect knowledge through various knowledge graphs, the agent framework also enables domain experts and software agents to create and extend a task knowledge graph by providing and maintaining their task entities, corresponding policies, rules and provenance data.

This thesis proposes concepts and an associated reference model that allow a generic and semantic definition of tasks and agents. Domain experts and agent developers are supported by these concepts in the creation and integration of their tasks and agents, so that domain experts in particular do not need to have knowledge of formal description languages and programming languages. They ultimately need to instantiate these concepts with logical rules that determine states, while the agent framework takes care of the rest by starting and running an elaborated, predefined standard process that prepares agents for their intended tasks. The thesis specifies different types of agents, on the one hand the worker agents, which work closely with the framework, and on the other hand, the consumer agents, which reuse the services and provided knowledge of the agent framework to solve their own tasks.

However, it was also shown that knowledge can also be provided implicitly through behavioural and process-specific datasets by integrating wrapper components, i.e. plug-ins, in a modular way, which manage a dataset-specific transformation of the data's implicit knowledge into semantic task entities. To this end, this thesis has proposed probabilistic approaches that enable MDPs to be described through statistical data analysis and modelling.

The obtained semantic task entities are used by the simulation component to simulate, maintain and update states and provide feedback for agents concerning their performed actions. Through the simulation and training of RL agents, default policies can be learned that can later be adapted to specific environments and individual users. In this way, agents can learn a well-defined behaviour before they go live, in order not to put the user or themselves at risk in the real environment.

Since the simulation component has to recognise states from the rule descriptions of the task entities and also has to derive what feedback, i.e. reward or punishment, should be

returned for an executed action in a certain state, methods were presented that enable the simulation component to derive the aforementioned things.

The evaluation conducted as part of the work showed that agents without prior knowledge who trained policies via the simulation component of the framework were able to learn reward-maximising policies for heterogeneous tasks relatively quickly, as measured by training steps and episodes, compared to other agents. In addition, the trained default policies could be successfully adapted to new task requirements. The simulation component showed its strength and relevance, especially when ambiguous states were given within the tasks, since the evaluated framework agents were able to eliminate this ambiguity by the feedback provided to them through an adapted task representation and simulation. Transferred to industrial practice and real-world applications, the simulation component would allow domain experts and developers to pre-train agents with well-defined standard policies and later adapt their behaviour, i.e. their policies, at runtime based on individual feedback, e.g. from users or the environment.

In summary, it can be stated that the outlined cold-start problem for software agents could be successfully solved with the proposed approaches and concepts implemented in the agent framework.

7.1.2. RQ2: Context-awareness and personalisation of agents

Policy adaptation and agent context awareness is a desired capability of the proposed agent framework, as trained standard policies are not always sufficient to meet individual requirements and user preferences, especially when environments change dynamically. The agent framework solves this problem through an approach that combines reinforcement learning with rule mining. Therefore, agents implement reinforcement learning and adapt their policies by changing feedback from the current environment. The collected data on the adapted policies is then evaluated using different rule mining approaches.

Due to the versatility of tasks and their characteristics, rule mining selection strategies are required to independently select the right rule mining algorithm for the task at hand. For example, tasks can be organised into stages that represent a path of sub-goals leading to the completion of a task. For this reason, an approach is presented to classify tasks based on their characteristics and to select suitable rule mining algorithms. In order to process datasets as input for rule mining, this paper proposes a dataset structure that agents have to adhere to when creating the corresponding datasets so that the rule mining algorithms used can evaluate the datasets. In addition, a methodology for pre-processing datasets is presented that allows raw data to be pre-processed depending on the rule mining algorithm chosen.

Despite the rule mining algorithms adopted for the agent framework, another algorithm called *Best Sequence Fit* was developed and proposed as part of the work presented, which takes into account the sequential nature of some tasks if there are datasets that reflect

sequential tasks, e.g. organised in stages.

Context awareness of agents is achieved within the agent framework by representing context via entity embeddings, which reflect MDP entities and their semantic property relationships by their spatial distance from each other. The proposed methodology uses the numerically embedded context representation to search for similar entities and create policies that belong to the agent's current context. Thus, another contribution of this work is an approach that combines entity embeddings and ensembles of agents to assemble context-aware policies.

This approach enables web agents to request the most appropriate, i.e. reward-maximising, actions based on their current context. As a result, agents working in heterogeneous contexts can react immediately to context changes. The evaluation has shown that the proposed approach for policy combination is much faster compared to reinforcement learning, especially when the sequential tasks or activities become more complex due to a large state and action space.

7.1.3. RQ3: Mining, sharing and reusing of produced expert knowledge

The paradigm of the framework is to be open and enable cooperation. This is achieved through policy sharing and reuse, while task entity descriptions are openly accessible and extensible to all types of agents. For example, obtaining rules from behavioural or demonstration data is a means of deriving MDPs, i.e. task entity descriptions, and policies that can be shared publicly on the web, e.g. via a knowledge graph, with other agents. The advantage of using time series datasets as a knowledge source is that they can provide hidden knowledge that may not have been considered or provided by experts.

Since agents are to be enabled to independently evaluate shared policies, the agents who created the policies also have to provide provenance data. This provenance data is proposed as a semantic representation that describes the policy learning process. For example, information about the origin of the policies and statistical information about the performance of the policies during training, as well as the associated task description, have to be provided so that agents retrieving the policies can decide whether the policies are trustworthy, suitable and useful for their own purposes.

This work presented a methodology that enables the pre-processing and mining of behavioural data and proposed a semantic format that enables the representation of provenance data. It also presented an approach that enables the comparison of tasks in terms of their similarity, so that agents searching for appropriate solution strategies for their own tasks can find them through the framework. Importantly, agents who want to participate in the agent framework and use its services need to adhere to the APIs and recommendations of the agent framework. Therefore, concepts and program flows for the implementation of framework agents have been defined and proposed in this thesis to enable developers to implement agents that adhere to the framework's process flows. A distinction was made between worker and consumer agents.

7.1.4. The simulation of phenomena by mathematical models

Natural and social phenomena have always been of interest to science. For this reason, the simulation of phenomena is an important part of scientific work in order to gain insights into the dynamics, i.e. the long-term and short-term effects of phenomena. The dynamics of phenomena are usually considered from the perspective of cause and effect. In order to be able to describe the dynamics, mathematical models are needed with which phenomena and their development over time can be simulated in order to gain insights and make predictions.

The proposed agent framework addresses the aforementioned points by enabling the integration of mathematical models through a formal representation, i.e. an ontology, that considers equations, variables and parameters in addition to rules, so that effects of intended actions can be described by domain experts. The simulation of these mathematical equations allows agents to try out different behaviours and strategies to predict future outcomes of their behaviour.

Domain experts do not need programming skills to define simulation tasks that represent phenomena. They only need to instantiate the simulation tasks with model equations to test their simulation scenarios with different action sequences that reproduce specific behaviours or strategies. In addition, the framework can be used to create different task entity descriptions that represent different or alternative MDP tasks. The agent framework also facilitates the extension of existing tasks by linking new states, features and actions. Task descriptions can also be extended with additional agents representing different actors within an activity or task.

The creation and integration of new tasks is directly provided by a user interface, e.g. a content management system or a chat bot. In the context of the work, it is proposed that both the implementation of the user interface and the instantiation of task entities should be oriented and guided by the concepts and properties of the underlying framework ontology. Therefore, the agent framework allows introspection and shared understanding of the underlying concepts through the framework ontology provided.

7.1.5. Summary of Contributions

In this thesis, a reference model and architecture for an agent framework was developed and described that reflects the life cycle of software agents from integration to execution and their collaboration. Different user roles involved in the life cycle of software agents were identified. Different concepts and approaches were evaluated and presented that implement or support the different process steps. The main goal was always to create an openly accessible agent framework that enables agents to generate knowledge, e.g. strategies, (semi-)autonomously and to share this knowledge with other agents on the web. Furthermore, methods were presented that enable agents to retrieve strategies tailored to their own problems. Process knowledge implicit in datasets was made explicit by implementing various adopted and reused, but also newly developed rule mining

approaches, so that MPDs and policy rules could be derived, which can be retrieved by agents via the framework and adapted and reused for their own tasks. In this work, it is proposed that agents using the framework also contribute data themselves to sustain the creation of new knowledge that can be extended and reused by other agents.

7.2. Future Work

Almost every scientific work reaches its limits due to lack of time and other resources. For this reason, this section provides an outlook on possible future work to extend the agent reference framework.

7.2.1. Evaluation of Framework Usability

In Sec. 4.2.1 and Sec. 7.1.4, it was claimed and argued that the agent framework aims at providing, compared to other frameworks, simplification in integrating mathematical models, new tasks, and new environments for users who have little or no programming skills and no knowledge of formal rule languages. However, it was omitted in the scope of this thesis to evaluate the agent framework with different users. Future work would consist of a) evaluating the usability of the agent framework and b) comparing it to other related agent frameworks by involving experts from different domains with different technical knowledge and skills.

7.2.2. Security concerns and trust

In the scope of this work, neither *security* concerns nor *trust* were addressed although the provision of provenance data were discussed. However, security and trust are important topics that have to be taken into account in order to counteract the misuses of software agents for unethical activities. It is essential to understand at all times that agents only reproduce and assimilate what they learn from data produced by humans. This implies that biases can also be adopted, which can lead to agents exhibiting misbehaviour such as discrimination. To prevent this, approaches (e.g. from Explainable AI) need to be found that recognise and prevent human-made biases and misuse by agents.

7.2.3. Incorporating fuzzy states by fuzzy logic

The proposed agent framework assumes state representations consisting of features with clearly separable ranges of values. In reality, however, states with numerical features may not always be clearly separable but fuzzy. To capture state probabilities and overlaps in states and their transitions, it would be useful to provide fuzzy states for the agent framework in future work by applying fuzzy logic and considering probability density functions that represent data distributions of Markov decision processes. A possible task here would be to extend the framework ontology with features that indicate probabilities of states. Linguistic variables of domain experts as well as datasets could be used to enhance entities of the ontology and specify fuzzy states reflecting different domain knowledge. As fuzzy states are integrated, agents could be implemented to determine the degree to which an observation belongs to different states. In this context, future work could consist of investigating whether agents with fuzzy states can generalise to tasks better than agents that recognise and evaluate sharply separable states.

7.2.4. Open-source and platform-independent framework distribution

The approaches evaluated were implemented as loosely coupled software modules without being integrated into a platform or overall distribution. A future task will be to combine these modules into a coordinated and orchestrated system so that the framework can be installed and run directly as a platform-independent open source software distribution.

7.2.5. Environmental design and Generative Models for varying Simulations

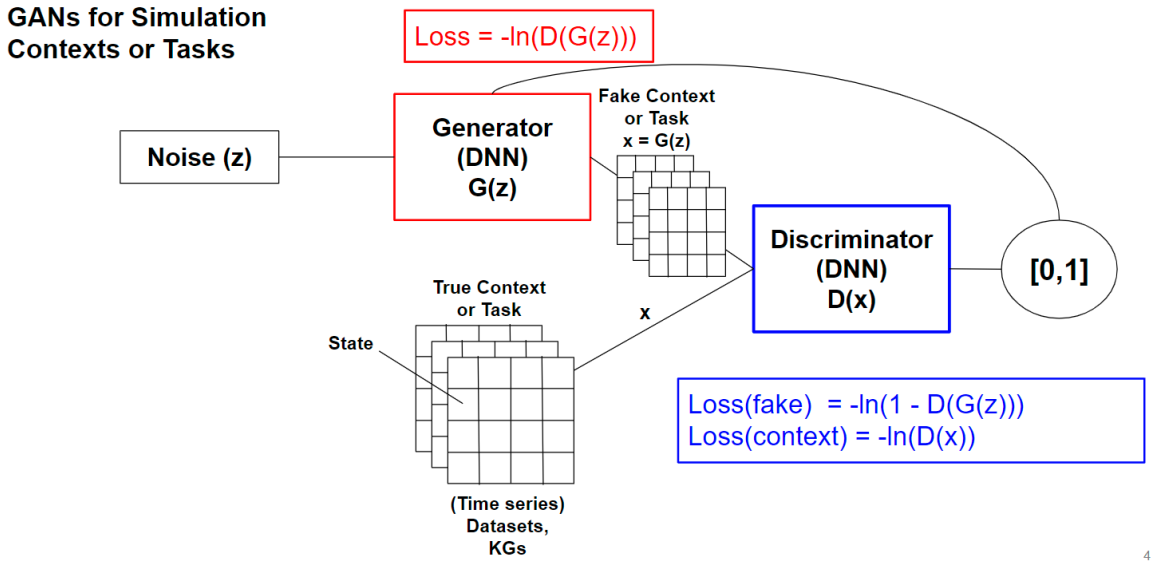
An interesting area of research (see [127, 292, 245, 255, 215, 89]) is *environmental design* for simulation purposes, where different, diverse, realistic and possible simulation scenarios are computer-designed to adapt more reliable, flexible and thus more powerful software agents to environmental conditions. The goal of *environmental design* is to achieve greater generalisability of agents. For example, [60] proposes a "*unsupervised environmental design (UED)*" approach to simulate possible and reasonable variations of 2-dimensional game environments. However, a challenge might be to transfer the considered approaches from game environments to real environments. Future work might therefore consist of developing approaches that allow for the automated, generative creation of different simulation environments and tasks that replicate and vary different and possible real-world scenarios in order to increase the generalisability of agents. Existing approaches that could be helpful here are knowledge graphs that describe different, varying real-world scenarios and provide the environment entities, as well as *attention-based transformers* [262], *variational auto encoders (VAE)* and *generative adversarial networks (GANs)* that consider and reproduce the probabilistic data distributions of Markov decision processes (MDPs) to create alternative simulation scenarios. In this way, GANs could also generate and train rarely occurring or unusual simulation scenarios by taking greater account of outliers in the underlying probability distributions of process data and states.

In this context, one research question could be what task representation and task coding can be used in a GAN or other generative approaches to generate alternative, realistic simulation scenarios, since GANs usually use numerical representations, e.g. pixels of images. Another research question could investigate whether the newly generated states of the GAN are actually possible and realistic based on the previous states and actions performed. Figure 7.1 illustrates two conceivable applications of GANs for creating different task contexts (see Fig. 7.1a) and strategies (see Fig. 7.1b).

