A Semantic Use Case Simulation Framework for Training Machine Learning Algorithms

Nicole Merkle[1], Stefan Zander[2], and Viliam Simko[1]

[1] FZI Forschungszentrum Informatik am KIT, Information Process Engineering, Haid-und-Neu-Str. 10-14, D-76131 Karlsruhe, Germany,
merkle@fzi.de, simko@fzi.de

[2] University of Applied Sciences Darmstadt, Institute for Computer Science, Schöfferstrasse 8B, D-64295 Darmstadt, Germany,
stefan.zander@h-da.de

Abstract. To train autonomous agents, large training data sets are required to provide the necessary support in solving real-world problems. In domains such as healthcare or ambient assisted living, such training sets are often incomplete or do not cover the unique requirements and constraints of specific use cases, leading to the cold-start problem. This work describes a semantic simulation framework that generates qualitative use case specific data for Machine-Learning (ML) driven agents, thus solving the cold-start problem. By combing simulated data with axiomatically formalized use case requirements, we are able to train ML algorithms without real-world data at hand. We integrate domain specific guidelines and their semantic representation by using SHACL/RDF(S) and SPARQL CONSTRUCT queries. The main benefits of this approach are (1) portability to other domains, (2) applicability to various ML algorithms, and (3) mitigation of the cold-start problem or sparse data.

1 Introduction

Real-world problems and domain-specific use cases are often so diverse and complex, that a thorough formalization of all requirements and interdependencies at design time is difficult, if not impossible. Moreover, use cases are often not generalizable because their performance depend on individual and varying parameters (e.g. user-preferences, goal-oriented requirements, context-dependent requirements etc). In order to provide sufficient support in such situations, agents usually require formal specifications (e.g. in the form of rules) or large datasets that cover domain and problem characteristics. However, problem-specific datasets are neither always available nor—in most of the cases sufficiently—representative for a given problem. Especially not in situations, where agents have to deal with personalization and context-dependent use cases in indeterministic and heterogeneous environments. Privacy policies as well as regulative, governmental, and technical restrictions further contribute to the lack of sufficient data. Moreover, as starting, virtual agents have neither an appropriate trained model nor an individual knowledge representation that enables them to train a specific and

individualized prediction model on-the-fly. In most of the cases, the only thing available are guidelines or domain expert knowledge, which need to be represented in a machine-processable way. Furthermore, teaching a machine learning-based system needs time, especially when the data have to be gathered and preprocessed first during its runtime. This leads to the undesirable situation that an agent acts in a random and uncertain way (cold-start problem) at the beginning of its lifecycle, resulting in an unwanted and inappropriate behaviour for a given situation or context [5]. Especially, in critical situations (e.g. user interactions) this can be problematic. Moreover, almost every performed agent activity generates new data, since every activity has an impact to the states, conditions and actors in an environment. The presented approach aims at providing use case specific and personalized synthetic data in order to overcome the outlined cold-start problem. The assumption underlying our approach is that semantically created synthetic simulation data serve as substitute of missing real-world data, thus enabling machine learning based agents to immediately perform appropriate activities although the required data are not available at the beginning of an agent's life cycle.

In order to accomplish this objective, our approach provides a simulation framework that utilizes ontological semantics for defining and processing general meta-model specifications for use cases of different domains. Under consideration of the discussed problems and challenges, our approach addresses the following research questions:

RQ1 Does the presented simulation framework generate data of sufficient quality to solve the cold-start problem in uncontrolled environments?

RQ2 Is the simulation framework capable of adapting its data-generation process to different problem domains sufficiently?

In order to evaluate the proposed simulation framework, we a) apply our approach in two heterogeneous domains—healthcare and smart home environment—in order to show the universal validity of our approach and b) test if it sufficiently solves the cold-start problem.

The remainder of this work is structured as follows: Section 2 discusses related work. Section 3 presents the fundamental concepts of the semantic simulation framework. Section 4 demonstrates through two example applications from the health-care and smart home domain, that the approach is sufficiently generic for its application in different problem domains. Section 5 evaluates the approach by applying it to the two example use cases. Therefore, we consider appropriate guidelines from the Chronic Kidney Disease Pathway (CKDPathway)[3] and rules for controlling devices in an smart home environment in order to compare our results with the results of some trained and applied ML algorithms. Section 6 concludes the work by deriving *lessons learned* and provides an outlook to future activities.

