Semantic Modelling of Control Logic in Automation Systems

Knowledge-Based Support of the Engineering and Operation of Control Logic in Building and Industrial Automation Systems

Zur Erlangung des akademischen Grades eines

DOKTORS DER INGENIEURWISSENSCHAFTEN (Dr.-Ing.)

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

genehmigte

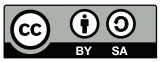
DISSERTATION

von

M. Sc. Georg Ferdinand Schneider

Tag der mündlichen Prüfung: 18. Februar 2019

Hauptreferentin: Prof. Dr. Dr.-Ing. Dr. h. c. Jivka Ovtcharova
Korreferent: Prof. Dr.-Ing. Hendro Wicaksono



Zusammenfassung

Automatisierungssysteme schaffen in vielen Bereichen die Grundlagen, auf denen heutige, moderne Industriegesellschaften basieren. Obwohl in der Vergangenheit wichtige Errungenschaften in der Forschung zur Automatisierungstechnik erreicht wurden, bestehen weiterhin Herausforderungen bezüglich des Engineerings und des Betriebs von Automatisierungssystemen, die die Nutzung und den Einsatz dieser Systeme erschweren. Als Gründe für diese Probleme sind die Komplexität dieser Systeme durch ihre schiere Größe und ihre Komplexität aufgrund der Kombination von cyber und physikalischen Komponenten zu nennen. Des Weiteren führt der zunehmende Einsatz von Informations- und Kommunikationstechnologien zu einer weiteren Verflechtung dieser System über ihre bisherigen, hierarchischen Strukturen hinaus und damit zu einer weiteren Zunahme der Komplexität. Eine weitere Herausforderung ist, dass für ein reibungsloses Engineering und einen reibungslosen Betrieb dieser Systeme eine Vielzahl von Beteiligten aus unterschiedlichen Fachdisziplinen zusammenarbeiten müssen. Dies wird durch die Heterogenität der eingesetzten Softwarewerkzeuge und Datenformate erschwert, die einen automatisierten Austausch von Wissen behindern.

Folglich besteht ein dringender Bedarf an Methoden, die die wissensintensiven Aufgaben in Zusammenhang mit dem Engineering und dem Betrieb von Automatisierungssystemen im Kontext heterogener Softwarewerkzeuge und Datenformate unterstützen und, als Antwort auf die Komplexitätszunahme, automatisieren. Eine Voraussetzung für die Entwicklung solcher Methoden ist die formale Repräsentation von Domänenwissen mit Hilfe eines Modells. Die Analyse des Stands der Technik in dieser Arbeit zeigt, dass kein Ansatz existiert der es erlaubt einen wesentlichen Bestandteil der Domäne Automatisierungssystem, die Domänen Regelung und Steuerung und Regelungslogik, explizit zu beschreiben und dieses Wissen mit angrenzenden Domänen zu vernetzen.

Ein wesentlicher Beitrag dieser Arbeit besteht in der Vorstellung eines neuartigen, semantischen Modells, dass es erlaubt, sowohl Wissen der Domänen Regelung und Steuerung, als auch der Domäne Regelungslogik explizit und formal zu beschreiben. Zusätzlich ist es nun erstmals möglich dieses Wissen mit angrenzendem Domänenwissen, wie zum Beispiel aus dem Maschinenbau oder der Elektrotechnik, zu vernetzen. Das Modell wird unabhängig von der Implementierung in der Unified Modeling Language spezifiziert und mit Hilfe von Semantic Web Technologien implementiert. Das Modell ist in zwei Schichten aufgebaut. Auf der oberen Ebene wird allgemeines Wissen der Domäne Regelung und Steuerung modelliert, dass, wie in der Arbeit demonstriert, leicht mit angrenzenden Domänen verbunden werden kann. Auf

der unteren Ebene wird das allgemeine Wissen der Domäne Regelung und Steuerung, um die Domäne der Regelungslogik erweitert und für die jeweilige Regelungslogik explizit spezifiziert.
Zur Validierung des Modells wird in zwei separaten Fallstudien evaluiert, ob es das notwendige Wissen für zwei neuartige wissensbasierte Methoden repräsentieren kann. In der ersten Fallstudie wird eine wissensbasierte Methode zur Verbesserung des Betriebs von Automatisierungssystemen in Gebäuden prototypisch umgesetzt und getestet. Dabei ermöglicht das entwickelte Modell Faktenwissen, das aus dem Engineering der Regelungslogik gewonnen wurde, formal zu beschreiben. Dieses Wissen wird dann genutzt, um automatisiert Regeln zu instanziieren, die es ermöglichen automatisiert zu überprüfen, ob die tatsächlich implementierte Regelungslogik sich im Betrieb genauso verhält wie ursprünglich entworfen. In der zweiten Fallstudie wird eine wissensbasierte Methode zur Unterstützung des Engineerings von industriellen Automatisierungssystemen vorgestellt. Hier wird gezeigt, dass, basierend auf dem neuen Modell, die gleichzeitige formale Verifikation von verschiedenen Regelungsverfahren und die gleichzeitige formale Verifikation von Regelungsverfahren und Wissen über die automatisierte Anlage möglich ist. Zusätzlich, wird gezeigt, dass die Methode inkrementelle Aktualisierungen des Faktenwissens ermöglicht und ein bidirektionaler Austausch von Fallwissen zwischen dem ursprünglichen Format und der Wissensbasis möglich ist.
Durch die Schaffung des neuen Modells ist nun die Möglichkeit gegeben formal und explizit Wissen der Domänen Regelung und Steuerung, sowie Regelungslogik zu beschreiben. Basierend auf diesem Modell werden zwei neuartige, wissensbasierte Methoden vorgestellt, die es ermöglichen das Engineering und den Betrieb von Automatisierungssystemen zu vereinfachen und zu verbessern.

Abstract

Automation systems are a key component of modern society. Despite important achievements by automation science and technology in the past, challenges related to the engineering and operation remain in the future and impede their deployment and use. In particular, this is caused by their complexity arising from their size, their cyber-physical nature and the further introduction of advanced information and communication technologies in the domain, which cause the dissipation of former hierarchical architectures prevalent in both building automation and industrial automation systems. In addition, the heterogeneity of tools and data formats utilised by various stakeholders during their engineering and operation causes problems in their development and deployment.

Hence, there is a strong need to develop methods, which support the knowledge-intensive tasks associated to the engineering and operation of these systems as well as automate these tasks to cope with the inherent complexity. A prerequisite to develop these methods is the representation of domain knowledge by means of a formal domain model. The analysis of the current state of the art as presented in this thesis reveals the absence of a formal model to explicitly describe domain knowledge on the automatic control and control logic domains.

The contribution aspect and novelty of this thesis is the presentation of a semantic model, to formally and explicitly describe domain knowledge on the automatic control and control logic domains in automation systems and link this knowledge to adjacent domains. The model is specified independently from its implementation in the Unified Modelling Language (UML) using an object-oriented modelling methodology and is implemented using semantic web technologies following an ontology-based approach. The model is separated in two layers to enable a seamless integration of the novel model with adjacent domains, such as mechanical and electrical engineering, and extend the model with explicit formal descriptions of different control logic types.

To validate the capabilities of the model in describing the respective domain knowledge, its ability to fulfil the knowledge requirements of two novel knowledge-based methods presented in two separate use cases is tested. In the first use case, a knowledge-based method to support the operation of control logic in building automation systems is introduced. The developed method allows the automated, rule-based verification of designed control logic with monitoring data obtained from a building. In the second use case, again, the novel formal domain model is used to enable a knowledge-based method for the support of the engineering of control logic in industrial automation. The method allows to automate the formal verification of the

designed control logic for different types of control logic and plant data, to incrementally verify changes and updates to the automatic control logic design and to support the bidirectional flow of information from the model to the target format and vice versa.
The formal definition of domain knowledge in automation systems offers the possibility to reduce the effort required for the engineering and operation of these systems. This thesis contributes to this effort by providing means to integrate knowledge on the control and control logic domains.

Acknowledgements

This thesis is the result of my research activity carried out from February 2015 until October 2018 during my time as a research associate at the Fraunhofer Institute for Building Physics (IBP) in Nürnberg, Germany as well as a PhD candidate at the Department of Mechanical Engineering at the Karlsruhe Institute of Technology (KIT) in Karlsruhe, Germany.

First of all I would like to express my special thanks for my supervisor Prof. Dr. Dr.-Ing. Dr. h. c. Jivka Ovtcharova, Head of the Institute for Information Management in Engineering (IMI) at KIT. Her thorough advice and guidance encouraged me many times and thrived me to pursue my research.

I would like to thank Prof. Dr.-Ing. Hendro Wicaksono for taking the role as the co-supervisor of my thesis as well as being a mentor and friend.

Special thanks go to my colleagues and supervisors from IBP. First of all, I would like to thank Simone Steiger for giving me the opportunity and freedom to pursue the research documented in this thesis as well as her advice and guidance in many occasions. I would like to thank Dr. Georgios D. Kontes for his support, his excellence in being a devil's advocate and his friendship. Special thanks go to Prof. Dr.-Ing. Gunnar Grün for his support, advice and guidance in many occasions. Additionally, I would like to express my gratitude to Dr. Dimitrios V. Rovas for giving me the opportunity to pursue this research, for planting the initial seed and for being an inspiring source on what research is about. I would like to thank our students Viet-Tung Hoang, Georg A. Peßler, Haonan Qiu, Wang Lingzhe, Michael Rißmann, Lars Nolting, Andre Michaelis, Markus Deschler and Max Blanck for the great time and for the great working attitude.

I would also further like to thank my fellow PhD students and colleagues within IBP including Dr. Matthias Mitterhofer, Aude Bougain, Peter S. Noisten, Maximilian Kienberger, Stephanie Türkdogan, Christina Kuba, Christa Dürig, Dr. Victor Norrefeldt, Arnav Pathak, Dr. Sebastian, Stratbücker, Dr. Sumee Park, Thomas Kirmayr, Marie Pschirrer, Georg Haag and Matthias Kersken. From the Energie Campus Nürnberg (EnCN) special thanks go to the EnCN BBQ friends including Susanna Bordin, Christina Betzold, Kyriaki Koutrouveli, Julian Buderus, Kutralingam Kandasamy, Christian Allar, Nelly Wedel, Eva Kränzlein, Frank Piepenbreier, Rajesh Chanda, Friedmann Eppelein, Ioanna Dimopoulou and Rudolf Heindel.

Being myself not based at the IMI but closely related to it I would like to express my special thanks to the colleagues and staff which I met at the IMI for including me as an 'external' very openly on every occasion I was visiting the institute. This includes in particular Karin Schmid,

Thomas Maier, Kiril Tonev, Polina and Victor Häfner, Matthes Elstermann. Special thanks go to Klemens Haas and Andreas Kimmig for the many pleasant conversations during lunch breaks and for the supply of excellent coffee.
Additionally, I would like to express special thanks to the members of the W3C Linked Building Data Community Group for their great discussions and exchange on semantic web technologies in the engineering domain. This includes in particular Prof. Dr. Pieter Pauwels, Mads H. Rasmussen, Dr. Walter Terkaj, Dr. Kris McGlinn, Dr. Maxime Lefrançois, Dr. Ana Roxin, Matthias Bonduel, Jyrki Oraskari, Ekatarina Petrova and many more from the group. Special thanks go to Prof. Dr. Pieter Pauwels for his support, guidance and friendship as well as for the great time while following his invitation to Ghent University, Ghent, Belgium.
I would like to acknowledge the generous funding provided by the state of Bavaria through the Energie Campus Nürnberg and financial support through the 'Aufbruch Bayern (Bavaria on the move)' initiative of the state of Bavaria.
Special thanks go to my parents for their support and for making me love science. In addition, special thanks go to my siblings, Alex W. Chatterton and Sebastian Brixner, who have provided tremendous help in the final part of this thesis on revisions, proofreading and for being such a wonderful bunch.
Finally, I would like to thank Pia for her support and patience and for letting me be an important part of her life.

Nürnberg, October 2018 *Georg F. Schneider*

Everyone has the right to education. Education shall be free, at least in the elementary and fundamental stages. Elementary education shall be compulsory. Technical and professional education shall be made generally available and higher education shall be equally accessible to all on the basis of merit.

United Nations General Assembly: Universal Declaration of Human Rights, Article 26 (1)

Table of Contents

1 Introduction

1.1 Background and Motivation

Automation of processes is at the heart of modern society. Traffic light systems in cities, autonomous driving, mass-production of goods, heating control in buildings, autopilots in planes and safety control in nuclear power plants are only some of the examples to be mentioned where automation takes a key role in enabling the reliable and safe operation of the respective processes. In particular, automation comes into play when processes are repetitive, or dangerous to humans and often machines can perform these tasks with higher precision, faster, more efficient and at lower cost [SSKD11]. The beginnings of automation technology can be traced back to the 1800s where mechanical feedback control by means of a centrifugal governor controlled the speed of James Watt's steam engine. The technology quickly advanced from mechanical systems to analogue electronics and, finally, digital control by the introduction of micro-electronics and communication networks in the 1970s and 1980s [SSKD11]. Dependent on the application area of automation technology different specialised domains emerged, such as building [DCVK16], industrial [SSKD11] and process [FSUT17] automation. In the past frequently novel technological advances made in one domain quickly have been adopted by the other domains [SSKD11].