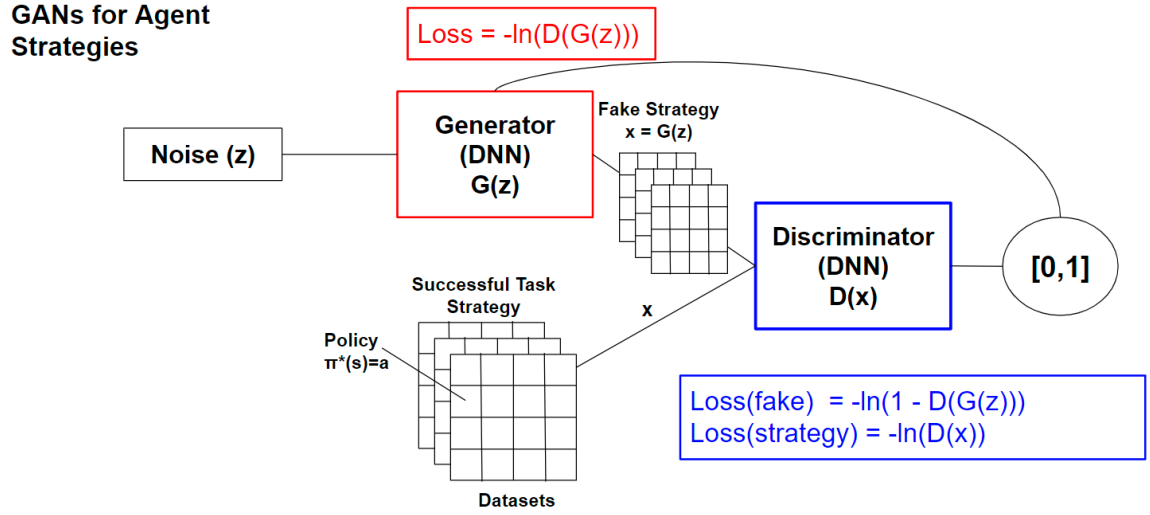
7.2.6. Large language models for transforming natural language into semantic task entity descriptions and policy instructions

Large language models (LLMs) provide a highly interesting means of transforming natural language into other representations. Recently, *ChatGPT*[1] and *Transformer Network* [262] architectures and *self-attention* methods have been developed and used to achieve impressive results. In terms of the agent framework, transformer architectures with attention

[1] `https://chat.openai.com/`



(a) Generating varying tasks and contexts by GANs.



(b) Generating varying policies for tasks by GANs.

Figure 7.1.: Generative adversarial models for task simulation and policy combination.

mechanisms could be used to translate natural language task descriptions from domain experts into semantic task entity descriptions and learned policy instructions. This includes to transform natural language texts from domain experts into knowledge graph representations of tasks. Methods such as Named Entity Recognition (NER) and Named Entity Linking (NEL) could be applied to build the semantic knowledge graphs automatically. In addition, methods would have to be developed that allow logical rules and mathematical differential equations to be extracted from the natural language texts so that the simulation of tasks and the programming of agents could be carried out in natural language. A scientific challenge in this context might be that natural language task descriptions can be incomplete and ambiguous. The scientific approach to be developed would then have to identify or infer which relevant information has been omitted and belongs to the modelling and execution of the task. This requires the collection of datasets that enable the mapping of language into knowledge graphs. For example, potential states, actions and trajectories of observation variables would then need to be extracted and transformed into rules and mathematical equations as well as semantic entities that can be mapped to associated semantic concepts.

7.2.7. A Wikidata knowledge graph for agent programs

In order to use the agent framework on a large scale, it would be conceivable, for example, that Wikidata[2] projects are founded with the aim of providing knowledge graphs for software agents that describe MDPs and common tasks for agents to solve. Agents could autonomously contribute their expertise by creating, developing and reusing different solution strategies in such a Wikidata knowledge graph.

In this way, a network of cross-domain process knowledge from e.g. health-care to industrial applications could be provided and maintained in a decentralised manner, which web-capable agents use worldwide to perform their tasks satisfactorily. The advantage would be that agents could choose from alternative solution strategies, i.e. policies, and solve different tasks without being specifically programmed for only one task. Their capabilities would thus be all-encompassing and flexible and the agents' focus would no longer be on training and solving a specific task, but on finding the right one strategy for the task at hand.

In the long term, such an agent framework with a corresponding web infrastructure could create standards or even norms for a digital ecosystem of software agents and thus also facilitate the cross-domain and ubiquitous use of software agents in everyday life.

[2] `https://www.wikidata.org/wiki/Wikidata:Main_Page`

Bibliography

[1] Wikipedia (english). *Compartmental models in epidemiology*. 2004. URL: https://en.wikipedia.org/wiki/Compartmental_models_in_epidemiology (visited on 09/30/2021).

[2] International Telecommunication Union (ITU). *Y.2060 : Overview of the Internet of things - Recommendation Y.4000/Y.2060 (06/12)*. 2012. URL: https://www.itu.int/rec/T-REC-Y.2060-201206-I (visited on 05/31/2023).

[3] Daghan L. Acay, Gil Tidhar, and Liz Sonenberg. "Extending Agent Capabilities: Tools vs. Agents". In: *2008 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology*. Vol. 2. 2008, pp. 259–265. DOI: 10.1109/WIIAT.2008.247.

[4] Joseph Ahn et al. "Covariance effect analysis of similarity measurement methods for early construction cost estimation using case-based reasoning". In: *Automation in Construction* 81 (2017), pp. 254–266. ISSN: 0926-5805.

[5] Rana Alaa, Mariam Gawich, and Manuel Fernández-Veiga. "Personalised Recommendation for Online Retail Applications Based on Ontology Evolution". In: *Proceedings of the 2020 6th International Conference on Computer and Technology Applications*. ICCTA '20. Antalya, Turkey: Association for Computing Machinery, 2020, pp. 12–16. ISBN: 9781450377492. DOI: 10.1145/3397125.3397134. URL: https://doi.org/10.1145/3397125.3397134.

[6] Carl Allen and Timothy Hospedales. "Analogies Explained: Towards Understanding Word Embeddings". In: *Proceedings of the 36th International Conference on Machine Learning*. Ed. by Kamalika Chaudhuri and Ruslan Salakhutdinov. Vol. 97. Proceedings of Machine Learning Research. Long Beach, California, USA: PMLR, Sept. 2019, pp. 223–231. URL: http://proceedings.mlr.press/v97/allen19a.html.

[7] Ioannis Antonoglou et al. "Planning in Stochastic Environments with a Learned Model". In: *International Conference on Learning Representations*. 2022. URL: https://openreview.net/forum?id=X6D9bAHhBQ1.

[8] Lora Aroyo and Chris Welty. "Crowd truth: Harnessing disagreement in crowdsourcing a relation extraction gold standard". In: *WebSci2013. ACM* 2013.2013 (2013).

[9] Octavian Arsene, Ioan Dumitrache, and Ioana Mihu. "Expert system for medicine diagnosis using software agents". In: *Expert Systems with Applications* 42.4 (2015), pp. 1825–1834. ISSN: 0957-4174. DOI: https://doi.org/10.1016/j.eswa.2014.10.026. URL: https://www.sciencedirect.com/science/article/pii/S0957417414006502.

[10] Radhakisan Baheti and Helen Gill. "Cyber-physical systems". In: *The impact of control technology* 12.1 (2011), pp. 161–166.

[11] Nikola Banovic and John Krumm. "Warming Up to Cold Start Personalisation". In: *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.* 1.4 (Jan. 2018). DOI: 10.1145/3161175. URL: https://doi.org/10.1145/3161175.

[12] K Suzanne Barber and Dung N Lam. "Architecting agents using core competencies". In: *Proceedings of the first international joint conference on Autonomous agents and multiagent systems: part 1*. 2002, pp. 90–91.

[13] K Suzanne Barber et al. "Multi-agent system development: Design, runtime, and analysis". In: *AAAI*. 2004, pp. 1006–1007.

[14] Dennis Barrios-Aranibar and Luiz Marcos Garcia Goncalves. "Learning Coordination in Multi-Agent Systems Using Influence Value Reinforcement Learning". In: *Seventh International Conference on Intelligent Systems Design and Applications (ISDA 2007)*. 2007, pp. 471–478. DOI: 10.1109/ISDA.2007.136.

[15] Behzad Behdani, Zofia Lukszo, and Rajagopalan Srinivasan. "Agent-oriented simulation framework for handling disruptions in chemical supply chains". In: *Computers & Chemical Engineering* 122 (2019). 2017 Edition of the European Symposium on Computer Aided Process Engineering (ESCAPE-27), pp. 306–325. ISSN: 0098-1354. DOI: https://doi.org/10.1016/j.compchemeng.2018.09.027. URL: https://www.sciencedirect.com/science/article/pii/S0098135418310093.

[16] Fabio Bellifemine, Agostino Poggi, and Giovanni Rimassa. "Developing multi-agent systems with a FIPA-compliant agent framework". In: *Software: Practice and Experience* 31.2 (2001), pp. 103–128. DOI: https://doi.org/10.1002/1097-024X(200102)31:2<103::AID-SPE358>3.0.CO;2-O. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/1097-024X%28200102%2931%3A2%3C103%3A%3AAID-SPE358%3E3.0.CO%3B2-O. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/1097-024X%28200102%2931%3A2%3C103%3A%3AAID-SPE358%3E3.0.CO%3B2-O.

[17] Richard Bellman. *Dynamic Programming*. Dover Publications, 1957.

[18] Rob H. Bemthuis et al. "An Agent-Based Process Mining Architecture for Emergent Behaviour Analysis". In: *2019 IEEE 23rd International Enterprise Distributed Object Computing Workshop (EDOCW)*. 2019, pp. 54–64. DOI: 10.1109/EDOCW.2019.00022.

[19] Safa Ben Ayed, Zied Elouedi, and Eric Lefevre. "Exploiting Domain-Experts Knowledge Within an Evidential Process for Case Base Maintenance". In: *Belief Functions: Theory and Applications*. Ed. by Sébastien Destercke et al. Cham: Springer International Publishing, 2018, pp. 22–30. ISBN: 978-3-319-99383-6.

[20] Ralph Bergmann and Armin Stahl. "Similarity measures for object-oriented case representations". In: *Advances in Case-Based Reasoning*. Ed. by Barry Smyth and Pádraig Cunningham. Berlin, Heidelberg: Springer Berlin Heidelberg, 1998, pp. 25–36. ISBN: 978-3-540-49797-4.

[21] James Bergstra and Yoshua Bengio. "Random search for hyper-parameter optimization." In: *Journal of machine learning research* 13.2 (2012).

[22] Andrea L. Bertozzi et al. “The challenges of modeling and forecasting the spread of COVID-19”. In: *Proceedings of the National Academy of Sciences* 117.29 (2020), pp. 16732–16738. ISSN: 0027-8424. DOI: 10.1073/pnas.2006520117. eprint: https://www.pnas.org/content/117/29/16732.full.pdf. URL: https://www.pnas.org/content/117/29/16732.

[23] Homanga Bharadhwaj. “Meta-Learning for User Cold-Start Recommendation”. In: *2019 International Joint Conference on Neural Networks (IJCNN)*. 2019, pp. 1–8. DOI: 10.1109/IJCNN.2019.8852100.

[24] A. Billard and D. Grollman. “Robot learning by demonstration”. In: *Scholarpedia* 8.12 (2013). revision #138061, p. 3824. DOI: 10.4249/scholarpedia.3824.

[25] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Berlin, Heidelberg: Springer-Verlag, 2006. ISBN: 0387310738.

[26] James Blinn et al. “The Simulation of Natural Phenomena (Panel Session)”. In: *SIGGRAPH Comput. Graph.* 17.3 (July 1983), pp. 137–139. ISSN: 0097-8930. DOI: 10.1145/964967.801142. URL: https://doi.org/10.1145/964967.801142.

[27] JesúS Bobadilla et al. “A collaborative filtering approach to mitigate the new user cold start problem”. In: *Knowledge-based systems* 26 (2012), pp. 225–238.

[28] Eleonora Borgia. “The Internet of Things vision: Key features, applications and open issues”. In: *Computer Communications* 54 (2014), pp. 1–31.

[29] Jeffrey M Bradshaw. “An introduction to software agents”. In: *Software agents* 4 (1997), pp. 3–46.

[30] Tim Brys et al. “Reinforcement learning from demonstration through shaping”. In: *Twenty-fourth international joint conference on artificial intelligence*. 2015.

[31] Tung Bui and Jintae Lee. “An agent-based framework for building decision support systems”. In: *Decision Support Systems* 25.3 (1999), pp. 225–237. ISSN: 0167-9236. DOI: https://doi.org/10.1016/S0167-9236(99)00008-1. URL: https://www.sciencedirect.com/science/article/pii/S0167923699000081.

[32] C. Burghart et al. “A cognitive architecture for a humanoid robot: a first approach”. In: *5th IEEE-RAS International Conference on Humanoid Robots, 2005.* 2005, pp. 357–362. DOI: 10.1109/ICHR.2005.1573593.

[33] Christopher Burr, Nello Cristianini, and James Ladyman. “An Analysis of the Interaction Between Intelligent Software Agents and Human Users”. In: *Minds Mach.* 28.4 (Dec. 2018), pp. 735–774. ISSN: 0924-6495. DOI: 10.1007/s11023-018-9479-0. URL: https://doi.org/10.1007/s11023-018-9479-0.

[34] Domenico Cantone et al. “Ontological Smart Contracts in OASIS: Ontology for Agents, Systems, and Integration of Services”. In: *ArXiv* abs/2012.01410 (2020).

[35] José M. Carcione et al. “A Simulation of a COVID-19 Epidemic Based on a Deterministic SEIR Model”. In: *Frontiers in Public Health* 8 (2020), p. 230. ISSN: 2296-2565. DOI: 10.3389/fpubh.2020.00230. URL: https://www.frontiersin.org/article/10.3389/fpubh.2020.00230.

[36] Rui M Castro et al. "Human active learning". In: *Advances in neural information processing systems*. Citeseer. 2008, pp. 241–248.

[37] W. Cedeno and V. R. Vemuri. "On the use of niching for dynamic landscapes". In: *Proceedings of 1997 IEEE International Conference on Evolutionary Computation (ICEC '97)*. 1997, pp. 361–366. DOI: 10.1109/ICEC.1997.592336.

[38] Maiga Chang et al. "Building Ontology-Driven Tutoring Models for Intelligent Tutoring Systems Using Data Mining". In: *IEEE Access* 8 (2020), pp. 48151–48162. DOI: 10.1109/ACCESS.2020.2979281.

[39] Sotirios P. Chatzis, Dimitrios Korkinof, and Yiannis Demiris. "A nonparametric Bayesian approach toward robot learning by demonstration". In: *Robotics and Autonomous Systems* 60.6 (2012), pp. 789–802. ISSN: 0921-8890. DOI: https://doi.org/10.1016/j.robot.2012.02.005. URL: https://www.sciencedirect.com/science/article/pii/S0921889012000334.