[3] http://ckdpathway.ca

2 Related Work

The cold-start problem has been extensively researched in the domain of recommender systems (e.g. [15,5,17,7]). Several solutions have been proposed to tackling both *item* and *user cold start* problems: Active learning, which is a special field of semi-supervised machine learning (cf. [16,2,14]), aims at evaluating the usefulness of data points in order to improve recommender system performance and data quantity. Collaboration and data exchange strategies are often deployed in agent-based system environments to help agents in providing appropriate assistance in new and unanticipated situations by learning from the experiences of other agents [12,4]. Hybrid approaches, e.g. combining content-based matching approaches and collaborative filtering [15] or association rules and clustering techniques[1] revealed to be helpful in mitigating the cold-start problem by deducing similarity indicators from content-based characteristics. In order to address the sparsity of user profiles, Yahoo! research developed a collective learning representation framework to tackle both the item cold start and the user cold start by using matrix factorization techniques (cf. [7]) such as alternating least squares and multiplicative updates [6].

Although several solutions have been proposed to tackle cold-start problems, the deployment of semantic technologies, ontology-based description frameworks, and semantic simulation frameworks have received only marginal attention. The synergies between recommender systems and semantic knowledge structures such as ontologies have first been studied by Middleton et al. [8]. Nouali et al. [11] demonstrated that a semantically enhanced description of user and resource information increases precision, coverage and quality of recommendation systems. A review about general semantic recommender systems has been published by [13] whereas DiNoia and Ostuni [3], Musto et al. [10], and Tomeo et al. [20] (among others) specifically focus on systems that have been extended with data from the Linked Open Data (LOD) initiative. Although the positive effects of semantic technologies on recommender systems, in particular for content-based filtering, are broadly acknowledged (cf. [21]), many recent approaches (e.g. [3,10,20]) started to analyze the extent to which recommender systems can be enhanced using LOD (cf. [10]). However, a semantic simulation framework comparable to one in this work was not specifically addressed. A very similar approach to the one proposed here that addresses the sparsity and scarcity problem was developed by Thanh-Tai et al. [19]. In contrast to generating new data via simulation, their approach uses a semantic model to generate similarity data for a given original dataset.

3 Approach

The presented simulation framework can be considered as part of a bigger system architecture that acts as a recommendation- and task-execution- framework for given use cases. Thereby, a use case constitutes an aggregate representation of evident states and possible activities that are performed by virtual agents in

order to solve given problems. Domain knowledge is necessary and provided by human experts in a formal representation. This domain knowledge is used by the simulator component in order to generate numerical state representations and compute reward or punishment values that are relevant for making decisions and executing activities in the appropriate use case. The transformation into a numerical state representation is required because in neural networks as well as in many other ML algorithms, only numerical representations are processed. Therefore, nominal values are encoded by applying one-hot-encoding to generated feature values. Moreover, all values are normalized before they are sent to the appropriate agent. The machine learning (ML) algorithms that the targeted agents apply, are either reinforcement learning (RL) (e.g. Deep-Q-Networks) or (un-) supervised (e.g. deep neural networks, collaborative filtering, K-Means, latent semantic analysis) ML algorithms. Therefore, the simulator generates during the online-training of the RL agent the appropriate CSV[4] files as a side-product. While the RL agent is trained online in several thousand episodes by means of rewards and punishments, the generated CSV files serve as datasets for the (un-) supervised algorithms.

The application of the entire system architecture is divided in two phases, a *learning phase* and an *adaption phase*. The learning phase addresses the training of machine learning (ML) models based on generated guideline-rules and domain knowledge data, represented as RDF(S) instances. The learning phase is conducted in order to avoid the cold-start of the ML components. However, before the learning phase can start, the simulator component requires a formal representation of use cases that require to be simulated. The general architecture and step-by-step process are represented in Figure 1. First, the domain

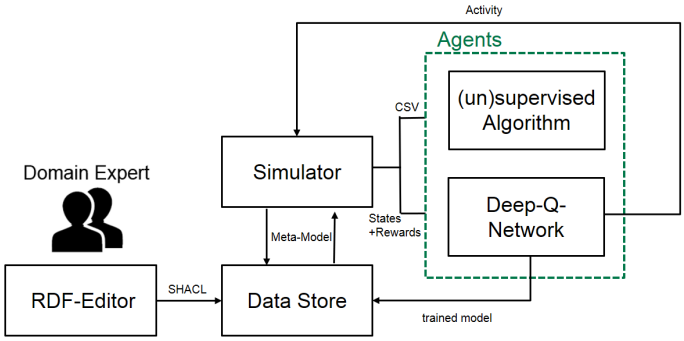


Fig. 1: Overview of the system architecture

expert specifies by means of an arbitrary RDF editor (e.g. Topbraid, Protge, Semantic MediaWiki) the use cases' A-Box representation, comprising states, activities and guideline-rules following the use case meta-model constraints. The