The successful application of automation technology is associated with solving some of the existing global challenges [VDI15]. One major challenge is the reduction of energy-related emissions of greenhouse gases originating from burning fossil fuels to meet the primary energy demand in industrialised and emerging countries. Figure 1.1 shows the concentration of greenhouse gases in the earth's atmosphere calculated from measuring the variation of deuterium (δD) in ice cores extracted from Antarctic glaciers [SQM$^+$07]. The measurements indicate that over the past 650,000 years changes in the concentration of relevant greenhouse gases occurred. However, the levels as observed in the past 150 years, when human activities related to burning fossil fuels set off, has not previously been identified. Hence, developing technologies to reduce the emissions is of utmost importance.

The buildings and industry sector together are responsible for a significant share in the primary energy demand in industrialised countries (e.g. buildings 43.3%, industry 28.5% in Germany [AGE17]). In the buildings sector the development of energy efficient buildings, which preserve the set comfort conditions and maintain a healthy indoor climate is an ultimate goal for

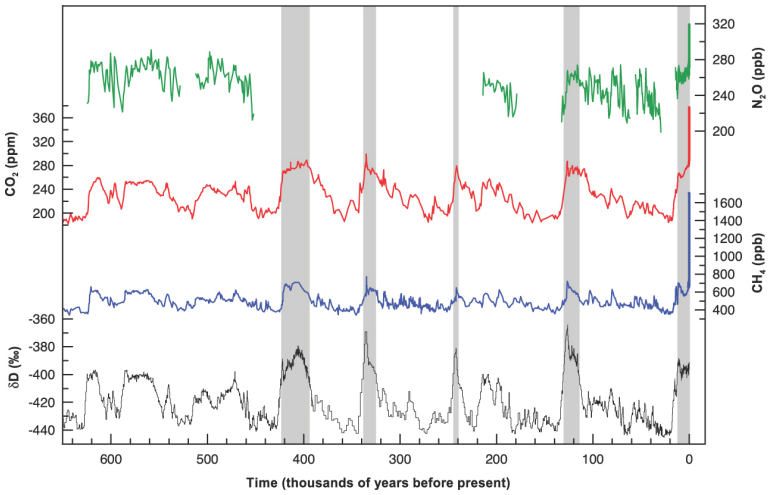


Figure 1.1: 'Variations of deuterium (δD) in antarctic ice, which is a proxy for local temperature, and the atmospheric concentrations of the greenhouse gases carbon dioxide (CO_2), methane (CH_4), and nitrous oxide (N_2O) in air trapped within the ice cores and from recent atmospheric measurements. Data cover 650,000 years and the shaded bands indicate current and previous interglacial warm periods' [SQM+07]. Source: Figure TS.1 in Solomon *et al.*: Contribution of Working Group I to the Fourth Assessment Report of the Intergovernmental Panel on Climate Change, Cambridge University Press.

scientists in the field to reduce the emissions of greenhouse gases. Building Automation Systems (BAS) are a prerequisite for energy efficient operation of buildings and the highest rating in terms of energy efficiency can only be achieved by deploying BAS [EN 17, USG16]. Similar efforts can be observed in the industrial domain and, in particular, the further introduction of Information and Communication Technologies (ICT) in the domain is understood as an enabler for energy efficient operation of productions plants [Eur08, Wic16]: 'Embedded, smart components and systems, sensor/actuator networks and control algorithms can be used to achieve a positive effect on emissions.' [Eur08].

Automation technology can help to significantly reduce the primary energy demand needed by technical processes and subsequently can help to reduce the emissions from greenhouse gases emitted from burning fossil fuels. Hence, an ultimate goal is to lower the barriers for deploying automation technologies for their successful usage. In addition, successful conversion of energy from renewable power plants depends on automation technology, e.g. modern wind turbines have automation systems installed, e.g. for pitch control of rotor blades.

1.2 Problems and Challenges of Building and Industrial Automation Systems

As outlined above, automation systems are an essential ingredient for tackling some of the future global challenges. Despite the rapid technological advancements of automation technology in the past [SSKD11], future challenges still remain. This section identifies problems and open challenges of automation systems, which are in the scope of this thesis: building and industrial automation systems.

1.2.1 Complexity of Automation Systems

The complexity of automation systems is a challenge for engineers. For the design and operation of these systems a significant share of domain and expert knowledge is required. Due to the complexity of the systems this knowledge needs to be shared across the distributed and multidisciplinary stakeholders. Main sources for this complexity are the size of the systems, their combination of physical and cyber parts and the paradigm change initiated through the digital transformation in automation systems.

The *size* of automation systems can be challenging as these are used for operating large and complex technical systems, such as refineries, car production plants or airports. For instance, the number of datapoints, i.e. an entity in a BAS with 'an addressable point of interaction between the control system and its domain objects' [DCVK16], in a commercial building, such as an airport or shopping mall, is in the range of 100,000.

A second reason for their complexity arises from their inherent combination of physical components, i.e. the actual plant under control, with cyber components, i.e. for example a control algorithm executed on a Programmable Logic Controller (PLC). These systems of combined computation and physics are termed Cyber-Physical Systems (CPS) [Lee08]. In particular, automation systems as described qualify as CPS [HVA16]. To distinguish the components of CPS, here, the three layered architecture proposed by Sztipanovits *et al.* [SKK+12] (see Figure 1.2) is adopted here. According to the authors CPS can be divided into a physical, platform and computation layer:

- *Physical Layer*: Physical systems, mechanical or electrical; The tangible part of CPS;
- *Platform Layer*: The computational platform of CPS is needed to execute the cyber part;
- *Computation Layer*: Here, cyber objects (software) of CPS are actually described, which can be generic algorithms or actual control logic of a physical plant. This software performs the computations, which significantly determine the resulting behaviour of a CPS.

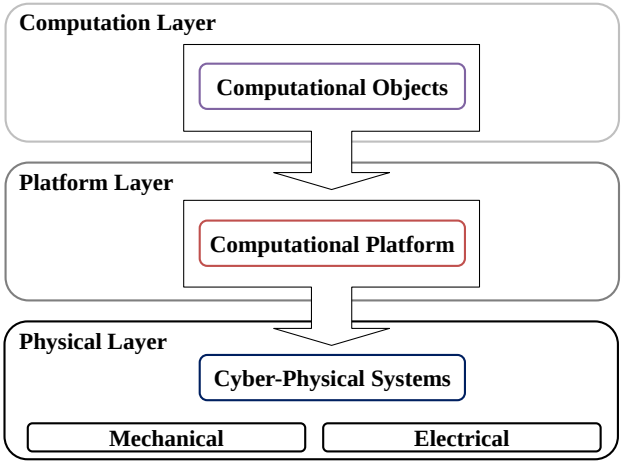


Figure 1.2: Layered structure of cyber-physical systems (adapted from Sztipanovits *et al.* [SKK+12]).

To cope with the complexity arising from automation systems being CPS, scholars conclude that appropriate means of abstraction need to be considered for each of the defined layers [SKK+12, Lee08]. This differentiation into components is also reflected in the distinguished steps in the engineering of automation systems, which typically involve mechanical (physical), electrical (platform) and software (computation) engineering [SH17]. Again, a considerable amount of domain specific and expert knowledge is required for the successful implementation of the respective tasks. However, to achieve the goal of delivering one CPS, the respective domain and expert knowledge of both domains needs to be integrated.

As emphasised above, the computation part of automation systems has a significant impact on the overall performance of the resulting system. In an automation system even if the physical part of the system has been designed in an excellent manner, an insufficient cyber part can spoil the efforts made. For instance, if highly efficient technical equipment and a sophisticated envelope are installed and designed for a building, an insufficiently designed and parametrised BAS is identified as one cause for the perceived gap between the predicted and actual energy demand of buildings [DW14].

In recent years, ICT originating from the world wide web are increasingly introduced in the automation domain, partly in response to global challenges, such as fulfilling the rising primary energy demand. These efforts align with the ubiquitous introduction of ICT technologies in various domains and industries termed *digital transformation* (e.g. Gimpel & Röglinger [GR15]). The digital transformation introduces a paradigm change in the automa-

tion domain from several hierarchical to two distinguished layers (see Figure 1.3). Classical approaches (left hand side in Figure 1.3) typically structure automation systems hierarchically into several layers, e.g. for BAS in the standard International Standards Organization (ISO) 16484 [ISO04b] and Industrial Automation Systems (IAS) in International Electrotechnical Commission (IEC) 62424 [IEC16]. In the hierarchical structures sensors and actuators attached to some physical system are placed at the bottom layer (Field layer/ Level 0). Low-level control logic embedded in field devices is placed in the layers above (Automation layer/ Level 1,2) and communicates with the sensors and actuators to operate some piece of equipment or process. At the top layers (Management layer/ Level 3,4) high-level functionalities are placed, which perform calculations or predictions and have access to other additional information apart from the physical system, e.g. a weather forcast for model-predictive control in BAS [KKGR16] or Enterprise Resource Planning (ERP) system for production and maintenance planning in IAS.

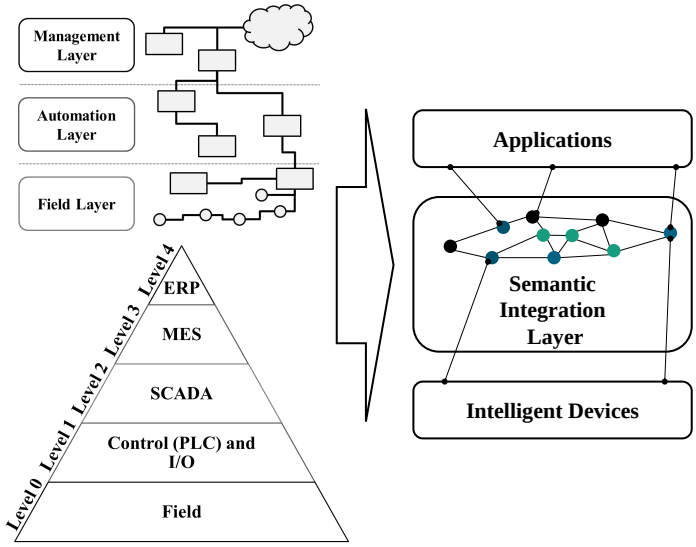


Figure 1.3: Paradigm change in automation domain [KDKO14] from hierarchical structures (upper left, [ISO04b], adapted; lower left, [IEC13d], adapted) to a two layered approach (right, e.g. Vogel-Heuser *et al.* [VHDB13]), where intelligent devices communicate with each other and applications through a semantic integration layer. ERP - Enterprise Resource Planning, MES - Manufacturing Execution System, PLC - Programmable Logic Controller, I/O - Input/ Output.

These hierarchical architectures tend to dilute nowadays into two distinct layers (right hand side in Figure 1.3). This paradigm shift envisions future automation systems to be constituted of one layer of intelligent devices, which communicate with each other (Internet of Things (IoT) [GBMP13]) and expose the captured information to a common semantic integration layer (e.g. Krueger *et al.* [KDKO14]). In a second layer applications leverage on the consistent, semantically unambiguously defined information from devices and other sources to act on the intelligent devices, while providing additional services alongside. To enable this seamless interaction of devices and software agents realising novel services, the semantics and the knowledge for the interaction and analysis need to be represented machine-understandable. In general, four key technological advances can be associated with the paradigm change and can be considered as enablers as well as prerequisites towards reaching the defined goals for two layered future architectures:

1. The processing power of embedded devices placed near to the actual equipment increased significantly over the past decades; Hence, more complex applications might be running on the equipment allowing the machines to communicate directly with each other (i.e. IoT) [KDKO14];
2. Availability of computational resources and data storage from cloud computing, which allow computational expensive applications, such as real-time data analysis [KDKO14];
3. Latency between low-level embedded devices and cloud services has decreased by magnitudes through novel communication technologies, e.g. fourth and fifth generation mobile data transfer (4G/5G), thus, enabling real-time communication and exchange;
4. Formal methods for the semantically precise, machine-interpretable definition of domain and expert knowledge to enable the seamless interoperation among heterogeneous devices and services [KDKO14].

Within the context of the digital transformation [GR15] the further introduction of advanced ICT in industrial automation is considered as the fourth step in the evolution of industry [KDKO14]. Associated with this a number of initiatives started, which aim at bundling research, funding and standardisation activities [KWH13, EA07, RCW13]. The Industrie 4.0 initiative [KWH13] is a prominent example for an initiative started by a German consortium. In their report, Kagermann *et al.* [KWH13] present recommendations on how the strategic initiative Industrie 4.0 can be implemented and formulate a research roadmap as well as requirements. The scholars conclude that, amongst others, semantic models are needed to enable self-descriptive devices and production systems exposing case and domain specific knowledge to others. In particular, this includes aspects of standardised descriptions of the automatic control domain, such as procedures and functions [KWH13, p. 45]. Also, from industry, e.g. Krueger *et al.* [KDKO14], there is the demand for the semantic description of engineering information and knowledge artefacts associated to the control of a plant. Similar demands are

made by Vogel-Heuser [VH14b] and Kleinemeier [Kle14], where in future the knowledge on how a production entity can be integrated with other entities or on how it can reconfigure itself to produce different goods is specified machine-interpretable. Hence, machines can integrate and configure *themselves* [HZ16]. With respect to IAS a key demand is to enable the seamless exchange of all related information among software tools and stakeholders [VH14b].
In the Architecture, Engineering, Construction and Facility Management (AEC/FM) industry increasingly ICT are introduced to interconnect systems, tools and stakeholders. Associated terms used within this context are *smart home*, *smart buildings* or *smart cities* dependent on the scale of interest. Over the life cycle of a building case specific information and knowledge from different domains [SBNM16] are generated, which typically are stored in disparate silos [COC+13]. This heterogeneity is a reason for the increased introduction of ICT in the AEC/FM domain and integrates particularly well with the Building Information Modeling (BIM) method [ETSL11]. BIM is a method, which focusses on the seamless exchange of information between stakeholders by means of model-based information exchange [ETSL11]. In the described context, BAS play a vital role in enabling these future smart systems in the AEC/FM domain. They provide the access to sensors and actuators placed in a building as well as allow the connectivity through the web to intelligent applications and services, which analyse the data in a context through specified knowledge and potentially integrate with other knowledge sources, such as formalised knowledge originating from BIM. Again, for the seamless integration of the various formats and tools as well as the scalable deployment of intelligent services, such as automated configuration and deployment of Fault Detection and Diagnosis (FDD) services (e.g. Dibowski *et al.* [DHR16a]), the semantics of the associated knowledge needs to be specified.