[40] Sotirios P. Chatzis and Dimitrios Kosmopoulos. "A Partially-Observable Markov Decision Process for Dealing with Dynamically Changing Environments". In: *Artificial Intelligence Applications and Innovations*. Ed. by Lazaros Iliadis, Ilias Maglogiannis, and Harris Papadopoulos. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 111–120. ISBN: 978-3-662-44654-6.

[41] Richard J Chen et al. "Synthetic data in machine learning for medicine and healthcare". In: *Nature Biomedical Engineering* 5.6 (2021), pp. 493–497.

[42] Sonia Chernova and Manuela Veloso. "Confidence-Based Policy Learning from Demonstration Using Gaussian Mixture Models". In: *Proceedings of the 6th International Joint Conference on Autonomous Agents and Multiagent Systems*. AAMAS '07. Honolulu, Hawaii: Association for Computing Machinery, 2007. ISBN: 9788190426275. DOI: 10.1145/1329125.1329407. URL: https://doi.org/10.1145/1329125.1329407.

[43] Scott Christley, Xiaorong Xiang, and Greg Madey. "An ontology for agent-based modeling and simulation". In: *Proceedings of the agent 2004 conference*. Citeseer. 2004.

[44] Jacob Cohen. *Statistical power analysis for the behavioural sciences*. Academic press, 2013.

[45] J. J. Cole et al. "Personalisation for User Agents". In: *Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multiagent Systems*. AAMAS '05. The Netherlands: Association for Computing Machinery, 2005, pp. 603–610. ISBN: 1595930930. DOI: 10.1145/1082473.1082565. URL: https://doi.org/10.1145/1082473.1082565.

[46] Carlo Combi et al. "Temporal similarity measures for querying clinical workflows". In: *Artificial Intelligence in Medicine* 46.1 (2009). Artificial Intelligence in Medicine AIME 07, pp. 37–54. ISSN: 0933-3657.

[47] Michael Compton et al. “The SSN ontology of the W3C semantic sensor network incubator group”. In: *Journal of Web Semantics* 17 (2012), pp. 25–32. ISSN: 1570-8268. DOI: https://doi.org/10.1016/j.websem.2012.05.003. URL: https://www.sciencedirect.com/science/article/pii/S1570826812000571.

[48] T.D. Cook, D.T. Campbell, and A. Day. *Quasi-experimentation: Design & Analysis Issues for Field Settings*. Rand McNally College Publishing Company, 1979. ISBN: 9780528620539. URL: https://books.google.de/books?id=OKdEAAAAIAAJ.

[49] Stephen Cranefield, Nir Oren, and Wamberto W. Vasconcelos. “Accountability for Practical Reasoning Agents”. In: *Agreement Technologies*. Ed. by Marin Lujak. Cham: Springer International Publishing, 2019, pp. 33–48. ISBN: 978-3-030-17294-7.

[50] Stephen Cranefield et al. “Ontologies for Interaction Protocols”. In: *Ontologies for Agents: Theory and Experiences*. Ed. by Valentina Tamma et al. Basel: Birkhäuser Basel, 2005, pp. 1–17. ISBN: 978-3-7643-7361-0.

[51] Padraig Cunningham. “A Taxonomy of Similarity Mechanisms for Case-Based Reasoning”. In: *IEEE Transactions on Knowledge and Data Engineering* 21.11 (2009), pp. 1532–1543. DOI: 10.1109/TKDE.2008.227.

[52] Josenildo C. da Silva et al. “Distributed data mining and agents”. In: *Engineering Applications of Artificial Intelligence* 18.7 (2005), pp. 791–807. ISSN: 0952-1976. DOI: https://doi.org/10.1016/j.engappai.2005.06.004. URL: https://www.sciencedirect.com/science/article/pii/S095219760500076X.

[53] Jessamyn Dahmen and Diane Cook. “SynSys: A synthetic data generation system for healthcare applications”. In: *Sensors* 19.5 (2019), p. 1181.

[54] Matthew Daigle and Kai Goebel. “Model-based prognostics under limited sensing”. In: *2010 IEEE Aerospace Conference*. 2010, pp. 1–12. DOI: 10.1109/AERO.2010.5446822.

[55] Charles Darwin. *On the Origin of Species by Means of Natural Selection*. or the Preservation of Favored Races in the Struggle for Life. London: Murray, 1859.

[56] S. Das et al. “Incorporating Expert Feedback into Active Anomaly Discovery”. In: *2016 IEEE 16th International Conference on Data Mining (ICDM)* (2016), pp. 853–858.

[57] LEENTJE DE BLESER et al. “Defining pathways”. In: *Journal of Nursing Management* 14.7 (2006), pp. 553–563. DOI: 10.1111/j.1365-2934.2006.00702.x. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1365-2934.2006.00702.x. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1365-2934.2006.00702.x.

[58] Paul De Boeck and Mark Wilson. “A framework for item response models”. In: *Explanatory Item Response Models: A Generalised Linear and Nonlinear Approach*. Ed. by Paul De Boeck and Mark Wilson. New York, NY: Springer New York, 2004, pp. 3–41. ISBN: 978-1-4757-3990-9. DOI: 10.1007/978-1-4757-3990-9_1. URL: https://doi.org/10.1007/978-1-4757-3990-9_1.

[59] Ronald Denaux, Lora Aroyo, and Vania Dimitrova. "An Approach for Ontology-Based Elicitation of User Models to Enable Personalisation on the Semantic Web". In: *Special Interest Tracks and Posters of the 14th International Conference on World Wide Web*. WWW '05. Chiba, Japan: Association for Computing Machinery, 2005, pp. 1170–1171. ISBN: 1595930515. DOI: 10.1145/1062745.1062923. URL: https://doi.org/10.1145/1062745.1062923.

[60] Michael Dennis et al. "Emergent Complexity and Zero-shot Transfer via Unsupervised Environment Design". In: *Advances in Neural Information Processing Systems*. Ed. by H. Larochelle et al. Vol. 33. Curran Associates, Inc., 2020, pp. 13049–13061. URL: https://proceedings.neurips.cc/paper/2020/file/985e9a46e10005356bbaf194249f6856-Paper.pdf.

[61] Michael Dennis et al. *Emergent Complexity and Zero-shot Transfer via Unsupervised Environment Design*. 2021. arXiv: 2012.02096 [cs.LG].

[62] Giorgia Di Tommaso. "SeRenA: A Semantic Recommender for All". In: *Proceedings of the 12th ACM Conference on Recommender Systems*. RecSys '18. Vancouver, British Columbia, Canada: Association for Computing Machinery, 2018, pp. 558–562. ISBN: 9781450359016. DOI: 10.1145/3240323.3240327. URL: https://doi.org/10.1145/3240323.3240327.

[63] Elvis Dohmatob, Guillaume Dumas, and Danilo Bzdok. "Dark control: The default mode network as a reinforcement learning agent". In: *Human Brain Mapping* 41.12 (2020), pp. 3318–3341. DOI: https://doi.org/10.1002/hbm.25019. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/hbm.25019. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/hbm.25019.

[64] Allen B. Downey. *Think Bayes*. Ed. by Mike Loukides and Ann Spencer. Sebastopol, California: O'Reilly Media, 2013. URL: http://it-ebooks.info/book/3055/.

[65] Richard O. Duda, Peter E. Hart, and David G. Stork. *Pattern Classification (2Nd Edition)*. New York, NY, USA: Wiley-Interscience, 2000. ISBN: 0471056693.

[66] A. Dumitrache et al. "Empirical Methodology for Crowdsourcing Ground Truth". In: *ArXiv e-prints* (Sept. 2018). arXiv: 1809.08888 [cs.HC].

[67] Anca Dumitrache, Lora Aroyo, and Chris Welty. "Crowdsourcing Ground Truth for Medical Relation Extraction". In: *ACM Trans. Interact. Intell. Syst.* 8.2 (July 2018), 11:1–11:20. ISSN: 2160-6455. DOI: 10.1145/3152889. URL: http://doi.acm.org/10.1145/3152889.

[68] Anca Dumitrache et al. "Empirical methodology for crowdsourcing ground truth". In: *Semantic Web* Preprint (2021), pp. 1–19.

[69] Arpad E. Elo. *The Rating of Chessplayers, Past and Present*. New York: Arco Pub., 1978. ISBN: 0668047216 9780668047210. URL: http://www.amazon.com/Rating-Chess-Players-Past-Present/dp/0668047216.

[70] Enrique Estellés-Arolas and Fernando González-Ladrón-de-Guevara. "Towards an integrated crowdsourcing definition". In: *Journal of Information science* 38.2 (2012), pp. 189–200.

[71] G. Faraco and L. Gabriele. “Using LabVIEW for applying mathematical models in representing phenomena”. In: *Computers & Education* 49.3 (2007), pp. 856–872. ISSN: 0360-1315. DOI: https://doi.org/10.1016/j.compedu.2005.11.025. URL: https://www.sciencedirect.com/science/article/pii/S0360131505001909.

[72] Nazim Fatès and Vincent Chevrier. “How important are updating schemes in multi-agent systems? an illustration on a multi-turmite model”. In: *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 1-Volume 1*. 2010, pp. 533–540.

[73] Fernando Fernandez, Javier García, and Manuela Veloso. “Probabilistic Policy Reuse for inter-task transfer learning”. In: *Robotics and Autonomous Systems* 58.7 (2010). Advances in Autonomous Robots for Service and Entertainment, pp. 866–871. ISSN: 0921-8890.

[74] J. R. FIRTH. “A synopsis of linguistic theory, 1930-1955”. In: *Studies in Linguistic Analysis* (1957). URL: https://ci.nii.ac.jp/naid/10020680394/en/.

[75] G.D. Forney. “The viterbi algorithm”. In: *Proceedings of the IEEE* 61.3 (1973), pp. 268–278. DOI: 10.1109/PROC.1973.9030.

[76] Jonathan Fuller and Luis J. Flores. “The Risk GP Model: The standard model of prediction in medicine”. In: *Studies in History and Philosophy of Science Part C: Studies in History and Philosophy of Biological and Biomedical Sciences* 54 (2015), pp. 49–61. ISSN: 1369-8486. DOI: https://doi.org/10.1016/j.shpsc.2015.06.006. URL: https://www.sciencedirect.com/science/article/pii/S1369848615000862.

[77] Nathan Fulton and André Platzer. “Verifiably Safe Off-Model Reinforcement Learning”. In: *Tools and Algorithms for the Construction and Analysis of Systems*. Ed. by Tomáš Vojnar and Lijun Zhang. Cham: Springer International Publishing, 2019, pp. 413–430. ISBN: 978-3-030-17462-0.

[78] Yang Gao et al. “Reinforcement learning from imperfect demonstrations”. In: *arXiv preprint arXiv:1802.05313* (2018).

[79] Javier Garcıa and Fernando Fernández. “A comprehensive survey on safe reinforcement learning”. In: *Journal of Machine Learning Research* 16.1 (2015), pp. 1437–1480.

[80] Marino Gatto et al. “Spread and dynamics of the COVID-19 epidemic in Italy: Effects of emergency containment measures”. In: *Proceedings of the National Academy of Sciences* 117.19 (2020), pp. 10484–10491. ISSN: 0027-8424. DOI: 10.1073/pnas.2004978117. eprint: https://www.pnas.org/content/117/19/10484.full.pdf. URL: https://www.pnas.org/content/117/19/10484.

[81] Y. A. Gerhana et al. “The implementation of K-nearest neighbor algorithm in case-based reasoning model for forming automatic answer identity and searching answer similarity of algorithm case”. In: *2017 5th International Conference on Cyber and IT Service Management (CITSM)*. 2017, pp. 1–5.

[82] Raji Ghawi and Jürgen Pfeffer. "Efficient Hyperparameter Tuning with Grid Search for Text Categorization using kNN Approach with BM25 Similarity". In: *Open Computer Science* 9.1 (2019), pp. 160–180. DOI: doi:10.1515/comp-2019-0011. URL: https://doi.org/10.1515/comp-2019-0011.

[83] Behzad Ghazanfari, Fatemeh Afghah, and Matthew E. Taylor. "Sequential Association Rule Mining for Autonomously Extracting Hierarchical Task Structures in Reinforcement Learning". In: *IEEE Access* 8 (2020), pp. 11782–11799. DOI: 10.1109/ACCESS.2020.2965930.

[84] Giulia Giordano et al. "Modelling the COVID-19 epidemic and implementation of population-wide interventions in Italy". In: *Nature Medicine* 26 (2020), pp. 855–860. DOI: 10.1038/s41591-020-0883-7. URL: https://doi.org/10.1038/s41591-020-0883-7.

[85] Norbert Glaser and Philippe Morignot. "The reorganization of societies of autonomous agents". In: *Multi-Agent Rationality*. Ed. by Magnus Boman and Walter Van de Velde. Berlin, Heidelberg: Springer Berlin Heidelberg, 1997, pp. 98–111. ISBN: 978-3-540-69125-9.

[86] David E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. New York: Addison-Wesley, 1989.

[87] Yang Gong-ping and Zeng Guang-zhou. "Mobile Agent Life State Management". In: *The Proceedings of the Multiconference on "Computational Engineering in Systems Applications"*. Vol. 1. 2006, pp. 448–451. DOI: 10.1109/CESA.2006.4281694.

[88] Thomas R. Gruber. "A Translation Approach to Portable Ontology Specifications". In: *Knowl. Acquis.* 5.2 (June 1993), pp. 199–220. ISSN: 1042-8143. DOI: 10.1006/knac.1993.1008. URL: http://dx.doi.org/10.1006/knac.1993.1008.

[89] Izzeddin Gur et al. *Adversarial Environment Generation for Learning to Navigate the Web*. 2021. DOI: 10.48550/ARXIV.2103.01991. URL: https://arxiv.org/abs/2103.01991.

[90] Danijar Hafner. *Benchmarking the Spectrum of Agent Capabilities*. 2021. DOI: 10.48550/ARXIV.2109.06780. URL: https://arxiv.org/abs/2109.06780.

[91] Jiawei Han, Jian Pei, and Yiwen Yin. "Mining Frequent Patterns without Candidate Generation". In: *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*. SIGMOD '00. Dallas, Texas, USA: Association for Computing Machinery, 2000, pp. 1–12. ISBN: 1581132174.

[92] Manjesh Kumar Hanawal et al. "Learning Policies for Markov Decision Processes From Data". In: *IEEE Transactions on Automatic Control* 64.6 (2019), pp. 2298–2309. DOI: 10.1109/TAC.2018.2866455.

[93] Steve Hanneke et al. "Theory of disagreement-based active learning". In: *Foundations and Trends® in Machine Learning* 7.2-3 (2014), pp. 131–309.