[4] comma separated values

simulator utilizes this A-Box representation for generating the numerical state representations (numerical state vector + reward and CSV) for the appropriate ML algorithms. Our approach utilizes different semantic web technologies, such as the *Shape Constraint Language (SHACL)*[5] and *SPARQL*[6] in order to represent and request simulation use cases, so that the simulator can process this representation. We use SHACL as a vocabulary because it allows to define validation constraints for RDF(S) graphs. Moreover, its application assures a closed-world assumption that is necessary in order to avoid inconsistency and undefined states within the simulation framework. Since the different states are defined by conditions and guideline-rules, the simulator transforms these rules into `SPARQL CONSTRUCT` queries in order to derive the appropriate state in that the simulation currently resides. This step is necessary because the simulator generates or adjusts during the training phase just single values of a state vector representation. Therefore, it needs to derive the new state after the values are updated. According to the constructed and derived state, the simulator reasons the appropriate reward value of a state. This reward value represents the acceptability of a state and is in particular required in reinforcement learning algorithms. According to the reinforcement learning paradigm [18], reward values are utilized in order to train strategies for a given task by rewards and punishments of the environment.

Later in the *adaption phase*, the ML algorithms adjust their model representation by collected real-world data that have been gathered during the system runtime. *Adaption* means in this context that individualized data is used in order to personalize general ML models that initially were only trained based on guideline-rules.

3.1 The Framework Architecture

The architecture of the framework consists of different components that interact with each other via event-driven Web protocols.

As soon as the simulation starts, the appropriate ML agent sends an initial message to the simulator. This message includes information about the requested use case and in how many iterations or episodes the training has to be performed. An episode starts on a random start state and finishes first, as soon as a goal state of the use case is achieved by the agent. Usually several thousand episodes are required in order to train an agent.

The use case information is necessary for the simulator in order to request from the graph store the appropriate meta-information regarding the specific use case that shall be simulated. The use case simulation meta-model (SHACL shape definitions) together with use case instances are stored in a graph database. The simulator and other components can request the entire meta-model graph representation via a SPARQL endpoint. Internally, the simulator generates an in-memory RDF store that is created and utilized using the RDF4J[7] API.

[5] https://www.w3.org/TR/shacl/
[6] https://www.w3.org/TR/rdf-sparql-query/
[7] http://rdf4j.org/

Based on semantically represented use case guidelines or use case pathways (e.g. clinical pathway), the simulator generates a numeric vector representation of the current use case state. This vector representation is sent as event message to the corresponding ML algorithm. The algorithm is capable, by means of the graph store, to send *activity* URIs to the simulator. Subsequently, the simulator looks up what *effects* the received activity might have. Considering for instance the healthcare domain, an *effect* can be the increase of the patient's *heartrate* value, after the patient has made some kind of sports activity. The simulator looks then up the last stored heartrate value in the state vector and increases this value, in order to send the updated state representation to the ML algorithm. The change of single values in the state vector ensures that the states are adapted according to the performed activity effects. This interaction between simulator and agent happens iteratively in thousands of episodes, depending on the agents iteration request or until it has finished its training.

During the training, the simulator stores all generated vector states as data samples in a CSV[8] file, so that this CSV file can be utilized by additional ML algorithms for training a use case specific model.

3.2 The Simulation Meta-Model

Often, use cases and domain guidelines are described by domain experts, in a human-readable documentation. In order to allow the simulator to process use case- as well as guideline-representations and generate realistic data for the algorithms, we propose a RDF(S)-based simulation meta-model that we discuss here. The appropriate shape definitions are described in the next sections. Figure 2 provides an overview of the simulation meta-model elements. The SHACL definition of the simulation meta-model can be found at this URL[9]. It is planned to enhance our devised meta-model by classes of already existing ontologies such as the universAAL[10] and QUDT[11] ontology in order to follow the Linked Open Data principles of the semantic web.

Simulation Tasks

A simulation task encapsulates all information, required by the simulator. It is linked to the use case that shall be simulated.

Use Cases

The use case node shape comprises the related *states-of-interest* and *activities* that an agent can perform or recommend. Moreover, it is linked to all required *observation features*, that are monitored by the agent in order to make decisions.