1.2.2 Heterogeneity in Automation Systems Engineering and Operation

In addition to the problems arising from the complexity of automation systems as presented above, the heterogeneity of data format, tools and stakeholders involved in the engineering and operation of automation systems is a problem when dealing with these systems. In particular, the engineering of control logic is a complex, knowledge-intensive task and needs to be closely interconnected with the actual engineering of the plant [Vya13, VHFST15, SH17]. In this collaborative effort, the exchange of engineering knowledge between multiple stakeholders over consecutive stages in a development cycle is required [SH17, ESS+17]. Typically, losses occur at handover between stages as heterogeneous tools and formats are used, which make the exchange cumbersome time and cost-consuming. During the course of the engineering of control logic in automation systems various knowledge artefacts are produced, such as conceptual designs, abstract specifications of control behaviour, contact plans to real hardware, etc.

For instance, Strahilov & Hämmerle [SH17] enumerate for the engineering of a production plant seven steps in the engineering process, where in each step one or more stakeholders are involved. These stakeholders work on and share knowledge spanning fifteen domains [SH17]. Typically, multiple, often incompatible tools with own formats exist for each domain. The combinatorial combination of stakeholders and domains with associated tools and formats causes the complexity as mentioned.

Various artefacts are related to control knowledge, which are generated during engineering and operation [Che15]. Figure 1.4 presents a control diagram as specified in ISO 16484-5 [ISO04b], which allows to visualise control knowledge in a very condensed fashion. These schemas are structured in three separated sections and the respective, depicted knowledge is explained in the following.

A common practice in control logic engineering is to encapsulate some functionality into reusable blocks. These blocks can be connected by their inputs and outputs, which is referred to as logical topology[1]. The logical topology is often illustrated in block diagrams, e.g. the lowest area in Figure 1.4.

Another important part of knowledge in the automatic control domain relates to the relationship of the controller to the actual physical plant (Middle area in Figure 1.4). This is enabled through sensors and actuators, which convert between real physical measurements or control actions and virtual values usually exposed as datapoints [DCVK16] on the communication bus of an automation system. Various ways exist to specify how the actual control logic behaves. For instance, a number of formalisms are developed to describe discrete control logic [Har87, IEC14a, LS17]. In particular, in linear control the transfer function and their plots [Abe10] are common practice to describe how the response of an element depends on its input signal. In the top area of Figure 1.4 control sequences [VDI07, ISO11] are depicted, which describe how the outputs of each depicted control actor can be derived from its inputs.

At some point in time the control logic design is implemented as actual source code. This source code then can be executed on a PLC. Also, the source code is part of the control knowledge. The conversion from the specification as a design to source code can depend on the implementation technology and, thus, certain capabilities of the programming language may require a certain way for the implementation. Knowledge of the parameters and alarming [DCVK16] is important and in combination with expert knowledge can be used for verification and debugging of operation [SPS17]. For instance, in Figure 1.5 the view provided by an Integrated Development Environment (IDE) for the programming of control logic in automation systems is given (Beckhoff TwinCAT 3.1 [Bec18]). The project structure with physical Inputs and Outputs (I/O) and reusable blocks of control logic separated in Program Organizational Unit (POU)s [IEC14a] is displayed. Also, code for the implementation of interfaces of a piece of control logic and the actual specification of the control logic is graphically illustrated using

[1] See Definition 2.5.

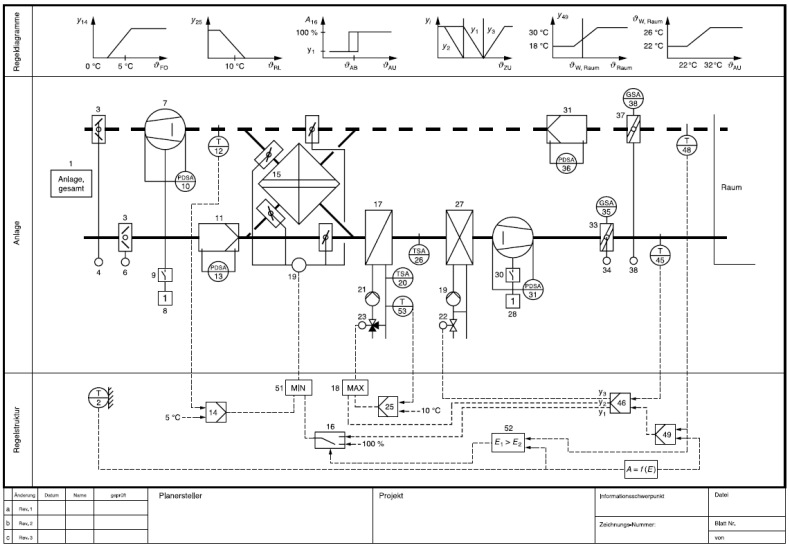


Figure 1.4: Control diagrams [ISO04b] are a method to visualise control logic knowledge in a very condensed manner. (Source: [VDI07], Wiedergegeben mit der Erlaubnis des Verein Deutscher Ingenieure (VDI) e.V., engl.: Reproduced with permission of the German Association of Engineers).

the Sequential Function Chart (SFC) [IEC14a] formalism. The design made and implemented can be exported and exchanged via the PLCopen XML [PLC09] format. However, in most projects the current mode of information exchange is often based on spreadsheets, drawings and textual descriptions [SPS17], [SH17]. This keeps the knowledge concealed for further reuse and exploitation in downstream services.

1.2.3 Summary and Gap Identification

In essence, the described problems and challenges associated with building and industrial automation systems are:

1. The complexity of automation systems arising from their size, their cyber and physical nature and the paradigm change for future automation systems through introduction of advanced ICT technologies. This complexity is only manageable through the use of software programs and automated engineering methods;

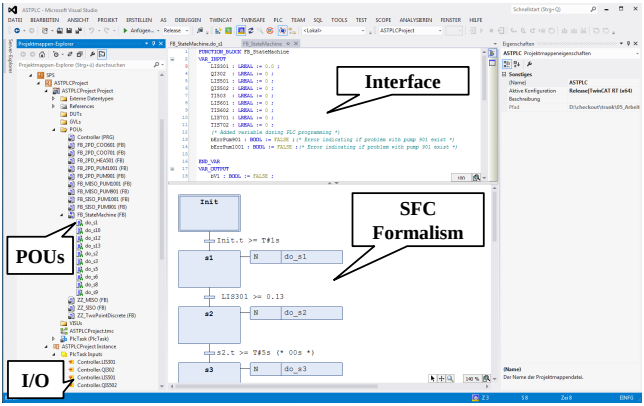


Figure 1.5: Example for programming a programmable logic controller in Beckhoff TwinCAT 3.1 [Bec18] using the Sequential Function Chart (SFC) formalism [IEC14a]. Interface variables can be defined and signals from sensors or actuators can be connected to these (I/O). Reusable blocks of control logic are separated in Program Organizational Units (POUs).

2. The heterogeneity of tools and data formats utilised in the design and operation of automation systems by various stakeholders involved over the life cycle. Consequently, engineering knowledge is distributed across these disparate silos and concealed for reuse and exploitation.

For instance, in BAS best practices and standards exist on how to control specific technical equipment to reduce energy consumption, while keeping comfort levels acceptable [ISO11]. It would be very useful if an embedded device shipped along with a piece of Heating Ventilation and Air Conditioning (HVAC) equipment explains to some supervisory control how it is controlled, such that the equipments energy demand is at its minimum. This avoids the repetitive implementation or copying of the same code (code clones [VHFST15]) in engineering projects. Also, potential errors when implementing might occur resulting in inefficient operation of the equipment. These defects and inconsistencies need to be automatically identified to be removed from the design. With respect to IAS these novel methods are even more important when designing self-explaining cyber-objects as envisioned for the next generation of automation systems [KDKO14]. For example, the novel methods should enable automated verification of the specified, implemented and executed control logic. In particular, the automated verification should be possible for different types of control logic as well as considering information and knowledge from adjacent domains, such as the physical plant [VHFST15]. During com-

missioning of an automation system often manual changes are applied to the code implemented in a PLC. These changes need to be fed back to the model hosting the control logic design and the same verification mechanisms should apply [VHFST15].
Hence, there is a strong need for novel, automated methods, which support the knowledge-intensive tasks conducted during the engineering and operation of control logic in automation systems. In particular, there is the strong need to address the prevalent heterogeneity in automation systems by enabling the sharing of use-case, domain and expert knowledge.

1.3 Objectives

To overcome the above stated problems the following objectives as well as research questions arising from these objectives are defined.

Objective 1 *Development of knowledge-based methods to support and automate the engineering and operation of control logic in automation systems.*

This objective addresses particularly the problem of complexity in the engineering and operation of automation systems. This complexity is not manually manageable by humans and, hence, to support the knowledge-intensive processes associated with the engineering and operation of control logic in automation systems, automated, knowledge-based methods need to be developed. The following research questions are formulated in association with the formulated objective.

- What formalisms do exist to enable automated, knowledge-based support for the engineering and operation of automation systems?
- What artefacts of use-case, domain and expert knowledge in control logic engineering can be formally specified?
- Which possibilities exist for the formal representation of knowledge to enable the support by machines?

Objective 2 *Development of a semantic model of control logic in automation systems.*

The described novel methods to support the knowledge-intensive tasks related to the engineering and operation of control logic in automation systems are only possible, if the respective domain knowledge is formally specified. Hence, a semantic model needs to be developed to address the problem of the utilised heterogeneous tools and formats. The semantic model is needed to integrate the distributed knowledge and to describe the control logic domain to form the basis for further intelligent applications. Associated research questions are:

- What are the fundamental concepts and relationships to explicitly model control logic in automation systems?
- What are the fundamental concepts and relationships to relate control logic to adjacent domains of automation systems?
- Which formal models do exist for reuse in formally specifying the domain of automation systems?

1.4 Contribution and Outline

This thesis aims to identify novel ways to describe the semantics of control logic in automation systems. The thesis structure aligns with the course of the research conducted and an overview is provided in Figure 1.6.

In Chapter 1 the research is first placed into its context and it is explained why BAS and IAS are relevant in solving some of the future global challenges such as energy. The relevant problems and challenges of BAS and IAS are identified, i.e. their complexity arising from their size, cyber and physical nature and the paradigm change in automation systems architectures as well as the heterogeneity of associated engineering tools and formats hiding this knowledge in disparate silos. Based on the described problems and challenges two objectives for this thesis are defined: Development of automated methods for the support of the knowledge-intensive tasks undertaken in the engineering and operation of control logic in automation systems and development of a semantic model for control logic in automation systems.

In response to these objectives Chapter 2 provides an overview of the basic concepts and terms related to the control logic domain in automation systems. Next, foundations of Knowledge Representation (KR) are discussed, which allow machines to process engineering knowledge as required by the objectives. Finally, a detailed analysis of the current state of the art is performed related to knowledge-based methods for the support of control logic engineering and operation of automation systems and existing formats and models for the description of domain knowledge of automation systems in general, automatic control and control logic in particular.

The findings and limitations arising from the analysis of the state of the art are summarised and requirements to overcome these limitations are defined in Chapter 3.

In response to the gap and identified shortcomings the main contribution of the thesis, a novel, layered semantic model for the automatic control domain and formal models to explicitly describe control logic in automation systems, are presented in Chapter 4. The semantic model is formalised using an ontology-based approach and explicit formal models of different control logic types are presented.