[94] Hado van Hasselt, Arthur Guez, and David Silver. "Deep Reinforcement Learning with Double Q-Learning". In: *Proceedings of the AAAI Conference on Artificial Intelligence* 30.1 (Mar. 2016). URL: https://ojs.aaai.org/index.php/AAAI/article/view/10295.

[95] Randy L Haupt and Sue Ellen Haupt. *Practical genetic algorithms*. John Wiley & Sons, 2004.

[96] Bilal Hawashin et al. "An Efficient Cold Start Solution for Recommender Systems Based on Machine Learning and User Interests". In: *2020 Seventh International Conference on Software Defined Systems (SDS)*. 2020, pp. 220–225. DOI: 10.1109/SDS49854.2020.9143953.

[97] Floris den Hengst et al. "Reinforcement learning for personalisation: A systematic literature review". In: *Data Science* 3.2 (2020), pp. 107–147.

[98] Matteo Hessel et al. "Rainbow: Combining improvements in deep reinforcement learning". In: *Thirty-second AAAI conference on artificial intelligence*. 2018.

[99] Thomas Hinrichs and Kenneth Forbus. "Learning Qualitative Models by Demonstration". In: (2012). URL: https://www.aaai.org/ocs/index.php/AAAI/AAAI12/paper/view/4959/5140.

[100] Pascal Hitzler, Markus Krötzsch, and Sebastian Rudolph. *Foundations of Semantic Web Technologies*. 2009.

[101] Pascal Hitzler, Markus Krötzsch, and Sebastian Rudolph. *Foundations of Semantic Web Technologies*. CRC Press, 2010.

[102] Pascal Hitzler et al. *Semantic Web: Grundlagen*. eXamen.press. Berlin: Springer, 2008. ISBN: 978-3-540-33993-9. DOI: 10.1007/978-3-540-33994-6.

[103] Jonathan Ho and Stefano Ermon. "Generative adversarial imitation learning". In: *Advances in neural information processing systems* 29 (2016), pp. 4565–4573.

[104] Sepp Hochreiter and Jürgen Schmidhuber. "Long Short-Term Memory". In: *Neural Computation* 9.8 (Nov. 1997), pp. 1735–1780. ISSN: 0899-7667. DOI: 10.1162/neco.1997.9.8.1735. eprint: https://direct.mit.edu/neco/article-pdf/9/8/1735/813796/neco.1997.9.8.1735.pdf. URL: https://doi.org/10.1162/neco.1997.9.8.1735.

[105] Aidan Hogan et al. "Knowledge Graphs". In: *ACM Comput. Surv.* 54.4 (July 2021). ISSN: 0360-0300. DOI: 10.1145/3447772. URL: https://doi.org/10.1145/3447772.

[106] Robert C. Holte. "Very Simple Classification Rules Perform Well on Most Commonly Used Datasets". In: *Mach. Learn.* 11.1 (Apr. 1993), pp. 63–90. ISSN: 0885-6125. DOI: 10.1023/A:1022631118932. URL: https://doi.org/10.1023/A:1022631118932.

[107] Mokter Hossain and Ilkka Kauranen. "Crowdsourcing: a comprehensive literature review". In: *Strategic Outsourcing: An International Journal* 8.1 (2015), pp. 2–22.

[108] Wei-Ning Hsu and Hsuan-Tien Lin. "Active learning by learning". In: *Twenty-Ninth AAAI conference on artificial intelligence*. 2015.

[109] Junyan Hu et al. "Voronoi-Based Multi-Robot Autonomous Exploration in Unknown Environments via Deep Reinforcement Learning". In: *IEEE Transactions on Vehicular Technology* 69.12 (2020), pp. 14413–14423. DOI: 10.1109/TVT.2020.3034800.

[110] Raymond Hubbard and R. Murray Lindsay. "Why P Values Are Not a Useful Measure of Evidence in Statistical Significance Testing". In: *Theory & Psychology* 18.1 (2008), pp. 69–88. DOI: 10.1177/0959354307086923. eprint: https://doi.org/10.1177/0959354307086923. URL: https://doi.org/10.1177/0959354307086923.

[111] Patrik Hummel et al. "Data sovereignty: A review". In: *Big Data & Society* 8.1 (2021), p. 2053951720982012. DOI: 10.1177/2053951720982012. eprint: https://doi.org/10.1177/2053951720982012. URL: https://doi.org/10.1177/2053951720982012.

[112] Shao-Ming Hung and Sidney N. Givigi. "A Q-Learning Approach to Flocking With UAVs in a Stochastic Environment". In: *IEEE Transactions on Cybernetics* 47.1 (2017), pp. 186–197. DOI: 10.1109/TCYB.2015.2509646.

[113] David Isern, David Sánchez, and Antonio Moreno. "Agents applied in health care: A review". In: *International Journal of Medical Informatics* 79.3 (2010), pp. 145–166. ISSN: 1386-5056. DOI: https://doi.org/10.1016/j.ijmedinf.2010.01.003. URL: https://www.sciencedirect.com/science/article/pii/S138650561000016X.

[114] Jeevamol Joy, Nisha S. Raj, and Renumol V. G. "Ontology-Based E-Learning Content Recommender System for Addressing the Pure Cold-Start Problem". In: *J. Data and Information Quality* 13.3 (Apr. 2021). ISSN: 1936-1955. DOI: 10.1145/3429251. URL: https://doi.org/10.1145/3429251.

[115] Peter Kaiser and Tamim Asfour. "Autonomous Detection and Experimental Validation of Affordances". In: *IEEE Robotics and Automation Letters* 3.3 (2018), pp. 1949–1956. DOI: 10.1109/LRA.2018.2808367.

[116] Anssi Kanervisto, Christian Scheller, and Ville Hautamäki. "Action Space Shaping in Deep Reinforcement Learning". In: *2020 IEEE Conference on Games (CoG)*. 2020, pp. 479–486. DOI: 10.1109/CoG47356.2020.9231687.

[117] Ken Kansky et al. "Schema Networks: Zero-shot Transfer with a Generative Causal Model of Intuitive Physics". In: *Proceedings of the 34th International Conference on Machine Learning*. Ed. by Doina Precup and Yee Whye Teh. Vol. 70. Proceedings of Machine Learning Research. PMLR, June 2017, pp. 1809–1818. URL: https://proceedings.mlr.press/v70/kansky17a.html.

[118] Hamdi Kavak et al. "Big data, agents, and machine learning: towards a data-driven agent-based modeling approach". In: *Proceedings of the Annual Simulation Symposium*. 2018, pp. 1–12.

[119] Khoshnaw, Sarbaz H. A., Salih, Rizgar H., and Sulaimany, Sadegh. "Mathematical modelling for coronavirus disease (COVID-19) in predicting future behaviours and sensitivity analysis". In: *Math. Model. Nat. Phenom.* 15 (2020), p. 33. DOI: 10.1051/mmnp/2020020. URL: https://doi.org/10.1051/mmnp/2020020.

[120] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2017. arXiv: 1412.6980 [cs.LG].

[121] W. Bradley Knox, Peter Stone, and Cynthia Breazeal. "Teaching Agents with Human Feedback: A Demonstration of the TAMER Framework". In: *Proceedings of the Companion Publication of the 2013 International Conference on Intelligent User Interfaces Companion*. IUI '13 Companion. Santa Monica, California, USA: Association for Computing Machinery, 2013, pp. 65–66. ISBN: 9781450319669. DOI: 10.1145/2451176.2451201. URL: https://doi.org/10.1145/2451176.2451201.

[122] Jens Kober and Jan Peters. "Imitation and reinforcement learning". In: *IEEE Robotics & Automation Magazine* 17.2 (2010), pp. 55–62.

[123] George Konidaris et al. "Constructing Skill Trees for Reinforcement Learning Agents from Demonstration Trajectories". In: *Advances in Neural Information Processing Systems*. Ed. by J. Lafferty et al. Vol. 23. Curran Associates, Inc., 2010.

[124] Konstantinos I. Kotis and Artem Katasonov. "An ontology for the automated deployment of applications in heterogeneous IoT environments 1". In: 2012.

[125] SB Kotsiantis and PE Pintelas. "Mixture of expert agents for handling imbalanced data sets". In: *Annals of Mathematics, Computing & Teleinformatics* 1.1 (2003), pp. 46–55.

[126] Sergey V. Kovalchuk et al. "Simulation of patient flow in multiple healthcare units using process and data mining techniques for model identification". In: *Journal of Biomedical Informatics* 82 (2018), pp. 128–142. ISSN: 1532-0464. DOI: https://doi.org/10.1016/j.jbi.2018.05.004. URL: https://www.sciencedirect.com/science/article/pii/S153204641830087X.

[127] Heinrich Küttler et al. "The nethack learning environment". In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 7671–7684.

[128] Yannis Labrou and Tim Finin. "A Semantics Approach for KQML—a General Purpose Communication Language for Software Agents". In: *Proceedings of the Third International Conference on Information and Knowledge Management*. CIKM '94. Gaithersburg, Maryland, USA: Association for Computing Machinery, 1994, pp. 447–455. ISBN: 0897916743. DOI: 10.1145/191246.191320. URL: https://doi.org/10.1145/191246.191320.

[129] Xuan Nhat Lam et al. "Addressing cold-start problem in recommendation systems". In: *Proceedings of the 2nd international conference on Ubiquitous information management and communication*. 2008, pp. 208–211.

[130] Hoang Le et al. "Hierarchical imitation and reinforcement learning". In: *International conference on machine learning*. PMLR. 2018, pp. 2917–2926.

[131] David B. Leake, Andrew Kinley, and David C. Wilson. "Case-Based Similarity Assessment: Estimating Adaptability from Experience". In: *AAAI/IAAI*. 1997.

[132] Sander J. J. Leemans, Dirk Fahland, and Wil M. P. van der Aalst. "Discovering Block-Structured Process Models from Event Logs - A Constructive Approach". In: *Petri Nets*. 2013.

[133] Francesco Leotta et al. "Pipelining user trajectory analysis and visual process maps for habit mining". In: *2017 IEEE SmartWorld, Ubiquitous Intelligence Computing, Advanced Trusted Computed, Scalable Computing Communications, Cloud Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI)*. 2017, pp. 1–8. DOI: 10.1109/UIC-ATC.2017.8397509.

[134] Jure Leskovec, Anand Rajaraman, and Jeffrey David Ullman. *Mining of Massive Datasets*. 2nd. USA: Cambridge University Press, 2014. ISBN: 1107077230.

[135] Benjamin Letham et al. "Interpretable classifiers using rules and Bayesian analysis: Building a better stroke prediction model". In: *The Annals of Applied Statistics* 9.3 (2015), pp. 1350–1371.

[136] Shuang Li et al. "Pre-trained language models for interactive decision-making". In: *arXiv preprint arXiv:2202.01771* (2022).

[137] T. Warren Liao, Zhiming Zhang, and Claude R. Mount. "Similarity measures for retrieval in case-based reasoning systems". In: *Applied Artificial Intelligence* 12.4 (1998), pp. 267–288. DOI: 10.1080/088395198117730. eprint: https://doi.org/10.1080/088395198117730. URL: https://doi.org/10.1080/088395198117730.

[138] Yuan-Hong Liao et al. "Synthesizing Environment-Aware Activities via Activity Sketches". In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2019.

[139] Petro Liashchynskyi and Pavlo Liashchynskyi. "Grid search, random search, genetic algorithm: a big comparison for NAS". In: *arXiv preprint arXiv:1912.06059* (2019).

[140] Blerina Lika, Kostas Kolomvatsos, and Stathes Hadjiefthymiades. "Facing the cold start problem in recommender systems". In: *Expert Systems with Applications* 41.4 (2014), pp. 2065–2073.

[141] Kuo-Sui Lin. "A case-based reasoning system for interior design using a new cosine similarity retrieval algorithm". In: *Journal of Information and Telecommunication* 4.1 (2020), pp. 91–104.

[142] Ying Liu et al. "Deep Reinforcement Learning for Dynamic Treatment Regimes on Medical Registry Data". In: *2017 IEEE International Conference on Healthcare Informatics (ICHI)*. 2017, pp. 380–385. DOI: 10.1109/ICHI.2017.45.

[143] Kun Lu et al. "Reinforcement Learning-powered Semantic Communication via Semantic Similarity". In: *ArXiv* abs/2108.12121 (2021).

[144] Jonathan M Mortensen, Mark Musen, and Natasha Noy. "Crowdsourcing the Verification of Relationships in Biomedical Ontologies". In: 2013 (Nov. 2013), pp. 1020–9.

[145] Laurens van der Maaten and Geoffrey Hinton. "Visualizing Data using t-SNE". In: *Journal of Machine Learning Research* 9 (2008), pp. 2579–2605. URL: http://www.jmlr.org/papers/v9/vandermaaten08a.html.

[146] F. Mannhardt et al. “Guided process discovery: a pattern-based approach”. English. In: *Information Systems* 76 (July 2018), pp. 1–18. ISSN: 0306-4379. DOI: 10.1016/j.is.2018.01.009.

[147] Gunjan Mansingh, Kweku-Muata Osei-Bryson, and Han Reichgelt. “Using ontologies to facilitate post-processing of association rules by domain experts”. In: *Information Sciences* 181.3 (2011), pp. 419–434. ISSN: 0020-0255. DOI: https://doi.org/10.1016/j.ins.2010.09.027. URL: https://www.sciencedirect.com/science/article/pii/S0020025510004810.

[148] Michael Margaliot. “Mathematical Modeling of Natural Phenomena: A Fuzzy Logic Approach”. In: *Fuzzy Logic: A Spectrum of Theoretical & Practical Issues*. Ed. by Paul P. Wang, Da Ruan, and Etienne E. Kerre. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 113–134. ISBN: 978-3-540-71258-9. DOI: 10.1007/978-3-540-71258-9_7. URL: https://doi.org/10.1007/978-3-540-71258-9_7.

[149] Leland McInnes, John Healy, and James Melville. *UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction*. 2018. arXiv: 1802.03426 [stat.ML].

[150] Scott W. McQuiggan and James C. Lester. “Learning Empathy: A Data-Driven Framework for Modeling Empathetic Companion Agents”. In: *Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multiagent Systems*. AAMAS ’06. Hakodate, Japan: Association for Computing Machinery, 2006, pp. 961–968. ISBN: 1595933034. DOI: 10.1145/1160633.1160806. URL: https://doi.org/10.1145/1160633.1160806.

[151] Marjorie McShane, Sergei Nirenburg, and Bruce Jarrell. “Modeling decision-making biases”. In: *Biologically Inspired Cognitive Architectures* 3 (2013), pp. 39–50. ISSN: 2212-683X. DOI: https://doi.org/10.1016/j.bica.2012.09.001. URL: https://www.sciencedirect.com/science/article/pii/S2212683X12000515.