[8] coma separated value
[9] https://raw.githubusercontent.com/vcare-project/vcare-models/master/simulation.shapes.ttl
[10] http://ontology.universaal.org/
[11] http://qudt.org/

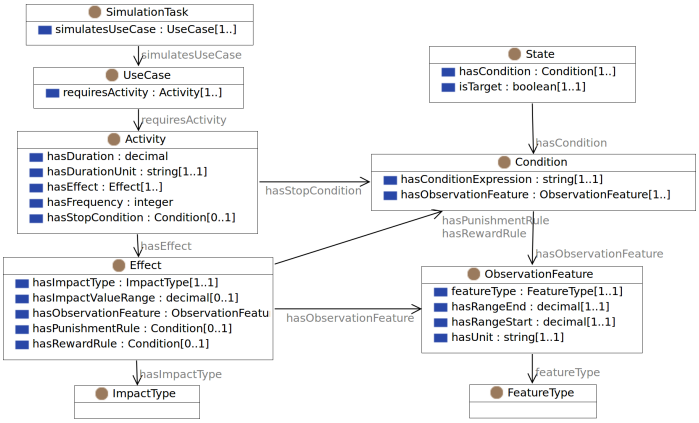


Fig. 2: UML representation of the simulation meta-model

States

The state shape is represented and derived by conditions that hold for the state. A state can be either a target state or an intermediate state, indicated by a boolean value. If a target state is achieved, the task is fulfilled by the agent and the next training episode can start.

State Conditions

State conditions reflect the guidelines for a use case. More concretely, they define whether all gathered observations apply to a single state. These conditions are specified in the form of mathematical expressions as illustrated below: The following expression types are allowed.

1. Ordinal expression — (e.g. Heart rate $<$ 100)
2. Nominal expression — (e.g. Eye-color: brown, blue, green)
3. Logical expression — (e.g. Heart rate $<$ 100 AND Heart rate $>=$ 40

The simulator takes these condition expressions for utilizing them within SPARQL query FILTER expressions. Moreover, the condition expressions provide the range boundaries in which the simulator can generate values in order to simulate a certain state. Therefore, the simulator parses the conditon expressions and preprocesses the generated values by feature scaling into a normalized numerical feature vector representation, sorts and keeps them in-memory in order to select the appropriate values in a context-dependent manner. Nominal values are transformed by *one-hot-encoding* into a numerical vector representation consisting of ones and zeros. Thereby, every nominal value has a fixed position inside the vector. The value one at the appropriate position indicates that the nominal feature is present while the value zero indicates that it is absent. As logical operators are *AND*, *OR* and *XOR* allowed, since they are sufficient for

specifying logical conditions. Furthermore, a state condition is linked to one or multiple *observation features*, since the conditions are build up by observation values.

Observation Features

An observation feature defines the feature type (e.g. nominal, numerical), the feature's value unit (e.g. kg, cm), and the value range containing a starting point number and ending point number. Within this range definition, the observation values are generated by the simulator.

Activities

Activities reflect what an agent or user can perform in order to achieve a goal and solve a task. The most important property of an activity in this approach, is its effect to the environment, since activity effects are changing the state of the environment. Therefore, activities lead to new observable states. Moreover, an activity can have a duration (e.g. 10 minutes) and can be performed in a certain frequency. Every activity has a stop condition that indicates whether it can be finished by the agent.

Effects

Since states are represented to the RL agent as numerical vectors, an effect can either *increase*, *decrease* a value by a certain amount or *convert* an one-hot-encoded nominal value to its opposite (e.g. true (1) to false (0) or vice versa). For this reason, the effect has an *impact* that indicates, whether a value shall be increased, decreased or converted. Moreover, an effect is linked to the appropriate observation feature to which it has an impact. Every activity is related at least to one effect, since activities cause state changes by their effects. The simulator is enabled by the effect representations to update state representations. Thereby, the *impact value range* property specifies how much the effect impacts an observation feature. Furthermore, the effects's reward- and punishment-rules are utilized by the simulator in order to reason rewards and punishments for certain performed activities in a given state because the effects of an activity are either desired or not desired effects depending on the observed state.