The modelling is evaluated in two use cases, as presented in Chapter 5, where the developed semantic models provide the basis for two knowledge-based methods. The first method presented

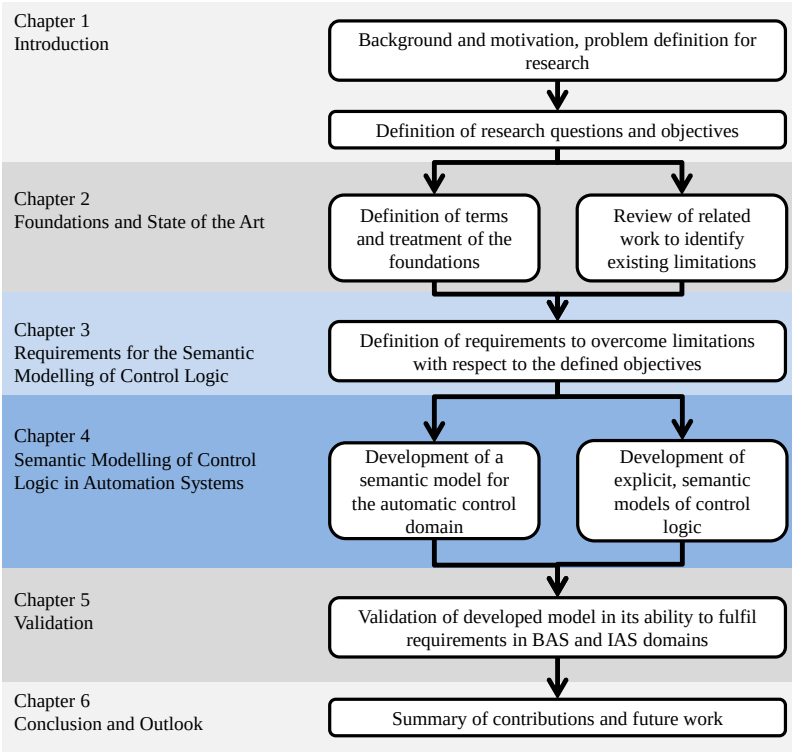


Figure 1.6: Overview of the organisation and structure of this thesis. BAS - Building Automation Systems, IAS - Industrial Automation Systems.

in Section 5.1 allows the automated rule-based verification of control logic in building automation. The second method described in Section 5.2 enables the knowledge-enhanced engineering of control logic in an application from industrial automation. Also, means to integrate this model with adjacent information domains are investigated in the use cases.
Finally, in Chapter 6 the contributions of the thesis to reach the defined goals are summarised and topics and open questions for future research are presented.

2 Foundations and State of the Art

This chapter briefly summarises the foundations of this thesis and provides a detailed analysis of the current state of the art of related work with respect to the defined objectives.
First, in Section 2.1 the terms and fundamental cornerstones of the domains of automatic control and control logic in BAS and IAS are briefly summarised, hence, providing a thorough description of the domain of interest as defined in the objectives. Next, the foundations and terms of KR are condensed in Section 2.2 as these technologies are the foundations for the intended development of knowledge-based methods.
Based on the defined objectives the current state of the art related to knowledge-based methods for the support of engineering and operation of control logic in automation systems are analysed in detail in Section 2.3. Furthermore, existing contributions related to formats and models for the description of the automatic control and control logic domains in automation systems is analysed in Section 2.4.
Finally, Section 2.5 summarises the findings of the analysis and identifies shortcomings in the reviewed works.

2.1 Automatic Control, Control Logic and Automation Systems

This section presents the basic building blocks of automatic control in automation systems. Where suitable, relevant terms are defined. A specific description of BAS and IAS is provided as these are the domains within the scope of the present thesis.

2.1.1 Overview on the Automatic Control and Control Logic Domains

Automation is an important domain in engineering science and many of today's achievements in engineering would not be possible without automation. It is particularly interesting when processes are repetitive or dangerous to humans and need to be executed fast, precisely and

automatically [SSKD11]. Automatic control is a specific domain within automation and is defined by IEC 60050-441 [IEC13b] as:

Definition 2.1 ***Automatic Control****: 'control of an operation without human intervention, in response to the occurrence of predetermined conditions' [IEC13b].*

The main goal of automatic control in technical systems is to align a controlled variable of a system with a given reference variable [Abe10]. In particular, this is of interest if changes to the reference variable happen or if external disturbances occur. The IEC 60050-351 [IEC13a] defines and standardises the terminology for the description of automatic control. In particular, closed-loop as defined in Definition 2.2 and open-loop control as defined in Definition 2.3 need to be differentiated.

Definition 2.2 ***Closed-loop control*** *is defined as: 'process whereby one variable quantity, namely the controlled variable is continuously or sequentially measured, compared with another variable quantity, namely the reference variable, and influenced in such a manner as to adjust to the reference variable' [IEC13a].*

The important feature of closed-loop control is that through its closed loop, the controlled variable effectuates itself by constant feedback. In Figure 2.1 terms and structure of a closed-loop controller as defined in the IEC 60050-351 is given. The abbreviated characters of the following discussion and as used in Figure 2.1 are given in Table 2.1.
The controlled variable (x) is measured by a measuring equipment and the resulting feedback variable (r) is compared in the comparing element with its reference variable (w). The calculated control difference variable (e) is then converted by the controlling element into a controller output variable (m), which then effectuates through the actuator the controlled system via the manipulated variable (y). Disturbances described by the disturbance variable (z) change the behaviour of the controlled system and cannot be changed by the controlling element directly. Only its effects on the controlled system can be mitigated. A number of terms are defined for referring to aggregates of the basic components. A controller constitutes of the controlling and comparing element. The final controlling equipment is constituted by the actuator and the final controlling element, which is also part of the controlled system. A controlling system includes both the controller and its actuator. Elements, which generate the final controlled variable (q) from the controlled variable (x), or, which convert the command variable (c) to the reference variable (w) are considered as part of the control system. The elements of the controlling system usually cannot access these directly. Finally, the whole aggregate, which includes all mentioned elements, is termed control system.
An example for closed-loop control is the control of a room temperature using a thermostat valve (e.g. [Abe10]). By turning the handle of the valve, a set point for the temperature of a room is defined. This setpoint is constantly compared with the room air temperature by some

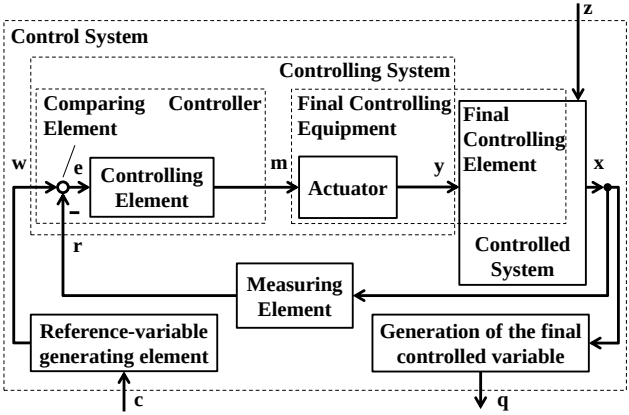


Figure 2.1: Structure and terms of a controller for closed-loop control (adapted from IEC 60050-351 [IEC13a]). The final controlling element is defined to be part of the controlled system. Abbreviated characters are summarised in Table 2.1 and are described in the text.

comparing element and, dependent on the calculated control difference variable, the valve is opened or closed by the thermostat. Thus, the thermostat can increase the amount of hot water to increase the amount of heat transmitted from the heater to the room air. Disturbances in this system are for example changes in the outdoor air temperature.
In contrast to closed-loop control, in open-loop control there is no feedback of the controlled variable to the controller.

Definition 2.3 ***Open-loop control*** *is a 'process whereby one or more variable quantities as input variables influence other variable quantities as output variables in accordance with the proper laws of the system' [IEC13a].*

Figure 2.2 presents a schema of an open-loop control system [IEC13a]. The terms and variables used are equally as in closed-loop control. The command variable (c) is an input to the control system and is processed by a controlling element. The controlling element generates the controller output variable (m), which is converted by an actuator to a manipulated variable (y). External disturbances (z) exist and can effectuate on the controlled variable (x) but are not evaluated by the controlling element for deriving the controller output variable. A feedback of the controlled variable to the controlling system is not realised.
An example for open-loop control is a heating curve [ISO05], which realises a constant-change of the feed temperature of a heating system dependent on the outdoor air temperature. The

Table 2.1: Description of abbreviated characters used in Figure 2.1.

Character	Description
c	Command variable
w	Reference variable
e	Control difference variable
m	Controller output variable
y	Manipulated variable
z	Disturbance variable
x	Controlled variable
q	Final controlled variable
r	Feedback variable
○	Summing point
●	Branching point

outdoor air temperature can, obviously, not be changed by the controller, but their is an effect on the heating system. Closed-loop and open-loop control can be combined, e.g. the constant change of the feed temperature of a heating system is combined with a closed-loop control of a thermostat.

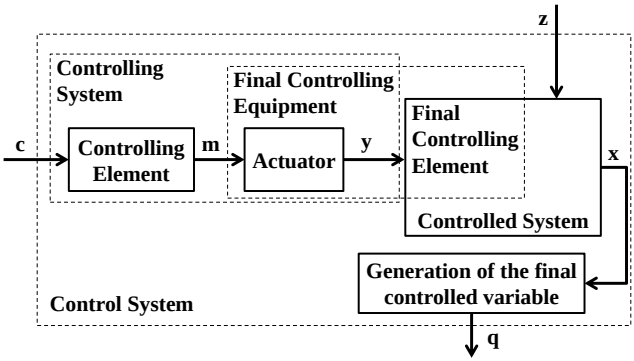


Figure 2.2: Structure and terms of a controller for open-loop control (adapted from IEC 60050-351 [IEC13a]). The final controlling element is defined to be part of the controlled system. Abbreviated characters are summarised in Table 2.1 and are described in the text.

In the automatic control domain the controller can be vertically and horizontally structured into distinct entities. A controller can be composed from several of these entities, which represent some encapsulated functionality. The entities can be connected to form larger networks for the automatic control of complex systems. Each of these entities has inputs and outputs as well as some means of processing, which performs the conversion from inputs to outputs.
Despite the definitions given in the IEC 60050-351, different terms are used in the domain when referring to such an entity, for instance, *actor* in generic actor oriented modelling [EJL+03], *function block* in IEC 61499 [IEC12], POU in IEC 61131-3 [IEC14a], *Block* from hybrid modelling and simulation [Mod17a], *controlling element* for open-loop control in IEC 60050-351 [IEC13a], and *controller* for closed-loop control in the standard IEC 60050-351 [IEC13a]. In this thesis the term *control actor* is used when referring to such an entity. This generalises the definition given by IEC 60050-351 as the definition introduced here also includes open-loop control.

Definition 2.4 *A **control actor** is an entity in a control system, which receives inputs and processes these inputs to determine its outputs [SSKD11]. The processing of the inputs to determine its outputs is defined by its control logic.*

As mentioned above several control actors can be connected vertically and horizontally to form control networks. This connection of control actors can have significant implications on the overall performance of the network. To avoid ambiguity in this regard the term *logical topology* is defined as follows:

Definition 2.5 ***Logical topology**: The logical topology refers to the connection of interfacing elements of control actors with the interfacing entities of other control actors or the connection of interfacing entities to datapoints [DCVK16].*

It should be noted that this definition goes beyond the term binding as defined by Domingues *et al.* [DCVK16], which refers to the connection of datapoints of distinct devices via the automation network.
How a control actor actually processes its inputs to determine its outputs is an important characteristic of each control actor. A plethora of methods exists for the control of technical systems. Various terms are used synonymically to refer to the actual behaviour of a controlling element, e.g. *control sequences* [CTM16, Dub11], *functional block* [IEC13a] or *function profiles* in BAS [ISO11]. To prevent ambiguity within this thesis the following term and definition is given when referring to the processing part, the *control logic*, of a control actor:

Definition 2.6 *The **control logic** in a control actor defines how information obtained from one or more inputs is utilised to determine one or more outputs.*

Typically control logic is differentiated into continuous control and non-linear control [Abe10]. The field of continuous control offers a theoretical rich set of methods and theories for the analysis, composition and prediction of the behaviour of the resulting control system [Abe10]. However, due to the linearity requirement, continuous control is often restricted to low-level control applications. Non-linear control includes a wide range of methods and formalisms spanning from simple two-point control [Abe10] to finite state machines [Lun09].
For the particular modelling of discrete control logic finite state machines are found to be a sufficient abstraction mechanism. Within automata theory finite state machines constitute a specialisation of the generic Turing machine [Lun09]. The basic building blocks for finite state machines used are the Mealy machine [Mea55] and the Moore machine [Lun09]. These have been mixed, adapted and extended to meet the specific needs of automation. Examples are GRAphe Fonctionnel de Commande Etapes/Transitions (GRAFCET) [IEC13c], SFCs [IEC14a] or Grafchart [VJS+98]. Not restricted to the automation domain are variants, such as state charts [Har87] and Unified Modeling Language (UML) [Obj15a] state machines. As mentioned, automatic control is a domain within automation systems. Automation systems combine physical entities, such as embedded devices, sensors and actuators, and virtual entities, such as control logic and, thus, are CPS. Two main application areas for automation technology, building and industrial automation, are introduced in the following sections.

2.1.2 Building Automation Systems

BAS evolved over the past decades to complex technical systems and are an important component of modern buildings. Emerging from hard-wired control units for Air Handling Unit (AHU), nowadays digital control and communication via buses represents the state of the art [SSKD11]. BAS encompass a number of services in residential and commercial buildings. These services aim to maintain the set comfort conditions for humans and enable a healthy indoor climate by controlling technical equipment, such as HVAC equipment or lighting [SSKD11]. Besides indoor comfort BAS also provide services related to safety, security, transportation, announcement systems, energy management, supply and disposal, communication and others [SSKD11]. BAS are a prerequisite for energy efficient operation of buildings and the highest rating with regard to the energy efficiency can only be achieved by deploying BAS [EN 17, USG16]. According to the definition given below building automation includes building control [MHH09].