[152] Nicole Merkle and Ralf Mikut. “Context-Aware Composition of Agent Policies by Markov Decision Process Entity Embeddings and Agent Ensembles”. In: *Semantic Web Journal (SWJ)* (2023). URL: https://semantic-web-journal.net/content/context-aware-composition-agent-policies-markov-decision-process-entity-embeddings-and-0.

[153] Nicole Merkle and Ralf Mikut. *Context-Aware Composition of Agent Policies by Markov Decision Process Entity Embeddings and Agent Ensembles*. 2023. arXiv: 2308.14521 [cs.AI].

[154] Nicole Merkle and Ralf Mikut. *Sources and Data of Submitted Semantic Web Journal Paper:"Context-Aware Composition of Agent Policies by Markov Decision Process Entity Embeddings and Agent Ensembles"*. Nov. 2023. DOI: 10.6084/m9.figshare.22215079. URL: https://figshare.com/articles/journal_contribution/Sources_and_Data_of_Submitted_Semantic_Web_Journal_Paper_Context-Aware_Composition_of_Agent_Policies_by_Markov_Decision_Process_Entity_Embeddings_and_Agent_Ensembles_/22215079/1.

[155] Nicole Merkle and Patrick Philipp. "Cooperative Web Agents by Combining Semantic Technologies with Reinforcement Learning". In: *Proceedings of the 10th International Conference on Knowledge Capture, K-CAP 2019, Marina Del Rey, CA, USA, November 19-21, 2019*. Ed. by Mayank Kejriwal, Pedro A. Szekely, and Raphaël Troncy. ACM, 2019, pp. 205–212.

[156] Nicole Merkle and Stefan Zander. "Agent-Based Assistance in Ambient Assisted Living Through Reinforcement Learning and Semantic Technologies - (Short Paper)". In: *On the Move to Meaningful Internet Systems. OTM 2017 Conferences - Confederated International Conferences: CoopIS, C&TC, and ODBASE 2017, Rhodes, Greece, October 23-27, 2017, Proceedings, Part II*. 2017, pp. 180–188. DOI: 10.1007/978-3-319-69459-7_12. URL: https://doi.org/10.1007/978-3-319-69459-7_12.

[157] Nicole Merkle and Stefan Zander. "Improving the Utilization of AAL Devices through Semantic Web Technologies and Web of Things Concepts". In: *Procedia Computer Science* 98.Supplement C (2016). The 7th International Conference on Emerging Ubiquitous Systems and Pervasive Networks (EUSPN 2016)/The 6th International Conference on Current and Future Trends of Information and Communication Technologies in Healthcare (ICTH-2016)/Affiliated Workshops, pp. 290–297. ISSN: 1877-0509. DOI: https://doi.org/10.1016/j.procs.2016.09.045. URL: http://www.sciencedirect.com/science/article/pii/S1877050916321779.

[158] Nicole Merkle and Stefan Zander. "Using a Semantic Simulation Framework for Teaching Machine Learning Agents". In: *Procedia Computer Science* 137 (2018). Proceedings of the 14th International Conference on Semantic Systems 10th, 13th of September 2018 Vienna, Austria, pp. 78–89. ISSN: 1877-0509.

[159] Nicole Merkle, Stefan Zander, and Viliam Simko. "A Semantic Use Case Simulation Framework for Training Machine Learning Algorithms". In: *Knowledge Engineering and Knowledge Management*. Ed. by Catherine Faron Zucker et al. Cham: Springer International Publishing, 2018, pp. 243–257.

[160] Mohammed Mesabbah, Waleed Abo-Hamad, and Susan McKeever. "A Hybrid Process Mining Framework for Automated Simulation Modelling for Healthcare". In: *2019 Winter Simulation Conference (WSC)*. 2019, pp. 1094–1102. DOI: 10.1109/WSC40007.2019.9004800.

[161] Mohammed Mesabbah and Susan McKeever. "Presenting a Hybrid Processing Mining Framework for Automated Simulation Model Generation". In: *2018 Winter Simulation Conference (WSC)*. 2018, pp. 1370–1381. DOI: 10.1109/WSC.2018.8632467.

[162] Zbigniew Michalewicz. "GAs: Why Do They Work?" In: *Genetic Algorithms+ Data Structures= Evolution Programs*. Springer, 1996, pp. 45–55.

[163] Tomas Mikolov et al. *Distributed Representations of Words and Phrases and their Compositionality*. 2013. arXiv: 1310.4546 [cs.CL].

[164] Tomas Mikolov et al. *Efficient Estimation of Word Representations in Vector Space*. 2013. arXiv: 1301.3781 [cs.CL].

[165] ALLEN E. MILEWSKI and STEVEN H. LEWIS. “Delegating to software agents”. In: *International Journal of Human-Computer Studies* 46.4 (1997), pp. 485–500. ISSN: 1071-5819. DOI: https://doi.org/10.1006/ijhc.1996.0100. URL: https://www.sciencedirect.com/science/article/pii/S1071581996901007.

[166] Marvin Minsky. “Steps toward artificial intelligence”. In: *Computers and Thought*. McGraw-Hill, 1961, pp. 406–450.

[167] Luis Miralles-Pechuán et al. “A Methodology Based on Deep Q-Learning/Genetic Algorithms for Optimizing COVID-19 Pandemic Government Actions”. In: *Proceedings of the 29th ACM International Conference on Information and Knowledge Management*. CIKM 20. Virtual Event, Ireland: Association for Computing Machinery, 2020, pp. 1135–1144. ISBN: 9781450368599.

[168] Luis Miralles-Pechuán et al. “A Methodology Based on Deep Q-Learning/Genetic Algorithms for Optimizing COVID-19 Pandemic Government Actions”. In: *Proceedings of the 29th ACM International Conference on Information and Knowledge Management*. CIKM ’20. Virtual Event, Ireland: Association for Computing Machinery, 2020, pp. 1135–1144. ISBN: 9781450368599. DOI: 10.1145/3340531.3412179. URL: https://doi.org/10.1145/3340531.3412179.

[169] T Mitchell et al. “Machine Learning”. In: *Annual Review of Computer Science* 4.1 (1990), pp. 417–433. DOI: 10.1146/annurev.cs.04.060190.002221. eprint: https://doi.org/10.1146/annurev.cs.04.060190.002221. URL: https://doi.org/10.1146/annurev.cs.04.060190.002221.

[170] Volodymyr Mnih et al. “Human-level control through deep reinforcement learning”. In: 518.7540 (Feb. 2015), pp. 529–533.

[171] Volodymyr Mnih et al. “Playing Atari with Deep Reinforcement Learning”. In: *CoRR* arXiv:cs/0204012 (2013). eprint: 1312.5602.

[172] Volodymyr Mnih et al. “Playing Atari with Deep Reinforcement Learning”. In: *CoRR* abs/1312.5602 (2013). arXiv: 1312.5602. URL: http://arxiv.org/abs/1312.5602.

[173] Christoph Molnar. *Interpretable Machine Learning*. 2018. URL: https://christophm.github.io/interpretable-ml-book.

[174] László Monostori. “Cyber-Physical Systems”. In: *CIRP Encyclopedia of Production Engineering*. Ed. by Sami Chatti and Tullio Tolio. Berlin, Heidelberg: Springer Berlin Heidelberg, 2018, pp. 1–8. ISBN: 978-3-642-35950-7. DOI: 10.1007/978-3-642-35950-7_16790-1. URL: https://doi.org/10.1007/978-3-642-35950-7_16790-1.

[175] Stefania Montani. “How to use contextual knowledge in medical case-based reasoning systems: A survey on very recent trends”. In: *Artificial Intelligence in Medicine* 51.2 (2011). Advances in Case-Based Reasoning in the Health Sciences, pp. 125–131. ISSN: 0933-3657.

[176] Richard D. Morey et al. “The fallacy of placing confidence in confidence intervals”. en. In: *Psychonomic Bulletin & Review* 23.1 (Feb. 2016), pp. 103–123. ISSN: 1069-9384, 1531-5320. DOI: 10.3758/s13423-015-0947-8. URL: http://link.springer.com/10.3758/s13423-015-0947-8 (visited on 04/14/2016).

[177] Eduardo Mosqueira-Rey et al. "Human-in-the-Loop Machine Learning: A State of the Art". In: *Artif. Intell. Rev.* 56.4 (Aug. 2022), pp. 3005–3054. ISSN: 0269-2821. DOI: 10.1007/s10462-022-10246-w. URL: https://doi.org/10.1007/s10462-022-10246-w.

[178] Senthilselvan Natarajan et al. "Resolving data sparsity and cold start problem in collaborative filtering recommender system using linked open data". In: *Expert Systems with Applications* 149 (2020), p. 113248.

[179] Jerzy Neyman. "Outline of a theory of statistical estimation based on the classical theory of probability". In: *Philosophical Transactions of the Royal Society of London. Series A, Mathematical and Physical Sciences* 236.767 (1937), pp. 333–380.

[180] Muaz A. Niazi and Amir Hussain. "A Novel Agent-Based Simulation Framework for Sensing in Complex Adaptive Environments". In: *IEEE Sensors Journal* 11.2 (2011), pp. 404–412. DOI: 10.1109/JSEN.2010.2068044.

[181] Nils J. Nilsson. *Introduction to Machine Learning: An Early Draft of a Proposed Textbook. Pages 175-188. http://robotics.stanford.edu/people/nilsson/mlbook.ht ml.* 1996.

[182] Farzad Niroui et al. "Deep Reinforcement Learning Robot for Search and Rescue Applications: Exploration in Unknown Cluttered Environments". In: *IEEE Robotics and Automation Letters* 4.2 (2019), pp. 610–617. DOI: 10.1109/LRA.2019.2891991.

[183] Brian Keith Norambuena. "Integration of Process Mining and Simulation: A Survey of Applications and Current Research". In: *WorldCIST*. 2018.

[184] Héctor Núñez et al. "A comparative study on the use of similarity measures in case-based reasoning to improve the classification of environmental system situations". In: *Environmental Modelling & Software* 19.9 (2004). Environmental Sciences and Artificial Intelligence, pp. 809–819. ISSN: 1364-8152. DOI: https://doi.org/10.1016/j.envsoft.2003.03.003. URL: https://www.sciencedirect.com/science/article/pii/S1364815203002020.

[185] Laurent Orseau, Tor Lattimore, and Marcus Hutter. "Universal Knowledge-Seeking Agents for Stochastic Environments". In: *Algorithmic Learning Theory*. Ed. by Sanjay Jain et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 158–172. ISBN: 978-3-642-40935-6.

[186] M. A. Osborne et al. "Towards Real-Time Information Processing of Sensor Network Data Using Computationally Efficient Multi-output Gaussian Processes". In: *2008 International Conference on Information Processing in Sensor Networks (ipsn 2008)*. 2008, pp. 109–120. DOI: 10.1109/IPSN.2008.25.

[187] D. Osinga. *Deep Learning Cookbook: Practical Recipes to Get Started Quickly*. O'Reilly Media, 2018. ISBN: 9781491995792. URL: https://books.google.de/books?id=TMFeDwAAQBAJ.

[188] Lin Padgham and Patrick Lambrix. "Agent capabilities: Extending BDI theory". In: *AAAI/IAAI*. 2000, pp. 68–73.

[189] Constantin-Valentin Pal and Florin Leon. "A Modified I2A Agent for Learning in a Stochastic Environment". In: *Computational Collective Intelligence*. Ed. by Ngoc Thanh Nguyen et al. Cham: Springer International Publishing, 2020, pp. 388–399. ISBN: 978-3-030-63007-2.

[190] Ashutosh Pandey et al. "Hybrid Planning for Decision Making in Self-Adaptive Systems". In: *2016 IEEE 10th International Conference on Self-Adaptive and Self-Organizing Systems (SASO)*. 2016, pp. 130–139. DOI: 10.1109/SASO.2016.19.

[191] Jung Yeon Park et al. "An explanatory item response theory method for alleviating the cold-start problem in adaptive learning environments". In: vol. 51. 2. 2019, pp. 895–909. DOI: 10.3758/s13428-018-1166-9. URL: https://doi.org/10.3758/s13428-018-1166-9.

[192] Emanuel Parzen. "On Estimation of a Probability Density Function and Mode". In: *The Annals of Mathematical Statistics* 33.3 (1962), pp. 1065–1076. DOI: 10.1214/aoms/1177704472. URL: https://doi.org/10.1214/aoms/1177704472.

[193] Pedro Patron, David M. Lane, and Yvan R. Petillot. "Interoperability of agent capabilities for autonomous knowledge acquisition and decision making in unmanned platforms". In: *OCEANS 2009-EUROPE*. 2009, pp. 1–8. DOI: 10.1109/OCEANSE.2009.5278269.

[194] L. Penserini et al. "Socially-based design meets agent capabilities". In: *Proceedings. IEEE/WIC/ACM International Conference on Intelligent Agent Technology, 2004. (IAT 2004).* 2004, pp. 72–78. DOI: 10.1109/IAT.2004.1342926.

[195] Loris Penserini et al. "From stakeholder intentions to agent capabilities". In: *Sixth International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS'07), Haway, USA*. 2007.

[196] Damien Poirier, Francoise Fessant, and Isabelle Tellier. "Reducing the Cold-Start Problem in Content Recommendation through Opinion Classification". In: *2010 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology*. Vol. 1. 2010, pp. 204–207. DOI: 10.1109/WI-IAT.2010.87.

[197] Artem Polyvyanyy et al. "Goal Recognition Using Off-The-Shelf Process Mining Techniques". In: *Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems*. AAMAS '20. Auckland, New Zealand: International Foundation for Autonomous Agents and Multiagent Systems, 2020, pp. 1072–1080. ISBN: 9781450375184.

[198] David L. Poole and Alan K. Mackworth. *Artificial Intelligence: Foundations of Computational Agents*. 2nd ed. Cambridge University Press, 2017. DOI: 10.1017/9781108164085.

[199] Chris Porhet et al. "Mining a Multimodal Corpus of Doctor's Training for Virtual Patient's Feedbacks". In: *Proceedings of the 19th ACM International Conference on Multimodal Interaction*. ICMI '17. Glasgow, UK: Association for Computing Machinery, 2017, pp. 473–478. ISBN: 9781450355438. DOI: 10.1145/3136755.3136816. URL: https://doi.org/10.1145/3136755.3136816.

[200] Stefan Poslad. "Specifying Protocols for Multi-Agent Systems Interaction". In: *ACM Trans. Auton. Adapt. Syst.* 2.4 (Nov. 2007), 15–es. ISSN: 1556-4665. DOI: 10.1145/1293731.1293735. URL: https://doi.org/10.1145/1293731.1293735.