3.3 The Generation of Numerical State Representations

In this section, we comment on how single state vector values are adapted in order to generate new states according to performed activity effects. Since the state conditions are mathematical expressions, the simulator queries these conditions from a given state representation and parses them in order to create, based on their operators, appropriate expressions. These expressions provide an instruction for the simulator to create values within certain ranges. Subsequently, every

observation feature value is sorted and stored in an in-memory data structure together with the generated values. The sorting of the values allows to update the state vector with higher or lower values depending on the current activity effects. Let us assume, we have a mathematical expression that expresses that the body mass index (BMI) requires to be in between a range of 18.5 and 25, as the depicted condition in Equation 1.

$$(\mathsf{BMI} \geq 18.5) \wedge (\mathsf{BMI} \leq 25.0) \tag{1}$$

This condition holds for instance for the state *TargetBMI*. The simulator parses this expression and selects the start and end value of the BMI range as well as the operators (e.g. >=, <=). Based on these expression tokens, values are generated for the state *TargetBMI* inside the given range.

Every observation feature is stored with a list of generated values, restricted by the given guideline conditions. As soon as the simulator receives an activity, it looks up its effect to the observation features from the meta-model and compares the previous observation feature values in order to increase, decrease or convert them. This is easy to handle, since the values are sorted. Therefore, the simulator is able to select values from the lists that are higher or lower or the opposite (0 versus 1) than the last stored value.

3.4 State Reasoning using SPARQL Queries

In every state-update-loop, the simulator needs to be aware of the currently generated state. Since only single observation values in the state vector are adapted, the state in which the simulation resides is not evident. Therefore, the simulator generates a statement for every updated observation, such as the following assertion:

```
:BMI :hasValue "22.4"^^xsd:double .
```

The simulator adds this assertion to the in-memory RDF(S) representation and infers the states in which an ML agent might be. This is done by using automatically generated `SPARQL CONSTRUCT` queries. Every generated query has the same structure:

```
CONSTRUCT {?agent :isInState ?state.}
WHERE {
  ?agent rdf:type :Agent.
  :BMI :hasValue ?bmi.
  FILTER(?bmi >= 18.5 && ?bmi <= 25.0)}
```

The `CONSTRUCT` queries are generated during the parsing of the condition expressions. Therefore, the simulator retrieves the states and their conditions and maps them to `SPARQL FILTER` expressions. The simulator generates by the

`CONSTRUCT` query a new graph statement that expresses that the agent is in a certain state. The state conditions are listed in the `WHERE` clause within `FILTER` functions. In this way structured queries are universally applicable since they allow to reason the current state of an agent according to the given observations. Considering the previous BMI example, the simulator would infer the *TargetBMI* state, provided that the given `FILTER` conditions hold.

3.5 The Reasoning of Rewards

The simulator has also the task to assign rewards to the agent according to the performed activity of the agent in a given state. This requires that the simulator determines the reasoned state of the agent and the appropriate effect of the performed activity. After the current state is reasoned as discussed in section 3.4, the simulator reasons by means of CONSTRUCT queries the appropriate reward value for a state and the performed activity. The SPARQL query looks as follows:

```
CONSTRUCT {:DecreaseBMI :hasReward "-1.0"^^xsd:double.}
WHERE {
?agent :performsActivity ?activity.
?activity :hasEffect :DecreaseBMI.
{?agent :isInState :Underweight.} UNION {?agent :isInState :NormalBMI.}}
```

In the given example, the simulator reasons a punishment of minus one for the *DecreaseBMI* effect, if the agent is in the state *Underweight* or in the state *NormalBMI*. Therefore, rewards and punishments are assigned based on states and activity effects, since an effect can be desired or not, depending on the given states.

4 Proof-of-Concept

We selected two use cases from different domains, in order to demonstrate the generalizability of the presented approach. The first use case (*CKDPathway*) stems from the healthcare domain. The objective of this use case is to allow a virtual coach to make activity recommendations based on the given clinical pathway and by made vital sign observations. The second use case is about the intelligent control of IoT devices in a smart home environment depending on user activities and environmental states. In the next subsections, we describe the use cases and their relevant entities.

4.1 The Chronic Kidney Disease Pathway

The CKDPathway provides guidelines for the diagnosis and treatment of Chronic Kidney Disease (CKD). The pathway recommends lifestyle management activities, such as e.g. low sodium diet, physical exercises and medication. Nine-

teen different states can be identified (CKDWithDiabetes, CKDWithoutDiabetes, NoCKDRisk, CKDRisk, TargetA1C etc.) and thirty-seven possible activities (FluidIntakeRegulation, FruitVegetableConsumption, Walking, TestACR, TesteGFR, etc.). Moreover, the CKDPathway contains target states (TargetBMI, TargetA1C, TargetBloodpressure, etc.) that represent the long-term objectives of the CKDPathway. Every state in our setting is represented by a numerical vector, consisting of twenty-one observation values. The sensed observation features are: eGFR, ACR, Hematuria, Diabetes, BMI, systolic value, diastolic value etc. The diagnosis of CKD requires different rules (e.g. eGFR $< 60\text{ml/min}/1.73^2$, ACR >= 3mg/mmol) that are also provided by the *CKDPathway*. For more details regarding the pathway, we refer to the appropriate online source[12] and our shape definition[13].