Definition 2.7 ***BAS*** *are defined as 'Equipment, software and services for automatic controls, monitoring and optimisation as well as operation and management, and for the energy-efficient, economic and reliable human intervention of the building services' [VDI13].*

When deploying BAS a variety of goals are expected to be reached including reduced costs, reduced energy consumption, reduced wiring, increased indoor comfort and flexibility [MHH09].

The main concepts and cornerstones of BAS are rigidly defined by Domingues *et al.* [DCVK16] and will be introduced in the following.
The classical way to structure BAS is a three layered approach [ISO04b], where a field, automation and management layer are distinguished (see Figure 2.3). Typically the functions and capabilities of components at these layers increase in complexity starting at the field layer and ending at the management layer.

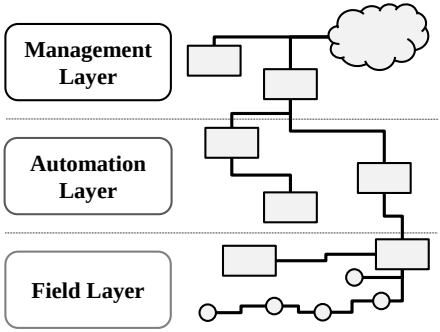


Figure 2.3: The differentiation of Building Automation Systems (BAS) into three layers according to the standard ISO 16484-2 [ISO04b]. Sensors and actuators (circles) at the field layer are connected (lines) via a computer network to BAS devices (rectangles), which also can be connected via the internet (cloud).

Starting at the *field layer* (bottom, Figure 2.3) sensors, actuators and controllers [DCVK16] are placed typically close to the actual associated equipment or space. At the *automation layer* embedded devices are placed, which connect to devices on the field layer as well as provide communication connections to the upper most *management layer*. Communication networks [DCVK16] connect the respective layers and a variety of protocols is used to enable the communication, for example the Building Automation and Control Networks, network protocol (BACnet) [ISO17a], Local Operating Network (LON) protocol [ISO12] and KNX (KNX) protocol [ISO07]. The control logic can be differentiated into low-level control logic, such as Proportional-Integral-Derivative (PID) control [Abe10] often deployed at the field and automation layer and supervisory control logic often deployed on the management layer. To avoid disambiguation the terms are defined as follows:

Definition 2.8 ***Low-level control logic*** *is control logic deployed on embedded devices, which are placed on site typically close to the actual piece of equipment. It processes a limited number of inputs with only simple information related to it, such as sensor measurements. The behaviour of low-level control logic can be changed by a supervisory control logic either by changing its input values or its parameters.*

Definition 2.9 ***Supervisory control logic*** *has access to input information that goes beyond simple sensor measurements possibly available from outside of the actual automation network. Supervisory control logic is not restricted to run on devices actually installed on site, often is placed at the management level and is, potentially, hosted somewhere else. Supervisory control logic provides information to low-level control logic by means of changing inputs, e.g. set points or parameters.*

However, it should be noted that the differentiation between low-level and supervisory control logic cannot always be clearly drawn.
During the evolution of BAS it became apparent that best practices on how to control a certain type of technical equipment need to be standardised to enable reuse and interoperability as well as comparability of vendors. Additionally, the procurement in public construction projects needs to be manufacturer-independent. Several national standards exist, e.g. the Germany VDI 3813 [VDI11a] for room automation, the German VDI 3814 [VDI09a], the German VDI 3525 [VDI07] or other national efforts (Hydeman *et al.* [HTE15]). Some of these national efforts are the basis for the international ISO 16484 family of standards for BAS [ISO11]. A more general overview on control logic types is published in Salsbury [Sal05]. Also, manufacturers provide best practices for the control of technical equipment, e.g. Honeywell [Hon97].

2.1.3 Industrial Automation Systems

A broad range of systems qualify as IAS. Here, the focus is set on automation systems, which produce goods and commodities, such as cars or consumer electronics [SSKD11]. A classical [FVC+17] approach to structure IAS is the automation pyramid as standardised in IEC 62264 [IEC13d] (see Figure 2.4). The bottom layer (*Level 0*) constitutes the field layer where sensors, actuators, drives, i.e. the production plant is abstracted. On the next layer *Level 1* the physical inputs and output (I/O) of the system as well as control functionalities executed by PLCs are placed. Supervisory Control and Data Acquisition (SCADA) is considered in the following layer *Level 2*, which constitutes of more sophisticated control and information processing tasks compared to Level 1. Systems executing tasks on Level 3 are termed Manufacturing Execution System (MES). These tasks involve coordination of production execution and management [FVC+17]. On the highest Level 4 long-term business activities of the enterprise including for example logistics are active [FVC+17], typically executed by an ERP system. Time scales vary significantly from milliseconds to minutes on Level 0 to 2 and from days to months on Level 3 and 4 [FVC+17]. The distinction into layers as defined in the IEC 62264 is one possibility for the structuring and other hierarchical approaches exist [SSKD11]. In particular, with the introduction of novel ICT the strict hierarchy from ERP to field level tends to dissolve more and more [FVC+17].

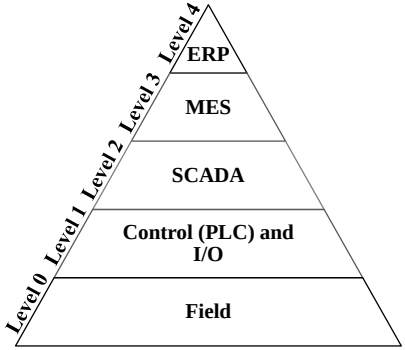


Figure 2.4: The structure of industrial automation systems into five hierarchically separated layers [IEC13d]. ERP - Enterprise Resource Planning, MES - Manufacturing Execution System, SCADA - Supervisory Control and Data Acquisition, PLC - Programmable Logic Controller, I/O - Input/ Output.

Scholars envision for future architectures a two level structure (e.g. Vogel-Heuser *et al.* [VHDB13]). The main idea inherited from the service-oriented architecture paradigm is the definition of a distributed service-based integration layer (see Figure 2.5). All software agents can access rel-

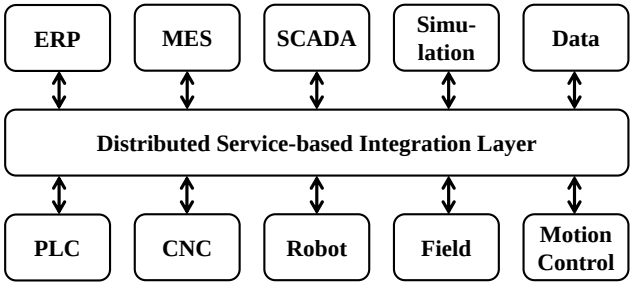


Figure 2.5: Future architecture of factory automation systems (adapted from Foehr *et al.* [FVC+17]). ERP - Enterprise Resource Planning, MES - Manufacturing Execution System, SCADA - Supervisory Control and Data Acquisition, PLC - Programmable Logic Controller, CNC - Computer Numerical Control.

evant services of components to provide or to retrieve information across hierarchies between the different components [FVC+17]. An example from automation systems engineering real-

ising such an architecture is the engineering service bus [NSMŠ15], which is analysed in detail in Section 2.3.

2.2 Foundations of Knowledge Representation

The knowledge-intensive processes related to the engineering and operation of automation systems require a deeper understanding on knowledge. In particular, a clear differentiation needs to be made among the terms data, information and knowledge (Section 2.2.1). Moreover, knowledge can be classified (Section 2.2.2) to identify, which part can be represented (Section 2.2.3) to be used in a Knowledge-Based System (KBS) (Section 2.2.4).

2.2.1 Data, Information and Knowledge

Often the terms data, information and knowledge are utilised synonymically in publications and documents. These terms need to be differentiated properly to avoid ambiguity. The guideline VDI 5610-1 [VDI09b] defines terms and concepts related to knowledge management in the domain of engineering and gives the following definitions for data, information and knowledge:

Definition 2.10 *'Data are objective facts, they cannot be interpreted without context and further backgrounds. They are to be taken as "raw material"' [VDI09b].*

Definition 2.11 *'Information are structured data with relevance and purpose, which can be put into a context, categorised, calculated and corrected.' [VDI09b].*

Definition 2.12 *'Knowledge is linked information, which enables to draw comparisons, to establish links and to make decisions.' [VDI09b].*

To illustrate and discuss the differences between data, information and knowledge an example is given in Figure 2.6. Following North [Nor98] the relationships in between the three entities can be described by a stair-like structure. However, it should be noted that this does not necessarily mean that there is an implicit hierarchy or quality difference among them [VDI09b].
In Figure 2.6 a time series of measurements of a temperature sensor is given. The data has a defined syntax as it is organised in columns and rows. In each cell characters are used to describe its content. To perform calculations or corrections on the data additional information is required, here presented as some meta-information provided along with the tabular data. This meta-data allows to put the tabular data into context, i.e. identify that the column headed by *OAT* is of type `Temperature` and has the unit `Celsius`. Now it would be possible to convert the observed values to other temperature units, e.g. Fahrenheit. The knowledge to

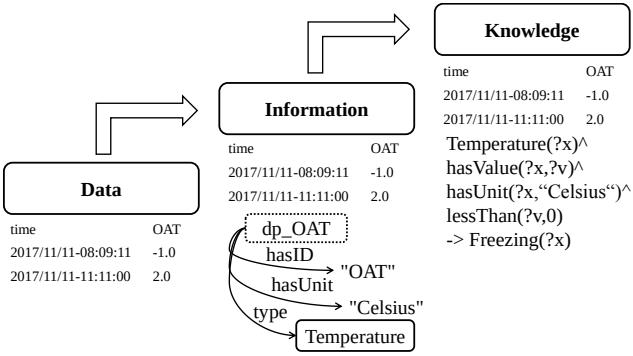


Figure 2.6: Example illustrating the difference between data, information and knowledge. Tabular data of a time series can be combined with contextual facts (e.g. unit is Celsius) to information for interpretation. Knowledge expressed as a rule allows to decide whether the freezing point of fresh water is reached or not (Stair like structure adapted from North [Nor98]).

identify if some measurement is below the point where fresh water freezes is formulated as a rule in Figure 2.6 (*Knowledge*). Hence, an entity having both the information and the described knowledge could decide whether a particular measurement is below the freezing point. To allow machines to utilise knowledge as defined by the rule KR is required as introduced in Section 2.2.3.

2.2.2 Knowledge Classification

A number of disciplines show interest in understanding and reusing the principles of knowledge-based processing of information artefacts. For instance, anthropologists and neurobiologists are interested to determine how individuals, both animals and humans, use knowledge for decision making. From a business management perspective knowledge is one of the most valuable assets companies have [TSD11].

In computer science expert systems, a sub-domain of Artificial Intelligence (AI), deal with the representation of knowledge in a knowledge base such that software programs can provide a response to questions or derive a decision based on this knowledge [BL04]. With respect to KR in the context of AI (see Section 2.2.3) it needs to be considered that different categories of knowledge exist and some cannot be externalised. According to Polanyi [Pol09] knowledge can be differentiated into *explicit* and *tacit* (also *implicit*) knowledge. This initial differentiation is revised in Nonaka and Konno [NK98] and Probst *et al.* [PRR12] in the context of a business perspective. As a result of these efforts the ASHEN model presented by Snowden [Sno05]

aims to overcome some of the identified limitations of the first models to classify knowledge. The ASHEN model is illustrated in Figure 2.7 and supports to identify knowledge suitable for KR. It provides useful means to distinguish between knowledge categories and to assess if some certain kind of knowledge can be specified for the purpose of KR in AI. It may be noted that approaches exist to convert implicit to explicit knowledge and vice versa [NK98]. The following categories can be distinguished [Sno00]:

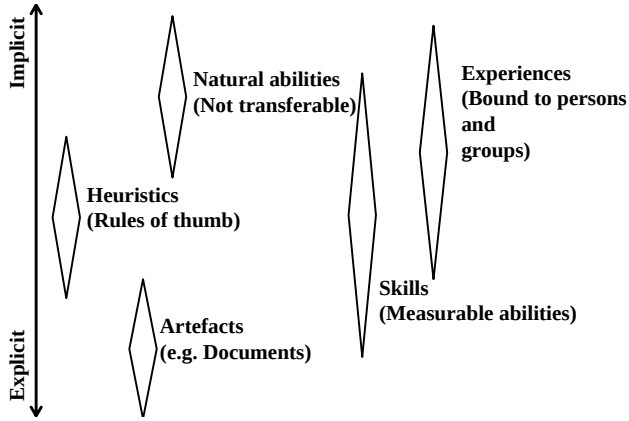


Figure 2.7: Schema of the ASHEN model [Sno00] to differentiate knowledge categories into implicit and explicit knowledge (adapted from VDI 5610-1 [VDI09b]).

- **A**rtefacts include all artefacts produced by humans, which might be codified to be machine readable. Typically a large share of this knowledge can be represented by means of KR. For example this includes technical drawings or manuals describing how a certain process works [Sno00];
- **S**kills are the abilities a person can acquire, e.g. through practising. The successful acquisition of skills can be measured by some quality measure [Sno00];
- **H**euristics or 'rules of thumb' [Sno05] are possibilities to derive decisions without complete information or with only a limited amount of time. This way of thinking might incorporate a simple set of questions, which are sequentially asked [Sno00];
- **E**xperiences consist of mainly tacit knowledge. It is considered more valuable above the other mentioned artefacts [Sno00] as experience in the context of an organisation is distributed across individuals and might not be replicated simply through time restrictions;

- **N**atural abilities or natural talent is the final component being intrinsically implicit knowledge, which is a characteristic of an individual or not. In particular, it cannot be manufactured or transferred [Sno00];

The classification scheme of the ASHEN model helps to identify, which knowledge is suitable for representation. In an engineering context artefacts, such as simulation files, drawings, contact plans or tables with component specifications or product data occur. Hence, according to the ASHEN model, these artefacts are particularly suitable for KR.