[201] Bob Price and Craig Boutilier. "Accelerating reinforcement learning through implicit imitation". In: *Journal of Artificial Intelligence Research* 19 (2003), pp. 569–629.

[202] Xavier Puig et al. "Virtualhome: Simulating household activities via programs". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 8494–8502.

[203] Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley Series in Probability and Statistics. Wiley, 1994. ISBN: 978-0-47161977-2. DOI: 10.1002/9780470316887. URL: https://doi.org/10.1002/9780470316887.

[204] Runtao Qiao, Shuhan Yan, and Beijun Shen. "A Reinforcement Learning Solution to Cold-Start Problem in Software Crowdsourcing Recommendations". In: *2018 IEEE International Conference on Progress in Informatics and Computing (PIC)*. 2018, pp. 8–14. DOI: 10.1109/PIC.2018.8706279.

[205] L. Rabiner and B. Juang. "An introduction to hidden Markov models". In: *IEEE ASSP Magazine* 3.1 (1986), pp. 4–16. DOI: 10.1109/MASSP.1986.1165342.

[206] L.R. Rabiner. "A tutorial on hidden Markov models and selected applications in speech recognition". In: *Proceedings of the IEEE* 77.2 (1989), pp. 257–286. DOI: 10.1109/5.18626.

[207] Steven J. Rennie et al. *Self-critical Sequence Training for Image Captioning*. 2017. arXiv: 1612.00563 [cs.LG].

[208] Jeff Rickel and W Lewis Johnson. "Integrating pedagogical capabilities in a virtual environment agent". In: *Proceedings of the first international conference on Autonomous agents*. 1997, pp. 30–38.

[209] Juan Jesús Roldán et al. "Analyzing and improving multi-robot missions by using process mining". In: *Autonomous Robots* 42 (2018), pp. 1187–1205.

[210] Oscar J. Romero. *Architectural Middleware that Supports Building High-performance, Scalable, Ubiquitous, Intelligent Personal Assistants*. 2019. DOI: 10.48550/ARXIV.1906.02068. URL: https://arxiv.org/abs/1906.02068.

[211] Murray Rosenblatt. "Remarks on Some Nonparametric Estimates of a Density Function". In: *The Annals of Mathematical Statistics* 27.3 (1956), pp. 832–837. DOI: 10.1214/aoms/1177728190. URL: https://doi.org/10.1214/aoms/1177728190.

[212] Neil Rubens et al. "Active Learning in Recommender Systems". In: *Recommender Systems Handbook*. Boston, MA: Springer US, 2015, pp. 809–846. ISBN: 978-1-4899-7637-6.

[213] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Third. Series in Artificial Intelligence. Upper Saddle River, NJ: Prentice Hall, 2010. URL: http://aima.cs.berkeley.edu/.

[214] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. 3rd ed. Prentice Hall, 2010.

[215] Fereshteh Sadeghi and Sergey Levine. *CAD2RL: Real Single-Image Flight without a Single Real Image*. 2017. arXiv: 1611.04201 [cs.LG].

[216] Shaker H El-Sappagh and Mohammed Elmogy. "Case based reasoning: Case representation methodologies". In: *International Journal of Advanced Computer Science and Applications* 6.11 (2015), pp. 192–208.

[217] Iqbal H Sarker. "Context-aware rule learning from smartphone data: survey, challenges and future directions". In: *Journal of Big Data* 6.1 (2019), pp. 1–25.

[218] Iqbal H. Sarker and Flora D. Salim. *Mining User Behavioural Rules from Smartphone Data through Association Analysis*. 2018. arXiv: 1804.01379 [cs.DB].

[219] Stefan Schaal et al. "Learning from demonstration". In: *Advances in neural information processing systems* (1997), pp. 1040–1046.

[220] Mike Schaekermann et al. "Resolvable vs. Irresolvable Disagreement: A Study on Worker Deliberation in Crowd Work". In: *Proceedings of the 2018 ACM Conference on Computer Supported Cooperative Work and Social Computing (CSCW 18)*. Best Paper Award. New York City, NY, 2018. DOI: 10.1145/3274423.

[221] Daniela Schmidt et al. "An Ontology for Collaborative Tasks in Multi-agent Systems." In: *Ontobras*. 2015.

[222] Burr Settles. *Active Learning Literature Survey*. Computer Sciences Technical Report 1648. University of Wisconsin–Madison, 2010. URL: http://axon.cs.byu.edu/~martinez/classes/778/Papers/settles.activelearning.pdf.

[223] Burr Settles. "Active learning literature survey". In: (2009).

[224] Nava A. Shaked. "Avatars and virtual agents – relationship interfaces for the elderly". In: *Healthcare Technology Letters* 4.3 (2017), pp. 83–87. DOI: https://doi.org/10.1049/htl.2017.0009. eprint: https://ietresearch.onlinelibrary.wiley.com/doi/pdf/10.1049/htl.2017.0009. URL: https://ietresearch.onlinelibrary.wiley.com/doi/abs/10.1049/htl.2017.0009.

[225] Bracha Shapira. "Recommender Systems Handbook". In: *Springer US*. 2010.

[226] B. H Shekar and Guesh Dagnew. "Grid Search-Based Hyperparameter Tuning and Classification of Microarray Cancer Data". In: *2019 Second International Conference on Advanced Computational and Communication Paradigms (ICACCP)*. 2019, pp. 1–8. DOI: 10.1109/ICACCP.2019.8882943.

[227] Haobin Shi, Mingjun Xin, and Wenjie Dong. "A Kind of Case Similarity Evaluation Model Based on Case-Based Reasoning". In: *2011 International Conference on Internet of Things and 4th International Conference on Cyber, Physical and Social Computing*. 2011, pp. 453–457. DOI: 10.1109/iThings/CPSCom.2011.57.

[228] Tianlin Shi et al. "World of Bits: An Open-Domain Platform for Web-Based Agents". In: *Proceedings of the 34th International Conference on Machine Learning*. Ed. by Doina Precup and Yee Whye Teh. Vol. 70. Proceedings of Machine Learning Research. International Convention Centre, Sydney, Australia: PMLR, June 2017, pp. 3135–3144. URL: http://proceedings.mlr.press/v70/shi17a.html.

[229] Felipe Leno Da Silva et al. "Agents Teaching Agents: A Survey on Inter-Agent Transfer Learning". In: *Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems*. AAMAS '20. Auckland, New Zealand: International Foundation for Autonomous Agents and Multiagent Systems, 2020, pp. 2165–2167. ISBN: 9781450375184.

[230] David Silver et al. "Mastering the game of Go with deep neural networks and tree search". In: *Nature* 529.7587 (2016), pp. 484–489. DOI: 10.1038/nature16961. URL: https://doi.org/10.1038/nature16961.

[231] B. W. Silverman. *Density Estimation for Statistics and Data Analysis*. London: Chapman & Hall, 1986.

[232] *SMAPPI-Smart-Apple-Picker-Agent*. https://github.com/nmerkle/SMAPPI-Smart-Apple-Picker-Agent/https://nmerkle.github.io. Accessed: 2023-04-23.

[233] Internet Society. *The Internet of Things: An Overview - Understanding the Issues and Challenges of a more connected World*. Oct. 2015. URL: https://www.internetsociety.org/resources/doc/2015/iot-overview/ (visited on 05/31/2023).

[234] Mohammad Soleymannejad and Alireza Basiri. "Using OWA Approach to Solve Cold-Start Problem of Recommender Systems". In: *2020 10th International Conference on Computer and Knowledge Engineering (ICCKE)*. 2020, pp. 500–507. DOI: 10.1109/ICCKE50421.2020.9303692.

[235] Stuart Soltysiak et al. "Knowing me, knowing you: practical issues in the personalisation of agent technology". In: *Proceedings of the third international conference on the practical applications of intelligent agents and multi-agent technology*. Citeseer. 1998, pp. 467–484.

[236] Mark Stamp. "A revealing introduction to hidden markov models". In: *Science* (Jan. 2004), pp. 1–20.

[237] P. Stingl. *Mathematik für Fachhochschulen: Technik und Informatik ; mit über 1000 Aufgaben und Lösungen*. Hanser, 2009. ISBN: 9783446420656. URL: https://books.google.de/books?id=O%5C_GhQQAACAAJ.

[238] S. Strogatz. *Infinite Powers: The Story of Calculus - the Language of the Universe*. Atlantic Books, 2019. ISBN: 9781786492944. URL: https://books.google.de/books?id=fi7tvQEACAAJ.

[239] Halit Bener Suay et al. "Learning from demonstration for shaping through inverse reinforcement learning". In: *Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems*. 2016, pp. 429–437.

[240] Jennifer J. Sun et al. “Task Programming: Learning Data Efficient Behaviour Representations”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2021, pp. 2876–2885.

[241] Yunsick Sung and Kyungeun Cho. “Collaborative programming by demonstration in a virtual environment”. In: *IEEE Intelligent Systems* 27.2 (2012), pp. 14–17.

[242] Richard S. Sutton and Andrew G. Barto. *Introduction to Reinforcement Learning*. 1st. Cambridge, MA, USA: MIT Press, 1998. ISBN: 0262193981.

[243] Valentina Tamma et al. *Ontologies for Agents: Theory and Experiences*. Jan. 2005. ISBN: 978-3-7643-7237-8. DOI: 10.1007/3-7643-7361-X.

[244] Matthew E Taylor, Halit Bener Suay, and Sonia Chernova. “Integrating reinforcement learning with human demonstrations of varying ability”. In: *The 10th International Conference on Autonomous Agents and Multiagent Systems-Volume 2*. Citeseer. 2011, pp. 617–624.

[245] Open Ended Learning Team et al. “Open-ended learning leads to generally capable agents”. In: *arXiv preprint arXiv:2107.12808* (2021).

[246] Luis Orlindo Tedeschi. “Assessment of the adequacy of mathematical models”. In: *Agricultural Systems* 89.2 (2006), pp. 225–247. ISSN: 0308-521X. DOI: https://doi.org/10.1016/j.agsy.2005.11.004. URL: https://www.sciencedirect.com/science/article/pii/S0308521X05002568.

[247] F. Tempola, A. Arief, and M. Muhammad. “Combination of case-based reasoning and nearest neighbour for recommendation of volcano status”. In: *2017 2nd International conferences on Information Technology, Information Systems and Electrical Engineering (ICITISEE)*. 2017, pp. 348–352.

[248] STEPHEN TENCH, HANNAH FRY, and PAUL GILL. “Spatio-temporal patterns of IED usage by the Provisional Irish Republican Army”. In: *European Journal of Applied Mathematics* 27.3 (2016), pp. 377–402. DOI: 10.1017/S0956792515000686.

[249] Georgios Theocharous et al. “Reinforcement Learning for Strategic Recommendations”. In: *ArXiv* abs/2009.07346 (2020).

[250] Andrea L. Thomaz, Guy Hoffman, and Cynthia Breazeal. “Reinforcement Learning with Human Teachers: Understanding How People Want to Teach Robots”. In: *ROMAN 2006 - The 15th IEEE International Symposium on Robot and Human Interactive Communication*. 2006, pp. 352–357. DOI: 10.1109/ROMAN.2006.314459.

[251] S.P. Thompson and M. Gardner. *Calculus Made Easy*. St. Martin’s Publishing Group, 1998. ISBN: 9780312185480. URL: https://books.google.de/books?id=BBIFtidWdUC.

[252] Sebastian Thrun. “Exploration in active learning”. In: *Handbook of Brain Science and Neural Networks* (1995), pp. 381–384.

[253] Sebastian Thrun and Lorien Pratt, eds. *Learning to Learn*. USA: Kluwer Academic Publishers, 1998. ISBN: 0792380479.

[254] Ling Tian et al. "Knowledge graph and knowledge reasoning: A systematic review". In: *Journal of Electronic Science and Technology* 20.2 (2022), p. 100159. ISSN: 1674-862X. DOI: https://doi.org/10.1016/j.jnlest.2022.100159. URL: https://www.sciencedirect.com/science/article/pii/S1674862X2200012X.

[255] Josh Tobin et al. *Domain Randomization for Transferring Deep Neural Networks from Simulation to the Real World*. 2017. arXiv: 1703.06907 [cs.RO].

[256] Andrei Tour, Artem Polyvyanyy, and Anna Kalenkova. "Agent System Mining: Vision, Benefits, and Challenges". In: *IEEE Access* 9 (2021), pp. 99480–99494. DOI: 10.1109/ACCESS.2021.3095464.

[257] Allan Tucker et al. "Generating high-fidelity synthetic patient data for assessing machine learning healthcare software". In: *NPJ digital medicine* 3.1 (2020), pp. 1–13.

[258] Amos Tversky and Daniel Kahneman. "Judgment under Uncertainty: Heuristics and Biases". In: *Science* 185.4157 (1974), pp. 1124–1131. DOI: 10.1126/science.185.4157.1124. eprint: https://www.science.org/doi/pdf/10.1126/science.185.4157.1124. URL: https://www.science.org/doi/abs/10.1126/science.185.4157.1124.

[259] Amos Tversky and Daniel Kahneman. "The Framing of Decisions and the Psychology of Choice". In: *Behavioural Decision Making*. Ed. by George Wright. Boston, MA: Springer US, 1985, pp. 25–41. ISBN: 978-1-4613-2391-4. DOI: 10.1007/978-1-4613-2391-4_2. URL: https://doi.org/10.1007/978-1-4613-2391-4_2.

[260] Rainer Unland. "Chapter 1 - Software Agent Systems". In: *Industrial Agents*. Ed. by Paulo Leitão and Stamatis Karnouskos. Boston: Morgan Kaufmann, 2015, pp. 3–22. ISBN: 978-0-12-800341-1. DOI: https://doi.org/10.1016/B978-0-12-800341-1.00001-2. URL: https://www.sciencedirect.com/science/article/pii/B9780128003411000012.

[261] Frank Van Harmelen, Vladimir Lifschitz, and Bruce Porter. *Handbook of knowledge representation*. Elsevier, 2008.

[262] Ashish Vaswani et al. "Attention is All you Need". In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon et al. Vol. 30. Curran Associates, Inc., 2017.

[263] John Venn. *The Logic of Chance, 3rd edition An Essay on the Foundations and Provice of the Theory of Probability*. 3rd. The Guttenberg Project, 2018. URL: https://www.gutenberg.org/files/57359/57359-h/57359-h.htm.

[264] Ayushi Verma and Sushil Kumar. "Cognitive Robotics in Artificial Intelligence". In: *2018 8th International Conference on Cloud Computing, Data Science & Engineering (Confluence)*. 2018, pp. 65–70. DOI: 10.1109/CONFLUENCE.2018.8442725.