4.2 Intelligent Smart Home Control Systems

The *intelligent smart home control* (SHC) use case has the objective to control or regulate context-dependent devices in the environment. The state space consists of different states, such as e.g. (1) *UserInFrontOfTV*, (2) *TVOn*, (3) *TVOff*, (4) *RoomTemperatureCold*, (5) *RoomTemperatureWarm*, (6) *RoomBrightnessDark*, (7) *RoomBrightnessBright*. The concrete objectives of this use case are:

- Autonomously switch the TV on or off, depending on the user location.
- Autonomously regulate the brightness in the rooms according to the user location, states of the installed lamps and brightness sensors.
- Autonomously regulate the temperature in the rooms according to the preferences of the user and the room temperature.

Considering these objectives, we determined the following observation features that are related to the previous mentioned states:

- BrightnessInLUX - Numerical value
- TemperatureInCelsiusDegree - Numerical value
- DistanceToTVInCentimeters - Numerical value
- UserLocatedRoom - Nominal value (Name of the room)

The according activities of the smart home system are: (1) *TurningLightOn*, (2) *DimLight*, (3) *TurningLightOff*, (4) *IncreaseTemperature*, (5) *DecreaseTemperature*, (6) *TurnTVOn*, (7) *TurnTVOff*. An excerpt of the appropriate rules are defined in Equation 2. The appropriate states are determined by the given rule conditions.

$$\begin{aligned} \mathsf{BrightnessInLUX} \leq 100 &\rightarrow \mathsf{RoomBrightnessDark} \\ \mathsf{TemperatureInCelsiusDegree} \geq 25 &\rightarrow \mathsf{RoomTemperatureWarm} \\ \mathsf{UserLocatedRoom} : \mathsf{XY} &\rightarrow \mathsf{UserInRoomXY} \end{aligned} \tag{2}$$