2.2.3 Knowledge Representation

The intelligent behaviour of humans or animals through the use of knowledge is a fascinating phenomenon. Researchers related to the field of AI are interested in developing software programs, which may use knowledge to deduce novel insights by means of reasoning [BL04].

Definition 2.13 *KR is 'the field of study concerned with using formal symbols to represent a collection of propositions believed by some putative agent.' [BL04].*

Hence, formal symbols are use to represent a collection of propositions, e.g. as embodied in the following sentence: *Georg studied mechanical engineering*. Within this sentence the symbols *Georg* are used to refer to the author of this thesis and the symbols *studied* to a verb. The proposition of this sentence is if the statement encoded in this sentence may be evaluated to be true or not. Brachman and Levesque [BL04] describe a putative agent as a generic abstraction for human persons as well as computer programs.
To enable computer programs to deduce new insights from symbolically represented knowledge *reasoning* is required. Here, reasoning refers to the formal manipulation of represented knowledge to produce new representations [BL04].
A different approach to define KR is given by Davis *et al.* [DSS93]. Instead of providing a single definition the authors describe five main roles of KR in the context of AI.

1. KR is a *surrogate* meaning that there is a need to have a surrogate for the real world to enable an entity to determine consequences through thinking. In contrast there could be also the possibility to simply act upon the real world [DSS93];
2. KR is associated with some *ontological commitment*. Davis *et al.* stress the fact that when representing the real world there always are decisions on what should be contained in the surrogate and what should be omitted. 'The commitments are, in effect, a strong pair of glasses that determine what we can see, bringing some part of the world into sharp focus at the expense of blurring other parts.' [DSS93];
3. KR involves the notion of *intelligent reasoning* meaning to generate new expressions from old. They conclude that a diverse set of intelligent reasoning methods exist and can be quite diverse dependent on the tools utilised;

4. Davis *et al.* make the claim for *pragmatically efficient computation*, where computation is used synonymically to thinking. To enable the use of KR methods and tools it needs to be computationally efficient. In particular, the need for guaranteed computational efficiency is made [DSS93];
5. KR is considered as 'a medium of human expression and communication'. Formally specifying knowledge means also specifying human expressions. The results of KR should be easily understandable, in particular, when results obtained from machines need to be communicated to humans again.

The actual technology used for representing knowledge should be separated from the theoretical understanding of KR. Different technologies exist for the symbolic representation of knowledge [PA16, BL04]. This includes logics such as First Order Predicate Logic (FOL) [HKR10], rules [HPSB+04, KB13], frames [Min74] and semantic networks [Sow14]. Each of these technologies has its benefits and drawbacks, which need to be considered when implementing solutions using one of them [BL04].

2.2.4 Knowledge-Based System

The original goal in formally representing knowledge is to enable intelligent applications through a KBS. Examples are the automated verification of compliance to standards [PVDV+11] or checking of the interoperability of devices for the automated design of BAS [Dib13]. In Figure 2.8 a generic architecture of a KBS is presented. The architecture is adapted and simplified from Runde [Run11].

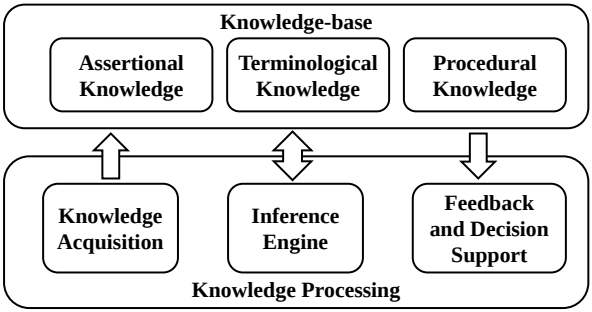


Figure 2.8: A generic architecture of a knowledge-based system (adapted, simplified from Runde [Run11]).

A KBS consists of a knowledge base and a knowledge processing component. The knowledge base stores the symbolical representation of knowledge. Following the ASHEN model (see

Figure 2.7) only explicit knowledge can be considered here. The stored knowledge can be distinguished dependent on the originating source [Run11, BHS07] as well as its use [Run11, BKI03, Sab16] into assertional, terminological and procedural knowledge.
Assertional knowledge constitutes knowledge, which often consists of facts on a specific use case. For example instantiating the individual `AHUController` as an individual of the concept `ControlActor` constitutes such knowledge. *Terminological knowledge* constitutes knowledge on an abstract level, which can be transferred to different use cases. This includes descriptions of the domains of interest, such as concepts and the relationships among them. Following the previous example relating the concept `ControlActor` via the `hasInput` relationship to a concept `Input` is an example for such knowledge. Finally, *Procedural Knowledge* includes knowledge, which can be represented using causal relationships, such as rules. Here the antecedent (*'IF'*) implies the consequence. Extending the previous example the rule *If the unit of an input and output are equal, then they can be connected* specifies some procedural knowledge, which could be used to infer the connection of `ControlActors`.
The components of the knowledge-processing module have access to the stored knowledge in the knowledge-base. Each knowledge type needs to be acquired by some processing component. For assertional knowledge this often requires the formalisation of use case specific instant data and information by implementing suitable adapters for automated processing. Terminological and procedural knowledge typically is defined manually by formalising the domain of interest, e.g. through ontologies or rules. A prerequisite for a KBS is the availability of an *inference engine*, which automatically draws conclusions on the available knowledge by means of reasoning. Finally, the feedback and decision support component communicates the inferred results of the KBS back to human agents or to technical systems.
After presenting the corner stones of KR in this section, Section 2.3 analyses the properties of contributions from literature, which use KR to develop KBS in the context of BAS and IAS.

2.3 Knowledge-based Methods Related to the Engineering and Operation of Control Logic in Automation Systems

As defined in Objective 1 one goal of this thesis is the development of knowledge-based methods to support and automate the engineering and operation of control logic in automation systems. During the engineering process of automation systems a variety of artefacts are created, which are particularly suitable for KR.
KR and semantic technologies are understood to provide means for the design of automated methods to solve knowledge-intensive tasks related to the engineering and operation of technical systems [TAO98]. This approach is particularly interesting in multi-disciplinary, hetero-

geneous engineering environments of automation systems and increasingly gets into the focus of industry and academia [BS16a, ESS+17].
From decades of research in computer science a wide variety of formalisms for KR is available. This includes amongst others logics, such as FOL [HKR10] or F-Logic [KLW95], rules [HPSB+04, KB13], frames [Min74] and semantic networks [Sow14]. Each of these technologies has its benefits and drawbacks, which need to be carefully balanced [BL04]. A recent contribution by Ekaputra *et al.* [ESS+17] analyses literature from semantic web and automation systems engineering research areas and their effect on ontology-based data integration. A finding of the authors is that the de facto standard for knowledge-representation in automation systems are ontologies and most works use Semantic Web Technologies (SWT) (see Chapter A), such as Web Ontology Language (OWL) [W3C12] for implementation. Nevertheless, other means for implementation exists such as frames and successful applications are reported by Wiesner *et al.* [WMM11] and Tudorache & Alani [TA16].
In the following contributions are analysed with respect to their potential to reach Objective 1, i.e. knowledge-based methods to support and automate the engineering and operation of control logic in automation systems. The contributions are analysed with respect to the following properties and the results are summarised in Table 2.2. Finally, findings drawn from this analysis are presented in Section 2.5.

- the applied KR methodology;
- the used knowledge modelling language;
- the utilised technology for reasoning;
- the means for knowledge acquisition;
- if reported, the utilised knowledge engineering methodology
- the potential reuse of existing formal knowledge, e.g. through ontology reuse.

An interesting piece of research using knowledge-based methods for the engineering of automation system allow the automated design [DPK10, Dib13] and the semi-automated engineering and commissioning [RFW09, Run11] of control logic in BAS.
Dibowski & Kabitzsch [DPK10] and Dibowski [Dib13] present a knowledge-based approach to automatically design room automation systems. The approach relies on the definition of a semantic model for BAS devices and BAS function profiles. It allows to define requirements to the automation functionality of a room and then the presented solution derives automatically a logical topology. In addition it matches the resulting network of function profiles to devices and generates, ultimately, interoperable, multi-vendor BAS. A related contribution presents the concept of KBS to support the engineering of automation systems [RFW09, Run11]. The approach describes the semi-automatic configuration of BAS devices based on matching requirements with actual functionalities of the devices.

Table 2.2: Results from the analysis of knowledge-based methods related to the engineering and operation of control logic in automation systems. KR - Knowledge Representation, KE - Knowledge Engineering, OWL - [W3C12], Semantic Web Rule Language (SWRL), SPARQL Protocol and RDF Query Language (SPARQL), Pellet - [SPG+07], JESS - [Hil03], Hermit - [GHM+14], ifcOWL - [PT16], BAS - Building Automation Systems, UML - Unified Modeling Language.

Reference	KR methodology	Knowledge modelling language	Reasoning	KE methodology	Ontology reuse/ import	Application area
[Dib13]	Ontology	OWL/ SWRL/ SPARQL	Pellet, Query engine	own	-	Automated design of room automation systems
[Run11]	Ontology	OWL/ SWRL	JESS	own	-	Support of BAS engineering and commissioning
[DHR16b]	Ontology	OWL/ SWRL/ SPARQL	Pellet/ Query engine	own	ifcOWL	Automated setup of virtual BAS sensors
[HK13]	Ontology	OWL/ SWRL	Pellet	own	-	Control application engineering
[MB12]	Ontology	OWL/ SWRL/ SPARQL	Query engine	own	-	Integrated knowledge sharing in the automation systems engineering life cycle
[HWD+13]	Ontology	OWL/ SWRL	Pellet	own	-	Verification of the semantics of UML behavioural models
[KP15]	Ontology	OWL/ SWRL	Pellet/ HermiT	own	-	Verification of UML models

Both approaches follow an ontology-based approach for the KR and implement their domain models using OWL. Due to the complexity [Dib13] of the expressed knowledge Dibowski [Dib13] determines interoperable BAS solutions by defining integrity constraints in the SPARQL [PS17]. Additional knowledge is specified using the SWRL [HPSB+04]. For reasoning the Pellet reasoner [SPG+07] and a query engine for the processing of the SPARQL queries is used. Runde [Run11] follows a different approach. In addition to knowledge expressed using OWL, Runde [Run11] uses rules defined in the SWRL. To reason upon the expressed knowledge he uses the JESS reasoning engine [Hil03]. No explicit knowledge engineering method is mentioned. Both develop novel ontologies from scratch without explicitly mentioning ontology reuse.

Contributions related to automate tasks needed for FDD in BAS are presented in Dibowski *et al.* [DHR16b, DVHR16, DHR16a], where, with regard to control logic, the automated setup of virtual sensors in BAS [DHR16b] is particularly interesting. The methodology and designed KBS of the respective approaches are similar. Here, the analysis is based on the descriptions related to the automated setup of virtual sensors [DHR16b]. The approach specifies formally the needs for setting up a virtual sensor for FDD. This specification can be matched against a formal domain model of the building and its technical equipment to determine, which virtual sensors can be instantiated automatically. The necessary knowledge is represented via ontology and implemented using OWL and SWRL. The matching procedure is realised by executing SPARQL queries on the processed knowledge-base. Additionally, reasoning is performed using the Pellet [SPG+07] reasoner for consistency checking of the existing knowledge-base. No specific knowledge engineering method is described. The approach reuses the ifcOWL [PT16] ontology.

A KBS presented by Hästbacka & Kuikka [HK13] supports the engineering of control application development through formal semantics provided by ontologies. It allows to check interface semantics or to identify potential control interlocks. The approach formalises a UML profile using an ontology implemented in OWL and additional procedural knowledge is implemented using SWRL. In the approach reasoning is performed using the Pellet reasoner and the reuse of existing ontologies is not described.

The Engineering Knowledge Bus (EKB) [MB12, Mos16] sets out to overcome the heterogeneity of formats and tools used for the general engineering of complex mechatronic systems, such as automation systems. The approach proposes the development of abstract domain ontologies for the mechanical, electrical and software (control) domains. Tools within these domains are integrated via tool specific ontologies. The ontologies described are implemented using OWL and additional rules using SWRL. Additional expert knowledge is encoded as SPARQL queries, which are executed against the knowledge base to ensure for example model consistency and an own structured knowledge engineering approach is followed. The reuse of existing ontologies is not detailed in the approach.