[265] Vijay Verma and Rajesh Kumar Aggarwal. "A comparative analysis of similarity measures akin to the Jaccard index in collaborative recommendations: empirical and theoretical perspective". In: *Soc. Netw. Anal. Min.* 10.1 (2020), p. 43.

[266] Nicolas Verstaevel et al. “Principles and Experimentations of Self-organizing Embedded Agents Allowing Learning from Demonstration in Ambient Robotic”. In: *Procedia Computer Science* 52 (2015). The 6th International Conference on Ambient Systems, Networks and Technologies (ANT-2015), the 5th International Conference on Sustainable Energy Information Technology (SEIT-2015), pp. 194–201. ISSN: 1877-0509. DOI: https://doi.org/10.1016/j.procs.2015.05.056. URL: https://www.sciencedirect.com/science/article/pii/S187705091500856X.

[267] Iosif Viktoratos, Athanasios Tsadiras, and Nick Bassiliades. “Combining Community-Based Knowledge with Association Rule Mining to Alleviate the Cold Start Problem in Context-Aware Recommender Systems”. In: *Expert Syst. Appl.* 101.C (July 2018), pp. 78–90. ISSN: 0957-4174. DOI: 10.1016/j.eswa.2018.01.044. URL: https://doi.org/10.1016/j.eswa.2018.01.044.

[268] Damian G. Walker et al. “Generalisability, Transferability, Complexity and Relevance”. In: *Evidence-Based Decisions and Economics*. John Wiley & Sons, Ltd, 2010. Chap. 5, pp. 56–66. ISBN: 9781444320398. DOI: https://doi.org/10.1002/9781444320398.ch5. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/9781444320398.ch5. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/9781444320398.ch5.

[269] Hongbing Wang et al. “A multi-agent reinforcement learning approach to dynamic service composition”. In: *Information Sciences* 363 (2016), pp. 96–119. ISSN: 0020-0255. DOI: https://doi.org/10.1016/j.ins.2016.05.002. URL: https://www.sciencedirect.com/science/article/pii/S0020025516303085.

[270] Wei Wang et al. “A Comprehensive Ontology for Knowledge Representation in the Internet of Things”. In: *2012 IEEE 11th International Conference on Trust, Security and Privacy in Computing and Communications*. 2012, pp. 1793–1798. DOI: 10.1109/TrustCom.2012.20.

[271] Zhenchen Wang, Puja Myles, and Allan Tucker. “Generating and evaluating cross-sectional synthetic electronic healthcare data: Preserving data utility and patient privacy”. In: *Computational Intelligence* 37.2 (2021), pp. 819–851.

[272] Jia Wangping et al. “Extended SIR Prediction of the Epidemics Trend of COVID-19 in Italy and Compared With Hunan, China”. In: *Frontiers in Medicine* 7 (2020), p. 169. ISSN: 2296-858X. DOI: 10.3389/fmed.2020.00169. URL: https://www.frontiersin.org/article/10.3389/fmed.2020.00169.

[273] Geoffrey I Webb and Mark Kuzmycz. “Feature Based Modelling: A methodology for producing coherent, consistent, dynamically changing models of agents’ competencies”. In: *User modeling and user-adapted interaction* 5.2 (1995), pp. 117–150.

[274] Ben Weber, Michael Mateas, and Arnav Jhala. “Learning from Demonstration for Goal-Driven Autonomy”. In: *Proceedings of the AAAI Conference on Artificial Intelligence* 26.1 (Sept. 2021), pp. 1176–1182. DOI: 10.1609/aaai.v26i1.8311. URL: https://ojs.aaai.org/index.php/AAAI/article/view/8311.

[275] Eric W. Weisstein. *Gaussian Function*. From MathWorld–A Wolfram Web Resource. URL: https://mathworld.wolfram.com/GaussianFunction.html.

[276] Julius Weyl, Daniel Glake, and Thomas Clemen. "Agent-Based Traffic Simulation at City Scale with MARS". In: *Proceedings of the Agent-Directed Simulation Symposium*. ADS '18. Baltimore, Maryland: Society for Computer Simulation International, 2018. ISBN: 9781510860131.

[277] Danny Weyns and Tom Holvoet. "Model for Simultaneous Actions in Situated Multi-agent Systems". In: *Multiagent System Technologies*. Ed. by Michael Schillo et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 105–118. ISBN: 978-3-540-39869-1.

[278] Maggie Wigness, John G. Rogers, and Luis E. Navarro-Serment. "Robot Navigation from Human Demonstration: Learning Control Behaviours". In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. 2018, pp. 1150–1157. DOI: 10.1109/ICRA.2018.8462900.

[279] C. Wohlin et al. *Experimentation in Software Engineering*. Computer Science. Springer Berlin Heidelberg, 2012. ISBN: 9783642290442. URL: https://books.google.de/books?id=QPVsM1%5C_U8nkC.

[280] Yikun Xian et al. "Reinforcement Knowledge Graph Reasoning for Explainable Recommendation". In: *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR'19. Paris, France: Association for Computing Machinery, 2019, pp. 285–294. ISBN: 9781450361729. DOI: 10.1145/3331184.3331203. URL: https://doi.org/10.1145/3331184.3331203.

[281] Yanwei Xing et al. "Combination Data Mining Methods with New Medical Data to Predicting Outcome of Coronary Heart Disease". In: *2007 International Conference on Convergence Information Technology (ICCIT 2007)*. 2007, pp. 868–872. DOI: 10.1109/ICCIT.2007.204.

[282] Ning Xiong. "Learning fuzzy rules for similarity assessment in case-based reasoning". In: *Expert systems with applications* 38.9 (2011), pp. 10780–10786.

[283] R.R. Yager. "On ordered weighted averaging aggregation operators in multicriteria decisionmaking". In: *IEEE Transactions on Systems, Man, and Cybernetics* 18.1 (1988), pp. 183–190. DOI: 10.1109/21.87068.

[284] Yuguang Yang, Michael A. Bevan, and Bo Li. "Efficient Navigation of Colloidal Robots in an Unknown Environment via Deep Reinforcement Learning". In: *Advanced Intelligent Systems* 2.1 (2020), p. 1900106. DOI: https://doi.org/10.1002/aisy.201900106. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/aisy.201900106. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/aisy.201900106.

[285] Juan Ye, Simon Dobson, and Susan McKeever. "Situation identification techniques in pervasive computing: A review". In: *Pervasive and Mobile Computing* 8.1 (2012), pp. 36–66. ISSN: 1574-1192. DOI: https://doi.org/10.1016/j.pmcj.2011.01.004. URL: https://www.sciencedirect.com/science/article/pii/S1574119211000253.

[286] Han Yu, Zhiqi Shen, and Cyril Leung. “From internet of things to internet of agents”. In: *2013 IEEE international conference on green computing and communications and IEEE Internet of Things and IEEE cyber, physical and social computing*. IEEE. 2013, pp. 1054–1057.

[287] Tong Yu and Hong Zhu. “Hyper-Parameter Optimization: A Review of Algorithms and Applications”. In: *ArXiv* abs/2003.05689 (2020).

[288] Qilong Yuan et al. “Flexible telemanipulation based handy robot teaching on tape masking with complex geometry”. In: *Robotics and Computer-Integrated Manufacturing* 66 (2020), p. 101990. ISSN: 0736-5845. DOI: https://doi.org/10.1016/j.rcim.2020.101990. URL: https://www.sciencedirect.com/science/article/pii/S0736584520302015.

[289] Chao Zhang et al. “Using Cognitive Models to Train Warm Start Reinforcement Learning Agents for Human-Computer Interactions”. In: *ArXiv* abs/2103.06160 (2021).

[290] Li Zhang et al. “An agent-based framework for distributed intelligent control systems”. In: *SICE 2004 Annual Conference*. Vol. 3. 2004, 2685–2688 vol. 3.

[291] Yong Zhang and Meng Joo Er. “Sequential active learning using meta-cognitive extreme learning machine”. In: *Neurocomputing* 173 (2016), pp. 835–844.

[292] Yunzhi Zhang, Pieter Abbeel, and Lerrel Pinto. “Automatic curriculum learning through value disagreement”. In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 7648–7659.

[293] Kangzhi Zhao et al. “Leveraging Demonstrations for Reinforcement Recommendation Reasoning over Knowledge Graphs”. In: *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR ’20. Virtual Event, China: Association for Computing Machinery, 2020, pp. 239–248. ISBN: 9781450380164. DOI: 10.1145/3397271.3401171. URL: https://doi.org/10.1145/3397271.3401171.

[294] QinPing Zhao. “Data acquisition and simulation of natural phenomena”. In: *Science China Information Sciences* 54.4 (2011), pp. 683–716.

[295] Fangming Zhu and Sheng-Uei Guan. “Towards evolution of software agents in electronic commerce”. In: *Proceedings of the 2001 Congress on Evolutionary Computation (IEEE Cat. No.01TH8546)*. Vol. 2. 2001, 1303–1308 vol. 2. DOI: 10.1109/CEC.2001.934341.

[296] Roland Zimmermann et al. “An Ontology for Agent-Based Monitoring of Fulfillment Processes”. In: *Ontologies for Agents: Theory and Experiences*. Ed. by Valentina Tamma et al. Basel: Birkhäuser Basel, 2005, pp. 323–345. ISBN: 978-3-7643-7361-0.

A. Reward Function Example

Platform developers who want to integrate new reward functions into the simulation component do not necessarily have to define SPARQL CONSTRUCT queries, but can implement a function as described in this example code.

To calculate the reward, several logical conditions need to be evaluated. In **line 20** it is checked whether additional goals have been achieved compared to the number of goals already achieved, and if so, the reward value is calculated by subtracting the number of goals last observed from the number of goals currently achieved. For example, if 2 goals have been achieved up to the penultimate action and now an additional goal has been achieved after the current action has been performed, this results in a reward value of 1, as only the current action and its goal achievement counts and the last actions and their goal achievement are no longer taken into account.

The example function is implemented as a *closure* [1]. First, the required variables (e.g. number of goal states in the task) are declared in the outer function **(line 2 - 4)**, then an inner function is returned, which is then invoked by the simulator after each action received from the agent **(line 5)**.

The outer function has as input a *task* instance representation, containing all required information, and a *goalSize* that indicates the number of goal states that can be reached within the task execution **(line 1)**. This number is required because the algorithm can then detect how many and whether all goal states were achieved during task execution. Based on this, the function decides which reward value to compute.

The inner function takes over as input parameters the last state, the last executed action of the agent, and the currently executed action of the agent **(line 5)**. Then, in **line 6 - 10**, a reward variable, a flag variable are declared and initialised. The reward variable is the value that is returned by the function while the flag variable indicates whether an action has been performed twice in sequence that led to an already existing state in the declared *stateList*.

The *updateState()* function updates the last state based on the passed action and returns the updated State **(line 8)**. Subsequently, the *reasonState()* function infers the new state based on the updated state representation and the meta information provided by the task

[1] Closures are applied in functional programming and are supported in all functional programming languages such as Python, JavaScript. For more details see `https://developer.mozilla.org/en-US/docs/Web/JavaScript/Closures`

Algorithm 10: ComputeSequentialReward

```
Input: task, goalSize, lastState, action, lastAction
Output: reward
Function ComputeSequentialReward(task, goalSize):
    stateList = []
    numGoals = goalSize
    numLastGoals = 0
    return Function (lastState, action, lastAction):
        reward = -1
        flag = false
        updatedState = updateState(task, lastState, action)
        reasonedStates = reasonState(task, updatedState)
        goalStates = reasonedStates.filter(state => state.isGoal)
        if reasonedStates.length > 0 AND statesList.length < reasonedStates.length
         then
            forall state IN reasonedStates do
                if state NOT IN stateList then
                    stateList.push(state)
                    flag = false
                else
                    flag = true
            if flag OR action == lastAction then
                reward = -1
            else if goalStates.length > numLastGoals then
                reward = goalStates.length - numLastGoals
                numLastGoals = goalStates.length
            else if numLastGoals > goalStates.length then
                reward = (numLastGoals - goalStates.length) * (-1)
                numLastGoals = goalStates.length
            else
                reward = stateList.length * 0.25
            lastAction = action
            if numLastGoals == numGoals then
                numLastGoals = 0
                reward = (stateList.length-1) * 0.25
                stateList.length = 0
        return reward
```

instance representation **(line 9)**. This is necessary because the updated state determines whether an action was useful or not.

The inner function also checks in **line 10** whether goal states prevail among the derived states, since a task can have several goal states, which describe sub-goals of a task and, if achieved, indicate the partial success of the agent behaviour. This is done by traversing a list of derived states using an arrow or lambda function and an iterator variable called *state*.

In **line 11** it is checked whether the reasoner function could derive states and whether the length of the *stateList*, which stores already occurred states of the implicitly given state sequence, has fewer state elements than the *reasonedStates* list. In this way, the function can detect whether the performed action had an impact that lead to additional, i.e. new states. In case the conditions in the *if-clause* are fulfilled, a *for-loop* starts that goes through all reasoned states and checks whether the inferred state already happened and is prevalent in the *stateList* that stores the sequence of passed states **(line 12 - 17)**. If the current state element is not yet in the *stateList*, then it is added to it and the boolean *flag* variable is set to false, otherwise it is set to true. The flag variable is required because it indicates whether a state is reached more than once, since some tasks may require a state to be traversed only once to speed up task execution and avoid redundant steps.

In case the *flag* variable is true or the last executed action is equal to the currently executed action, the reward value is set to -1, because in sequential tasks an action is only allowed once and repetitions of actions are not accepted to prevent the agent from getting stuck in a state **(line 18 - 19)**. However, if multiple repetitions of actions are to be supported, the corresponding action and its subsequent states have to be numbered according to their execution order so that they are unique and distinguishable from each other. In addition, a prerequisite is that the corresponding reward values differ from each other and also specify a sequence that indicates the agent's progress, because the experiments conducted have shown that the agent is better able to learn the different effects of actions within a sequence if the actions and subsequent states are easily distinguishable.

If one of the defined conditions in **line 22 or 25** is met, the variable named *numLastGoals* is set to the number of current goals.

However, it may also happen that an action leads to the reversal of goal states and thus have to be penalised as it happens in **line 23**. The corresponding reward value is then computed by penalising the difference between the previous number of goals and the current number of goals. The difference is multiplied by -1 **(line 24)**.

If none of the conditions are met, the current action can be assumed to have resulted in an intermediate state, since the previous conditions check whether all target states of a task have been reached incrementally or whether actions have been performed redundantly. The corresponding reward value for the intermediate step is then calculated by multiplying the state sequence length by any positive constant greater than 0, e.g. 0.25 **(line 27)**, because the more correct actions performed within the sequence, the more rewards

accumulated by the agent.