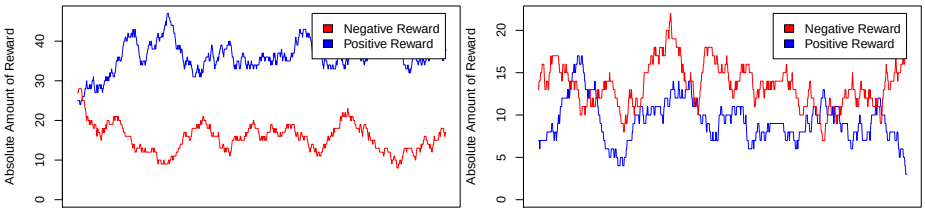
[12] http://ckdpathway.ca

[13] https://github.com/vcare-project/vcare-models/blob/master/vcare.shapes.ttl

5 Evaluation

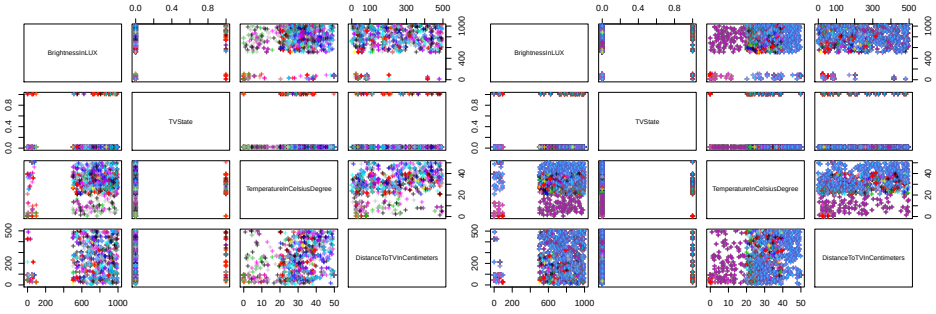
In this section, we aim at answering posed research questions RQ1 and RQ2. The goal is to explore if the generated simulation data can be used to avoid the cold-start problem. Moreover, we want to prove that our approach is applicable to multiple domains. Therefore, we evaluate if (a) the trained algorithms enable a RL agent to maximize its collected rewards by learning to recommend activities, yielding in the highest expected rewards and (b) the generated state vectors allow ML algorithms to predict, for unseen feature vectors, the correct states. Thereby, the clinical pathway guidelines as well as the given smart home rules provide the ground-truth of our evaluation. Based on the simulation meta-model, we created RDF(S) instances of the previously discussed use cases. Then, based on their representation, we trained a Deep-Q-Network RL algorithm[14], which was proposed by DeepMind in their paper [9]. We adopted for our evaluation, Karpathy's reinforcejs library because it implements different RL algorithms and is usable in browser as well as server environments. In order to prove the quality of the generated data, we counted the absolute rewards over time that the ML agent received during the evaluation phase. The diagram in Figure 3a represents the execution steps (x-axis) with a window size of 70, while the y-axis represents the collected relative absolute rewards. We shows the positive rewards and received punishments using differently colored lines, comprising 4000 data samples. Figure 3a shows that the *CKDPathway* use case provided a good separation between the negative and positive rewards. As the learning phase starts, the lines start to diverge in opposite directions. However, neither the blue line increases nor the red line decreases significantly. This indicates that the RL algorithm does not improve after some training time. In the ideal case the blue line would increase steadily, while the red line would decrease, so that the agent achieves the maximum expected reward. Figure 3b, representing the agent improvement for the *SHC* use case is even worse. It seems that the agent learns nothing and that it even collects more punishments than rewards. Considering this result, the agent learns nothing from the generated simulation data. In order to explore the separation of the generated states, we plotted every feature of the SHC case as multiple 2D-projections in Figure 4. Every state in the plot is represented by a specific color. The better the separation of states, the better is the generated data for training ML algorithms. Here, the 4000-samples data set (b) provides an apparent advantage over the 1000-samples data set (a). This insight indicates: the more data is generated the more are the states separable. A feature vector representing several states in the same time leads to an insignificant separation of states. Therefore, the approach can be improved if the simulator generates sparse feature vectors representing only one state in a single training step. Moreover, states, represented by the same features, require significant guideline-rules. This observation can be later used to improve the quality of the models prepared by the domain experts.

[14] Implementation from Karpathy under `https://github.com/karpathy/reinforcejs`



(a) (b)

Fig. 3: (a) CKD: Absolute Reward over time. (roll sum, win=70) and (b) SHC: Absolute Reward over time. (roll sum, win=70)



(a) 1000 samples (b) 4000 samples

Fig. 4: Distribution of states projected to two dimensions (pairs of observations).

Our evaluation has the following limitations. (1) We did not compare our approach against a baseline due to the cold-start problem – no data sets are available that can act as a baseline. For this reason, the domain expert's guideine-rules and the proposed clinical pathway provide already a baseline against that we can evaluate our approach. However, it is planned to implement additionally a rule-based agent in order to compare its performance against the trained RL agent's performance. (2) Another limitation is that we did not evaluate the training of a ML agent with and without the simulator. Therefore, it is required to show in a future evaluation that the simulator allows a faster convergence of the ML algorithm than a training without a simulator. In a later stage, we would like to prove that the trained RL agent generalizes better than a rule-based agent does due to the limitations of rules.

6 Conclusion and Outlook

We proposed and evaluated a semantic simulation framework that utilizes a general meta-model for use cases and their guideline rules. In the framework's evaluation, we showed for RQ1 and RQ2 its limitations when it comes to solving the cold-start problem and its application to multiple domains. We demonstrated

what a crucial role the domain experts play in the process. The approach is rather sensitive to the quality of rules and rewards modeled. Model validation (we used SHACL) is essential for keeping the quality of the A-Box representation at a usable level. We also showed that the approach can be improved by modeling the state representations using sparse numerical vectors (for better separation) or by providing more samples (which is usually the case in the ML world). Given that the proposed improvements are performed, we can conclude that (a) the simulation framework generates data of sufficient quality so that the cold-start problem can be solved in uncontrolled environments and (b) that the simulation framework is capable of adapting its data-generation process to different problem domains sufficiently. As supporting tools for the creation of use case specifications and agent profiles, we utilized Semantic MediaWiki (SMW)[15], which allows annotating wiki pages semantically. Domain experts can use SMW to easily model the required concepts for the simulation framework. For the implementation of the RL-agent, we adopted the library reinforcejs[16], which implements different RL algorithms. We demonstrated that rewards and punishments should be assigned depending on the given state and activity effects, since agents need to learn the effects that their activities might have in certain situations. Moreover, unwanted states have to be considered in the A-Box representation as well as activities with negative impact to the given states in order to assure that an agent does not get stuck in a certain state. Considering the evaluation results, our future work is to improve the approach by implementing the discussed lessons learned.