He *et al.* [HWD+13] present an approach, which utilises the formal basis of SWT for verification purposes of control logic. The approach relies on the specification of UML behavioural models including state invariants for the verification. A mapping of UML models and UML meta models is defined and the formal semantics allow, among others, to detect inconsistencies between different behavioural models, such as activity and sequence models. An ontology-based approach is used for KR implemented using OWL and SWRL. The Pellet reasoner is utilised to perform the inference task and no use of a knowledge engineering method and no reuse of existing models is reported.
To support model-driven engineering of software based on the UML Khan and Porres [KP15] present a logic-based approach for the validation of designs. The approach follows a similar path as presented by the authors He *et al.* [HWD+13] and allows to check the consistency of of UML class, object and state machine models. The authors provide translations to convert from a UML model to OWL. The approach is based on representing knowledge using ontology, where for implementation OWL and SWRL are used. The off-the-shelf Description Logics (DL) reasoners Pellet and HermiT [GHM+14] are used for the reasoning and neither the knowledge engineering method nor the reuse of existing ontologies is described.
A finding of this review, for the specific focus area of knowledge-based methods for the support of control logic engineering in automation systems, is that all found knowledge-based approaches use ontology for KR and implement their solutions using SWT. Different knowledge modelling languages and reasoners are used by the reported approaches and all do not specify a specific knowledge engineering methodology. This can be caused by the fact that SWT are the de facto standard when implementing KBS in automation systems engineering [BS16b, ESS+17].
All approaches review existing ontologies in the domain and discuss their benefits and shortcomings. Hence, a shared understanding of used terminology and knowledge can be assumed. However, the explicit knowledge reuse, e.g., through the import of an existing ontology is not reported by most approaches. Only [DHR16b] report on the reuse of the ifcOWL ontology [PT16]. A reason for this can be that the reuse of existing conceptualisations is a cumbersome and time-consuming process [Sch17]. Most alignment work needs to be down manually and methods automating this process are still under research [SE13].

2.4 Formats and Models for Automation Systems

This section reviews existing formats and models related to the modelling of BAS and IAS. The cited contributions are analysed and evaluated in terms of their ability to fulfil the criteria listed in Table 2.3. The criteria are defined with respect to reach Objective 2 of this thesis: develop-

ment of a semantic model of control logic in automation systems. Each referenced contribution is analysed whether it provides means to describe the following domains of interest:

- **Physical system**: Components, aspects and parts of physical systems, such as technical building equipment, valves, damper flaps, buildings, plants, pneumatic cylinders, conveyors, etc.;
- **Physical devices**: Real physical devices in an automation system, such as a temperature sensor mounted to a wall or a rail-mounted embedded device;
- **Interface entities**: The interfacing entities of a control actor[1], such as input, output or parameters;
- **Interface semantics**: Further annotation of interface entities through, e.g. its unit, quantity, medium or basic data type;
- **Logical topology**: Means to describe the logical topology of an automation solution[2];
- **Logical hierarchy**: Often a hierarchy can be established among several control entities, where some supervisory control actor supervises a low-level controller[3] by, e.g. changing its setpoint [ISO04b, IEC13d];
- **Control logic**: Core interest of this thesis is to analyse, whether contributions exist to explicitly specify control logic[4] apparent in BAS and IAS. This criterion evaluates positively, if this is technically possible by the evaluated approaches;
- **Data format**: The underlying data format and implementation language used for the specification for the respective format or model is reported. For example, Comma Separated Value (CSV), eXtensible Markup Language (XML) [BPSM+06] or OWL.

In the following sections contributions related towards the description and modelling of automation systems are analysed. First, in Section 2.4.1 and Section 2.4.2 data formats and ontologies from the domain of BAS are analysed. Next in Section 2.4.3 and Section 2.4.4 the respective contributions from the domain of IAS are analysed. Finally, in Section 2.4.5 and Section 2.4.6 generic ontologies and other approaches related to the topic are reviewed.

2.4.1 Data Formats for Building Automation Systems

Within this section existing data formats related to the information exchange between stakeholders from the BAS domain are analysed. Moreover, fundamental conceptualisations of the BAS domain and the definition of associated knowledge [DCVK16, CTM16] are reviewed.

[1] See Definition 2.4.
[2] See Definition 2.5.
[3] See Definition 2.8 and 2.9
[4] See Definition 2.6.

Table 2.3: Criteria for the analysis of related work; Criterion fulfilled (+), criterion partly fulfilled (○), criterion not fulfilled (-). CSV - Comma Separated Value, OWL - [W3C12]. Layout adapted from Dibowski [Dib13].

Criteria	**Possible evaluation**
Physical system	+ / ○ / -
Physical devices	+ / ○ / -
Interface entities	+ / ○ / -
Interface semantics	+ / ○ / -
Logical topology	+ / ○ / -
Logical hierarchy	+ / ○ / -
Control logic	+ / ○ / -
Data format	for Example: CSV, OWL, ...

The findings on analysing data formats and approaches for the BAS domain are summarised in Table 2.4.

Table 2.4: Results from the analysis of data formats related to the modelling of building automation systems. EXPRESS - [ISO04a], XML - [BPSM+06], OWL - [W3C12].

Criteria	Schein [Sch07]	Domingues *et al.* [DCVK16]	BCK [CTM16]	CDL [Wet18]	Benndorf *et al.* [BRCR17]
Physical system	-	-	-	-	+
Physical devices	+	+	-	-	+
Interface entities	+	+	+	+	+
Interface semantics	-	+	○	○	○
Logical topology	+	+	+	+	+
Logical hierarchy	+	+	+	-	-
Control logic	-	+	+	+	+
Data format	EXPRESS	None	Data base	Mode-lica	XML/ (OWL)

A domain model for BAS using the EXPRESS [ISO04a] data modelling language is presented by Schein [Sch07]. Using the model, it is possible to describe the building automation network and respective connections of devices. The model includes concepts to describe control actors and their logical topology as well as the hierarchy among them. The interface semantics of inputs and outputs can be specified using strings and a naming convention with limited expressibility. A downside of this methodology is that the naming convention provides the risk to introduce ambiguity and limits the maintainability of the approach. Actual control logic of a control actor cannot be described explicitly.

The prevalent heterogeneity among solutions for building automation is a major challenge in developing these systems. The 'unclear definitions and terminology' as mentioned by Domingues *et al.* [DCVK16] provided in literature and documentations of BAS are recognised as one source of this heterogeneity [DCVK16]. To address this issue the authors describe the basic building blocks of BAS including devices in BAS network, the interface and its semantics of BAS device along virtual datapoints, the connection of these datapoints into a logical topology and the hierarchy among them. Also, a textual description of control logic, such as schedules is provided. The valuable contribution of this work is the rigid definition of the basic building blocks of BAS. However, in this evaluation the absence of a formal model, though not the goal of the authors, is a limitation of their work.

Building Control Knowledge (BCK) as defined by Chen *et al.* [Che15, CTM16] refers to 'information that conveys the functionality, detailed control logics, algorithms, sequences, programming code, and the hierarchical structure of a (number of) controller(s)' [CTM16]. The domain knowledge related to the control of technical equipment in buildings is discussed on an abstract level by this contribution. The logical topology and hierarchy of control logic entities is mentioned and the authors conclude that control logic is manufacturer dependent and served in various formats and programming languages. The approach relies on a data base schema termed *M-BCK* [Che15], which provides means to exchange the specified knowledge but is not a formal model.

The OpenBuildingControl project [Wet18] constitutes an effort in developing and unifying the existing best-practice control sequences (control logic) for air handling systems in buildings. The goal of this efforts is to provide highly energy efficient control logic to stipulate its reuse for the operation of the controlled HVAC equipment. To exchange the specified control logic the effort aims at developing the *Controls Description Language (CDL)* [Wet18] and the model and documentation are available on the web. The definition of the CDL is based on the modelling language Modelica [Mod17a], which is an acausal modelling language for the simulation of technical systems and does not support KR. The scope of the approach is solely on control logic and their interfaces restricted to air handling systems in building HVAC. The connection of control actors can be described but not the hierarchy among them.

An attempt to extend the Industry Foundation Classes (IFC) [ISO13] model to specify commonly occurring control logic in the control of HVAC systems is presented in Benndorf *et al.* [BRCR17]. Via custom property sets [ISO13] the authors implement ways to model a heating curve, a time schedule and a PID temperature control. As the approach relies on the IFC it allows to describe buildings and their technical systems as well. Additionally, the authors claim to use reasoning for formal verification and exposing the data through a triple store, however, the evidence provided is limited.

2.4.2 Ontology-based Modelling of Building Automation Systems

Within the domain of building automation a number of contributions exist, which use ontology-based modelling for the description of the BAS domain. A recent overview is presented in [BGT17], where relevant and additional references are analysed in the following. In addition, a review on the use of SWT in the built environment is presented in Pauwels *et al.* [PZL17].

The *Haystack Tagging Ontology (HTO)* [CKAK15] is an approach to formalise knowledge on meta-describing datapoints in BAS and building management systems. A conversion method is provided to convert the tag system developed by the contributors to the Project Haystack[5] effort to HTO. The project aims at providing additional semantics to building monitoring data for the IoT. HTO provides means to describe technical equipment in buildings and their interfaces via datapoints in BAS. It is possible to specify, e.g. the quantity of a datapoint. Specific aspects of control logic are not within the scope of this contribution, such as logical topology and hierarchy as well as the explicit modelling of control logic.

The *Datenmodell- und Austauschformat für das Engineering in der Gebäudeautomation, (eng.: Data model and exchange format for the engineering of BAS) (DEG)* is a format and model specifically designed to support the engineering of BAS through knowledge-based methods. The approach is used to support and partially automate the commissioning of BAS [RFW09, Run11]. It allows to describe physical systems of a building and devices in BAS. Input and outputs of a control actor can be described and the logical topology among them but not hierarchical relationships among them. No detailed semantics of the interface entities can be specified. For the description of control logic a taxonomy is used and no explicit descriptions of control logic is possible.

To address interoperability issues arising from the heterogeneous communication protocols available in BAS, Reinisch *et al.* [RGPK08] describe an ontology-based integration approach. The work includes a description of control actors and their interfaces as 'function blocks' and uses a taxonomy to differentiate between them. The logical topology of several function blocks as well as annotations of the input and output variables can be defined. No means are provided to describe the logical topology and explicitly control logic.

A facility management model specifically designed for the integration of heterogeneous information sources required for energy management as developed in the *CASCADE ICT for Energy Efficient Airports (CASCADE)* research project is presented in Tomašević *et al.* [TBB+15]. Concepts and relationships to describe buildings, technical equipment and components of the BAS are modelled. Interface entities, their semantics and topology can be defined. However, control logic cannot be explicitly modelled.

[5] `https://project-haystack.org/`, Last accessed: 22 October 2018

For the modelling of energy systems the *Energy System Information Model (ESIM)* [KS15] is defined, which contains a submodel to describe BAS. Similar to the former reported approaches the focus lies on the description of aspects of physical systems and control actors. For the description of control logic a taxonomy is utilised and the explicit description of control logic is not possible.
The findings from analysing contributions related to the ontology-based modelling of BAS domain so far are summarised in Table 2.5.

Table 2.5: Results (1/3) from the analysis of ontologies related to the modelling of building automation systems. OWL - [W3C12].

Criteria	HTO [CKAK15]	DEG [Run11]	Reinisch *et al.* [RGPK08]	CASCADE [TBB+15]	ESIM [KS15]
Physical system	+	+	-	+	+
Physical devices	+	+	+	+	+
Interface entities	+	+	+	+	+
Interface semantics	○	○	+	+	○
Logical topology	-	+	+	+	+
Logical hierarchy	-	-	-	-	-
Control logic	-	-	-	-	-
Data format	OWL	OWL	OWL	OWL	OWL

The *Domotic Ontology (DogOnt)* [BC08] is an ontology for intelligent domotic devices. It sets out to serve as a domain model for smart home systems. A main distinction made by the modellers is the description of controllable and uncontrollable entities in a smart home. Hence, the ontology allows to describe the physical components of a home. Control logic descriptions are modelled by developing a taxonomy of functions, rather then explicitly describing control logic. These then are associated with devices, which execute the respective functionality. The current operation mode of a device can be specified from a finite set of states. The description of the interfaces of control actors and their semantics is not possible. Hence, the logical topology and hierarchy of control actors cannot be specified. Explicit modelling of control logic is not considered.
Extending on the work of DogOnt the *ThinkHome* project [RKK10, RKIK11, KRK12] constitutes a family of ontologies for energy related intelligent applications in smart homes. In addition to smart devices, it covers adjacent domains of interest, such as buildings, weather, energy and comfort conditions. The ontologies provide a strong means for describing the physical artefacts associated with the smart home domain. However, with respect to control logic only classifications can be utilised. Hence, knowledge related to the interfaces, logical topology and explicitly the actual control logic cannot be specified.

Driven by industry a reference ontology for smart appliances termed *Smart Appliances REFerence Ontology (SAREF)* has been developed [DdHR15]. In the development process existing ontologies related to the smart home domain are reviewed and amalgamated to the resulting SAREF core ontology. The initial version [DdHR15] is revised and extended into various domains [PVGC18, ETS17b]. A revision of the initially published version of SAREF is standardised in European Telecommunications Standards Institute (ETSI) TS 103 264 [ETS15]. Domain specific extensions are developed to describe physical systems from the buildings [PVGC18] and energy [ETS17b] domains. Similar to the above reviewed ontologies, e.g. DogOnt, ThinkHome, etc., which can be seen as predecessors of SAREF, it focusses largely on the description of devices and physical components in smart home systems and appliances. To describe actual control logic a taxonomy of functionalities, such as actuating, sensing or metering can be used. Interface entities and their semantics as well as knowledge related to the logical topology cannot be specified. In particular, control logic knowledge cannot be explicitly modelled.