The variable with the name *action* is assigned to the variable with the name *lastAction* so that the next time the inner function is called, both variables can again be compared to detect action repetitions **(line 28)**.

When all target states are reached, the variable named *numLastGoals* is set to 0 **(line 30)**, while the array named *stateList* is cleared **(line 32)** and the final reward value is determined by multiplying the number of states in the *stateList* array by an arbitrary constant, e.g. 0.25 **(line 31)**.

The function terminates with the return of the reward value **(line 33)**.

B. State Update

As introduced, the simulator has the task to update a state, i.e. its features, depending on the action the agent performed. For this reason, the simulator implements an update function that changes the feature values based on the performed action's effects. Algorithm 11 depicts the corresponding function.

The input parameters of the function are the task instance, the last captured state, the currently executed action sent by the agent and a dictionary of features referencing different states covering all possible value ranges of the corresponding feature **(line 1)**.

In **line 2** an empty object called *updatedState* is declared. The object is filled with updated feature values in the function call and returned when the function terminates **(line 55)**.

In **line 3** the partial or sub-states for the last state are derived. This is necessary because an update is to be carried out for all prevailing partial states. For this reason, each partial state needs to be considered and changed depending on the effects of the action.

A *for-loop* begins, which runs through all the effects of the current action. For each effect, the corresponding observable features that are affected are determined **(line 5)**.

Then a second *for-loop* starts, which runs through all partial states. In the case that the observable feature is present in the last state and in the partial state, the *if-branch* **(line 7)** is entered and an undefined value variable **(line 8)** is declared.

In **line 9**, the variable named *states* is assigned to an array of states that are affected by the currently considered feature. Then the array is sorted in increasing order, depending on the maximum range value of the feature **(line 10)**. Both the maximum and minimum range values of the feature determine the conditions of the state and are specified in the corresponding state's referenced rule expression.

Subsequently, the currently considered effect's type of impact is assessed **(line 11 - 26)**. As in Sec. 4.2 discussed, seven different types of impact are possible, i.e. INCREASE, DECREASE, CONSTANT, CONVERT, ON, OFF and COMPUTE.

If the impact type is INCREASE, the index of the current partial state is retrieved from the sorted states array **(line 12)**. Based on the index value, the next state is then selected from the sorted state array which is located in the array at the index value (i.e. index +1), since the associated state has a higher maximum feature value **(line 13)**. From the determined subsequent state, a feature value is randomly selected that lies in the range between the

Algorithm 11: UpdateStateToNextState

Input: task, lastState, action, stateRanges
Output: updatedState

```
1  Function updateStateToNextState(task, lastState, action, stateRanges):
2      updatedState = {}
3      partialStates = reasonState(task, lastState)
4      forall effect IN action.effects do
5          oFeat = effect.observationFeature
6          forall state IN partialStates do
7              if oFeat IN lastState AND oFeat in state.features then
8                  value = undefined
9                  states = stateRanges[oFeat].states
10                 states.sortIncreasing((a,b) => a.max - b.max)
11                 if effect.HasImpactType == 'INCREASE' then
12                     index = states.indexOf(state)
13                     nextState = states[index+1]
14                     value = getRandomDouble(nextState.min, nextState.max)
15                 else if effect.HasImpactType == 'DECREASE' then
16                     index = states.indexOf(state)
17                     previousState = states[index-1]
18                     value = getRandomInt(previousState.min, previousState.max)
19                 else if effect.HasImpactType == 'CONVERT' then
20                     value = (lastState[oFeat].value == 0) ? 1 : 0
21                 else if effect.HasImpactType == 'ON' then
22                     value = 1
23                 else if effect.HasImpactType == 'OFF' then
24                     value = 0
25                 else if effect.HasImpactType == 'COMPUTE' then
26                     term = ''
27                     forall (key,value) IN effect.equation.params do
28                         term += 'let ${key} = ${value};'
29                     forall (key,value) IN effect.equation.constants do
30                         term += 'let ${key} = ${value};'
31                     term += effect.equation.expression
32                     value = eval(term)
33                 updatedState[oFeat] = value
34     return updatedState
```

minimum and maximum value of the feature under consideration (line 14).

However, if the impact type is DECREASE, the adjacent state that has a lower maximum feature value is selected and a value is randomly determined that falls between the minimum and maximum values of the feature to be updated **(line 16 - 19)**.

The impact type named CONVERT lets the function convert the current feature value into its opposite. Thus, the feature is a binary or Boolean feature that can have either the value 0 for false or 1 for true **(line 20 - 21)**.

The impact type ON sets a Boolean feature value always to 1 while the type OFF sets the corresponding feature value always to 0 **(line 22 - 25)**.

In cases where the effect type is COMPUTE, a string term is created consisting of parameter and variable declarations and an equation that calculates the new feature value based on the feature value of the last state. The term formed is then passed to an *eval* function that executes the term representing the equation and returns the calculated feature value **(line 26 - 33)**.

The impact type with the name CONSTANT is not checked in the function because a constant feature value, as the name suggests, never changes during programme execution.

All updated feature values are stored in *updatedState* **(line 34)**, an object already declared in **line 2**.

Finally, the object *updatedState* is returned as soon as the function has finished its execution **(line 35)**.

C. Data Preprocessing

The FPGrowth algorithm is applied in order to extract frequent item sets. Algo. 12 shows how FPGrowth is implemented. First, the *preprocessData()* function takes as input parameters the agent's dataset, a support threshold value, that indicates the filter criterion for the samples and a boolean parameter that indicates whether the corresponding task is a sequential one because the function distinguishes between data that is emitted by sequential tasks and data that is emitted by non-sequential tasks **(line 1)**.

Subsequently, various local variables are declared, which serve as data structures for storing the intermediate and final results of the execution steps **(line 2 - 5)**.

The *filter-function* compares the reward value in each data sample with the defined threshold value and selects only those samples that have a reward value greater than the threshold. The selected samples are assigned to the variable named *filteredData* **(line 6)**.

The algorithm then checks whether the task being reproduced by the dataset is a sequential or a non-sequential task **(line 7)**. If it is a sequential task, the algorithm goes in a *for-loop*. **(line 9 - 19)** through all the filtered data samples and, using a lambda or arrow function [1], selects the samples that contain an action specified by the key name 'action' and stores them in the array named *wActions* **(line 12)**.

A second *for-loop* begins, which passes through the array called *wActions* **(line 13 - 16)**. Each action obtained is assigned a consecutive number and the action is stored together with the consecutive number in an object called *data* **(line 16)**. On exiting the second *for-loop*, this data object is stored in an array called *dataSamples* **(line 17)**. In order to avoid redundant data samples, the array named *actionsequences* stores action sequences that are non-identical to each other **(line 18 - 19)**.

After the end of the second *for-loop*, an object called *frequentItems* is created that implements the *FPGrowth* algorithm **(line 20)**. The constructor of the *FPGrowth* object takes the filtered data samples and a factor value that determines the *support* threshold for retrieving frequent item sets (see Eq. 2.42). For example, if the data sample size is 1000 and the support factor is set to 1, this means that the support threshold is 1% of the given sample size, which in this example gives a support value of 10 (see also Sec. 2.4.1). The FPGrowth algorithm is only executed for sequential tasks, containing stages, to determine

[1] see https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions/Arrow_functions

the most frequent actions per stage and their sequence number.

Algorithm 12: PreprocessData - Part 1

```
   Input: dataset, threshold, isSequential
   Output: preprocessedSamples, actionsequences
1  Function PreprocessData(dataset, threshold, isSequential):
2      preprocessedSamples = undefined
3      cleanedData = []
4      frequentSamples = []
5      actionsequences = []
6      filteredData = dataset.filter(a => a.reward > threshold)
7      if isSequential then
8          dataSamples = []
9          forall sample IN filteredData do
10             data = {}
11             order = 0
12             wActions = sample.rules.filter(rule => 'action' in rule)
13             forall rule IN wActions do
14                 action = rule.action
15                 order += 1
16                 data[action] = order
17             dataSamples.push(data)
18             if !(containsKey(data, actionsequences)) then
19                 actionsequences.push(data)
20         frequentItems = FPGrowth(dataSamples, 1)
```

After the frequent item sets have been determined, the programme continues with a *for-loop* that runs through all filtered data samples to remove special characters **(line 28, 37)** in the feature names of the dataset **(line 21 - 40)**. The algorithm again distinguishes between sequential and non-sequential task datasets **(line 23 - 32, 33 - 40)**. In the sequential *if-branch*, it starts a second *for-loop* that runs through all frequent item sets previously determined by the FPGrowth algorithm **(line 24 - 32)**. Only data samples whose associated actions exist in the frequent item set are considered and stored in the array that collects the cleaned data samples **(line 25 - 32)**.

In the *else-branch*, which examines non-sequential task data, all filtered data samples are processed and stored in the array of cleaned data samples **(line 34 - 40)**, since in non-sequential tasks frequent co-occurring features do not play a role, and adherence to ordered action sequences is not required.

After the *for-loop*, an *if-condition* again checks whether the task from which the dataset originates is a sequential task **(line 41)**. If the condition is true, frequent item sets are

again determined from the cleaned dataset by the FPGrowth algorithm **(line 42)**. This is necessary because the FPGrowth algorithm finds the most frequent combinations of features and actions in the dataset that promise high reward values. Subsequently, only samples that also contain an action are selected, as it is possible that frequent item sets are also found that only consist of feature-value pairs and do not contain an action **(line 44 - 45)**. In a final step, depending on which *if-else branch* was entered, the preprocessed data is returned for further processing either for sequential task data or non-sequential task data **(line 46, 48, 49)**.

Algorithm 14: PreprocessData - Part 2

```
21 forall sample IN filteredData do
22     forall rule IN sample.rules do
23         if isSequential then
24             forall items IN frequentItems do
25                 if rule.action IN items.items then
26                     data = {}
27                     forall feat IN rule.state do
28                         cleanFeat = feat.replace("/|-|+|*|.|,|;|:", "")
29                         data[cleanFeat] = rule.state[feat]
30                     data['Action'] = rule.action
31                     cleanedData.push(data)
32                     break
33         else
34             if rule.action !== undefined then
35                 data = {}
36                 forall feat IN rule.state do
37                     cleanFeat = feat.replace("/|-|+|*|.|,|;|:", "")
38                     data[cleanFeat] = rule.state[feat]
39                 data['Action'] = rule.action
40                 cleanedData.push(data)
41 if isSequential then
42     frequentItems = FPGrowth(cleanedData, 10)
43     forall rule IN frequentItems do
44         if 'action' IN rule.items then
45             frequentSamples.push({rule: rule})
46     preprocessedSamples = frequentSamples
47 else
48     preprocessedSamples = cleanedData
49 return {preprocessedSamples, actionsequences}
```

D. List of all Scientific Publications

Referenced Publications

(1) Improving the utilization of AAL devices through semantic web technologies and web of things concepts, N Merkle, S Zander - Procedia computer science, 2016.

(2) Agent-Based Assistance in Ambient Assisted Living Through Reinforcement Learning and Semantic Technologies: (Short Paper), N Merkle, S Zander - On the Move to Meaningful Internet Systems. OTM, 2017.

(3) Using a semantic simulation framework for teaching machine learning agents, N Merkle, S Zander - Procedia Computer Science, 2018.

(4) A Semantic Use Case Simulation Framework for Training Machine Learning Algorithms, N Merkle, S Zander, V Simko: 21st International Conference, EKAW 2018, Nancy, 2018.

(5) Cooperative web agents by combining semantic technologies with reinforcement learning, N Merkle, P Philipp - Proceedings of the 10th International Conference on Knowledge Capture (K-CAP), 2019.

(6) Context-Aware Composition of Agent Policies by Markov Decision Process Entity Embeddings and Agent Ensembles, N Merkle, R Mikut - Semantic Web Journal (SWJ), IOS Press, 2024.

Other First Author Publications

(1) Distributed Context-Aware Applications by Means of Web of Things and Semantic Web Technologies. N Merkle - The Semantic Web. Latest Advances and New Domains, 2016.

(2) Representing and Reasoning over User Intentions and Actions in Adaptive Ambient Assisted Living*, N Merkle, S Zander - Germany,(i).

(3) Self-Service Ambient Intelligence Using Web of Things Technologies. N Merkle, B Kämpgen, S Zander, 2016.

(4) Utilizing IoT Devices for Monitoring and Adjusting Clinical Pathway Exercises. N Merkle, S Zander - SWIT@ ISWC, 2017.

(5) Towards Self-Adaptive Medical Coaching Agents, Nicole Merkle, Stefan Zander, 2017, Zeitschrift, KSS Research Workshop - A Selection of Talks and Presentations on Designing the Digital Transformation, KIT Scientific Working Paper.

Co-Author Publications

(1) Augmented Living: Einsatz von Augmented Reality im Alltag, M Ksoll, M Prilla, A Rashid, T Herrmann, N Merkle - Mensch & Computer 2014-Workshopband, 2014.

(2) Augmented Hearing for Elderly People-From Requirements to Implementation, P Barralon, IRM Martins, N Merkle, S Schwarz - ICT4AgeingWell 2015, 2015.

(3) Enhancing the utilization of IoT devices using ontological semantics and reasoning, S Zander, N Merkle, M Frank - Procedia Computer Science, 2016.

(4) Continuous support for rehabilitation using machine learning, P Philipp, N Merkle, K Gand, C Gißke - it-Information Technology, 2019.

Eidesstattliche Versicherung

gemäß § 6 Abs. 1 Ziff. 4 der Promotionsordnung des Karlsruher Instituts für Technologie für die Fakultät für Wirtschaftswissenschaften

1. Bei der eingereichten Dissertation zu dem Thema „*A Domain-Independent Agent Framework for Modelling, Simulating and Solving Tasks in Internet of Things (IoT) Systems*“ handelt es sich um meine eigenständig erbrachte Leistung.

2. Ich habe nur die angegebenen Quellen und Hilfsmittel benutzt und mich keiner unzulässigen Hilfe Dritter bedient. Insbesondere habe ich wörtlich oder sinngemäß aus anderen Werken übernommene Inhalte als solche kenntlich gemacht.

3. Die Arbeit oder Teile davon habe ich *bislang nicht* an einer Hochschule des In- oder Auslands als Bestandteil einer Prüfungs- oder Qualifikationsleistung vorgelegt.

4. Die Richtigkeit der vorstehenden Erklärungen bestätige ich.

5. Die Bedeutung der eidesstattlichen Versicherung und die strafrechtlichen Folgen einer unrichtigen oder unvollständigen eidesstattlichen Versicherung sind mir bekannt. Ich versichere an Eides statt, dass ich nach bestem Wissen die reine Wahrheit erklärt und nichts verschwiegen habe.

Karlsruhe, den *09.10.2023*