Acknowledgement

This work is supported by the European Union (H2020) under the vCare project (grant agreement No. 769807).

References

1. A.K.Mariappan, H.S.: A hybrid approach to solve cold start problem in recommender systems using association rules and clustering technique. International Journal of Computer Applications 74(4), 17 – 23 (July 2013)
2. Carlson, A., et al.: Toward an architecture for never-ending language learning. In: AAAI. vol. 5, p. 3. Atlanta (2010)
3. Di Noia, T., et al.: Recommender Systems and Linked Open Data, pp. 88–113. Springer International Publishing, Cham (2015), doi:10.1007/978-3-319-21768-0_4
4. Godoy, D., Amandi, A.: An agent-based recommender system to support collaborative web search based on shared user interests. In: Haake, J.M., Ochoa, S.F., Cechich, A. (eds.) Groupware: Design, Implementation, and Use. pp. 303–318. Springer Berlin Heidelberg, Berlin, Heidelberg (2007)

[15] `https://www.semantic-mediawiki.org/wiki/Semantic_MediaWiki`
[16] `https://github.com/karpathy/reinforcejs`

5. Lika, B., Kolomvatsos, K., Hadjiefthymiades, S.: Facing the cold start problem in recommender systems. Expert Systems with Applications 41(4, Part 2), 2065 – 2073 (2014), doi:10.1016/j.eswa.2013.09.005
6. Mantrach, A.: Cold start solutions for recommender systems, https://research.yahoo.com/_c/uploads/SeminarUCSD.pdf
7. Mehta, R., Rana, K.: A review on matrix factorization techniques in recommender systems. In: 2017 2nd International Conference on Communication Systems, Computing and IT Applications (CSCITA). pp. 269–274 (April 2017)
8. Middleton, S.E., Alani, H., Roure, D.D.: Exploiting synergy between ontologies and recommender systems. CoRR arXiv:cs/0204012 (2002)
9. Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., Riedmiller, M.A.: Playing atari with deep reinforcement learning. CoRR arXiv:cs/0204012 (2013)
10. Musto, C., et al.: Introducing linked open data in graph-based recommender systems. Information Processing and Management 53(2), 405 – 435 (2017), doi:10.1016/j.ipm.2016.12.003
11. Nouali, O., Belloui, A.: Using semantic web to reduce the cold-start problems in recommendation systems. In: 2009 Second International Conference on the Applications of Digital Information and Web Technologies. pp. 525–530 (Aug 2009)
12. Palau, J., et al.: Collaboration analysis in recommender systems using social networks. In: Cooperative Information Agents VIII. pp. 137–151. Springer (2004)
13. Peis, E., et al.: Semantic recommender systems. analysis of the state of the topic. Hipertext.net 6, (online) (2008), http://hipertext.net/english/pag1031.htm
14. Rubens, N., et al.: Active Learning in Recommender Systems, pp. 809–846. Springer US, Boston, MA (2015), doi:10.1007/978-1-4899-7637-6_24
15. Schein, A.I., Popescul, A., Unger, L.H., Pennock, D.M.: Methods and metrics for cold-start recommendations. In: Proc. of SIGIR'02. pp. 253–260. Tampere, Finland (2002), doi:10.1145/564376.564421
16. Settles, B.: Active learning literature survey. Computer Sciences Technical Report 1648, University of Wisconsin–Madison (2010), http://axon.cs.byu.edu/~martinez/classes/778/Papers/settles.activelearning.pdf
17. Son, L.H.: Dealing with the new user cold-start problem in recommender systems: A comparative review. Information Systems 58, 87 – 104 (2016), http://www.sciencedirect.com/science/article/pii/S0306437914001525
18. Sutton, R.S., Barto, A.G.: Introduction to Reinforcement Learning. MIT Press, Cambridge, MA, USA, 1st edn. (1998)
19. Thanh-Tai, H., Nguyen, H.H., Thai-Nghe, N.: A semantic approach in recommender systems. In: Future Data and Security Engineering. pp. 331–343. Springer International Publishing, Cham (2016)
20. Tomeo, P., et al.: Exploiting linked open data in cold-start recommendations with positive-only feedback. In: Proc. of the 4th Spanish Conference on Information Retrieval. pp. 11:1–11:8. CERI '16, ACM, New York, NY, USA (2016), doi:10.1145/2934732.2934745
21. Yang, R., et al.: Using semantic technology to improve recommender systems based on slope one. In: Semantic Web and Web Science. pp. 11–23. Springer (2012), doi:10.1007/978-1-4614-6880-6_2