To enable the automated design of BAS a generic semantic device description model is presented in Dibowski and Kabitzsch [DPK10] and Dibowski [Dib13]. The main contribution of the work includes a semantic model for devices in BAS as well as function profiles. The models are a module in *BASont* [PHDK12], a layered ontology to formalise the BAS domain. BASont provides a number of concepts and relationships to describe the physical aspects of a building including buildings, technical equipment in buildings, BAS devices, etc. It allows to specify in detail the interface entities and semantics as well as the logical topology and hierarchy of the specified control functions. The approach uses a finite set of potential function profiles derived from a standard for room automation [VDI11b] to describe the control logic in BAS. The actual control logic cannot be explicitly modelled.

The results of analysing the mentioned ontologies are presented and can be compared in Table 2.6.

Table 2.6: Results (2/3) from the analysis of ontologies related to the modelling of building automation systems. OWL - [W3C12].

Criteria	DogOnt [BC08]	ThinkHome [RKIK11]	SAREF [DdHR15]	BASont [PHDK12]
Physical system	+	+	+	+
Physical devices	+	+	+	+
Interface entities	-	-	-	+
Interface semantics	-	-	-	+
Logical topology	-	-	-	+
Logical hierarchy	-	-	-	+
Control logic	-	-	-	-
Data format	OWL	OWL	OWL	OWL

Balaji *et al.* [BBF+16, BBF+18] present the *BRICK* ontology, which sets out to provide a unified meta data schema for buildings. It particularly focusses on describing the concepts and relationships of buildings, the technical equipment and datapoints exposing sensor measurements and actuator settings in BAS. The schema provides concepts to describe the logical topology of automation solutions. However, apart from the description of the semantics of control actors including the interfaces and the topology, no explicit description of control logic is included.

A widely accepted, open exchange format in the domain of AEC/FM are the IFC. The format is developed by buildingSMART[6] a non-profit organisation of experts from academia and industry. The released versions of the format are internationally standardised in ISO 16739 [ISO13]. The format is defined using the EXPRESS modelling language [ISO04a]. A version converted to OWL (*ifcOWL*) is available as well [PT16]. Within its most recent release IFC4 the standard provides some support to describe control logic in BAS. This includes the logical topology of automation solutions. The respective control logic is not included and a taxonomy is given to specify the type of a control logic.

The results of analysing the mentioned ontologies are presented and can be compared in Table 2.7.

Table 2.7: Results (3/3) from the analysis of ontologies related to the modelling of building automation systems. OWL - [W3C12].

Criteria	BRICK [BBF+18]	ifcOWL [PT16]
Physical system	+	+
Physical devices	+	+
Interface entities	+	+
Interface semantics	○	○
Logical topology	+	+
Logical hierarchy	-	-
Control logic	-	-
Data format	OWL	OWL

2.4.3 Data Formats for Industrial Automation Systems

A number of data formats exists for the exchange of information between software tools of automation engineers and control logic designers in the industrial automation domain.

[6] `https://www.buildingsmart.org/`, Last accessed: 22 October 2018

The *AutomationML* format is a XML based format, which is developed by an industrial consortium (AutomationML e.V.[7]). It is internationally standardised as IEC 62714-1 [IEC14b], and a detailed description can be found in Lüder *et al.* [LSD17]. It is build on top of the Computer Aided Engineering Exchange (CAEX) format [IEC16], which allows to specify the hierarchical structure and functions of manufacturing processes and plants, and exchanges various information about them [FD04]. Within AutomationML the PLCopen XML [PLC09] format is used to describe control logic of automation systems. The PLCopen XML format is an open, XML-based format developed by the non-profit PLCopen organisation[8]. It offers means to exchange PLC projects implemented in one of the five programming languages of the IEC 61131-3 [IEC14a]. Using the format, the logical topology of an automation solution and some semantics including the basic data types of the interface variables can be specified. The format includes the actual implemented control logic. For implementations using the textual programming language Structured Text (ST) the bare source code is stored.
Lüder *et al.* [LEHM11] present the transformation of Gantt charts, PERT charts, impulse diagrams, UML state machines and SFCs from IEC 61131-3 via an *Intermediate Modeling Layer [LEHM11] (IML)* meta model. Mapping rules are defined between the respective domain models to the IML meta model. Target format for exchange is the PLCopen XML format. Besides control logic the model is used to describe discrete manufacturing processes. The model forms the basis of a framework, which is used to evaluate the approach in studying a drill mechatronic unit [LEHM11]. The approach is connected to the development of the AutomationML and PLCopen XML standards with similar capabilities when combined.
To support engineers in developing control logic in automation systems the *MeiA●* method [ASB+18] is proposed. It implements an incremental development process, which allows different stakeholders involved in the design of the control logic of an automation system to collaboratively work on the analysis, design and implementation of the system. Aspects of physical systems related to the control logic in an automation system are not within the scope and cannot be modelled explicitly. The descriptions of control actors with interfaces is possible, however, the detailed interface semantics and hierarchy among control actors are not included. The format used in the methods allows the description of a single type of control logic using SFCs from IEC 61131-3.
The *vueOne* integrated development platform for the engineering and simulation of cyber-physical automation systems is presented by Harrison *et al.* [HVA16]. Within the system a number of tools and services are available, which exchange engineering information via an XML based format. The final result in terms of control logic can be generated as source code to be executed on a real PLC. The approach focusses on the design of the physical systems and control logic is considered more in the direction of source code implementation. An actual

[7] `https://www.automationml.org`, Last accessed: 22 October 2018
[8] `http://www.plcopen.org/`, Last accessed: 22 October 2018

explicit formal description of control logic is not provided. The description of SFC according to IEC 61131-3 is possible and the source code of these can be generated and exchanged.
Horn & Ebert [HE08] model all five languages of the IEC 61131-3 using the UML. They provide similar capabilities as the PLCopen XML format, but, additionally, support to specify source code implemented in the language ST [IEC14a]. The scope of the described meta-model is control logic and its interface entities. No specific semantics of the interface entities is described and discrete control logic, such as finite state machines can be specified. The approach reuses the IML model [LEHM11] for the representation and exports the result to an SFC specified in the PLCopen XML format.
The findings with respect to the data formats in IAS are summarised in Table 2.8.

Table 2.8: Results from the analysis of data formats related to the modelling of industrial automation systems, XML - [BPSM$^+$06], UML - [Obj15a].

Criteria	Automa -tionML [IEC14b]	IML [LEHM11]	MeiA● [ASB$^+$18]	vueOne [HVA16]	Horn & Ebert [HE08]
Physical system	+	-	-	+	-
Physical devices	+	-	-	+	-
Interface entities	+	-	+	+	+
Interface semantics	○	-	-	-	○
Logical topology	+	+	+	+	+
Logical hierarchy	-	-	-	-	-
Control logic	+	+	○	○	+
Data format	XML	XML	XML	XML	UML/XML

2.4.4 Ontology-based Modelling of Industrial Automation Systems

The use of ontology for the modelling of IAS has gained attention in academia and industry. A comprehensive overview on generic ontology-based data integration approaches in the domain is given in Ekaputra *et al.* [ESS$^+$17] and Biffl & Sabou [BS16b].
To improve the energy management of production systems the *SERENE* model [WJRO14, Wic16] allows the ontology-based integration of knowledge on a production plant. The model supports the description of knowledge on physical objects of a production plant including machines, tools, engineering resources as well as engineering processes, industrial automation devices and human actors. The focus of the approach is related to the energy management in production plants. With respect to the interfaces of control actors and their logical topology and hierarchy no knowledge can be specified. Additionally, the explicit modelling of control logic in IAS is not within the scope of the approach and, hence, not included.

The current standardised version of AutomationML [IEC14b] is implemented in XML. Researchers are aiming in converting it into the formal knowledge representation language OWL [KGG18]. The approach up to the time of writing focusses on the very top layer of the AutomationML format without going into detail, e.g. how to formalise the embedded PLCopen XML format. Hence, the evaluation is similar to the XML-based version. A detailed description is as well provided in Sabou *et al.* [SKN16]. As the ontology only captures the top-level of the AutomationML format, no explicit formal description of control logic is possible.

The *Common Concepts Ontology (CCO)*[9] is designed to serve as a domain ontology for mechatronical systems. It differentiates between mechanical, electrical and software engineering domains on a high level and may be extended with domain specific ontologies. Its structure and domains are detailed in Sabou *et al.* [SKN16]. It covers all domains on an upper level, however the detailed explicit specification of control logic is not described.

The *Automation Ontology (AutomOnto)* [NSMŠ15] is an ontology developed to support the simulation based engineering of production systems. It considers the physical plant, simulation, tag and parameter domains to allow the integration of virtual simulated pieces of the plant and physical components. All components are integrated in the *automation service bus*. The topology of control actors can be specified as well as its interfaces and their semantics. Control logic itself cannot be explicitly specified and is not discussed in the case of simulation.

An approach presented by Hästbacka & Kuikka [HK12, HK13] allows to enhance the engineering and reasoning on meta models to support the development of control applications by means of formal semantics through ontologies. The approach formalises the *UML Automation Profile (UML AP)* [RK07], an automation systems specific profile for the UML. The described ontology prototype allows to describe control actors and their interface entities including their semantics. The connection of interface entities and the hierarchy among control applications can be defined. In the approach the actual control logic is not specified in detail; rather function blocks representing 'parametrizable functionality provided by devices or subsystems' [HK13] can be modelled.

The findings of the above analysed contributions are summarised in Table 2.9.

Focussing on automation systems in the process industry the data model defined in standard *IEC 61512-3* [IEC08] strives for being a single domain model for the definition of batch processes in process automation including their control logic. Ontology-based formalisation of the standard is proposed [VH+14a, MEP10, HZ16] and allows to formally specify the respective physical systems, the interface entities and the logical topology of associated control actors. Interface semantics and the logical hierarchy are not detailed. The standard only allows the explicit description of a specific type of control logic, which complies to some recipe type of control logic using a finite set of states.

[9] `http://data.ifs.tuwien.ac.at/engineering/cco`, Last accessed: 22 October 2018

Table 2.9: Results from the analysis of ontologies related to the modelling of manufacturing automation systems. OWL - [W3C12].

Criteria	SERENE [Wic16]	AutomationMLOWL [KGG18]	CCO [SKN16]	AutomOnto [NSMŠ15]	UML AP [HK13]
Physical system	+	+	+	+	-
Physical devices	+	+	+	+	-
Interface entities	-	+	+	+	+
Interface semantics	-	○	-	+	+
Logical topology	-	+	+	○	+
Logical hierarchy	-	-	○	-	+
Control logic	-	-	-	-	○
Data format	OWL	OWL	OWL	OWL	OWL

To exchange information during the engineering of chemical process engineering the *Ontology for Computer-Aided Process Engineering (OntoCAPE)* [MMWY09] is a family of ontologies, which is developed to serve this purpose. The model provides basic building blocks of technical systems in general and then further details the descriptions down to a granularity level relevant to process automation. The focus of the ontologies is on exchanging engineering information on process engineering systems and includes for example the capability to describe differential equations required for the simulation of components. Aspects of the interfaces and topology/hierarchy of control actors can be defined. With respect to the explicit modelling of control logic it follows a taxonomy based approach, where a number of classes are specified to describe reoccurring control logic types in process automation.
An upper ontology to formalise the data model developed within the standard *ISO 15926* [ISO03] is presented in Batres *et al.* [BWL+07]. Again and similar to the previous reviewed approaches from process automation a taxonomy is established to describe the various types of control logic usually occurring in process automation domain.

2.4.5 Domain Independent Ontologies for Automation Systems

A number of domain independent ontologies exist, which cover aspects of the automation system domain. The respective ontologies are analysed in the following and the results are summarised in Table 2.11.
Researchers and practitioners in the semantic web domain have recognised that the description of sensors and actuators is a reoccurring pattern, which can be observed in various domains. Hence, the Semantic Sensor Network Ontology (SOSA) [HKC+17], a World Wide Web Con-

Table 2.10: Results from the analysis of ontologies related to the modelling of process automation systems, XML - [BPSM+06], OWL - [W3C12].

Criteria	IEC 61512-3 [IEC08]	OntoCAPE [MMWY09]	ISO 15926 [BWL+07]
Physical system	+	+	+
Physical devices	+	+	+
Interface entities	+	+	+
Interface semantics	-	+	-
Logical topology	+	+	+
Logical hierarchy	-	+	-
Control logic	○	-	-
Data format	XML/OWL	OWL	XML/OWL

Table 2.11: Results from the analysis of generic ontologies related to the modelling of automation systems. OWL - [W3C12].

Criteria	SOSA [HKC+17]	SEAS [LKGZ17]	oneM2M [TS 18]
Physical system	○	+	-
Physical devices	+	+	○
Interface entities	+	+	+
Interface semantics	-	+	-
Logical topology	○	+	-
Logical hierarchy	-	-	-
Control logic	-	-	-
Data format	OWL	OWL	OWL

sortium (W3C) recommendation, provides standardised concepts and relationships to describe sensor and actuator data on the web. Sensors and actuators form a part of each automation system and *SOSA* provides concepts and relationships for their description and its particular strength lies in the ability to serve as a top level ontology for this purpose. The ontology is defined independently of any application domain, such as automation systems. The general concept of a `Procedure` with inputs and outputs can be seen as a high level abstraction of a process converting inputs to outputs, e.g. control actors. However, the specification of how such a procedure actually behaves is not possible.

The *Smart Energy Aware Systems (SEAS)* ontology [LKGZ17] represents a family of ontologies, which formally describe technical systems on a generic level. The modelling of SEAS is closely connected to the modelling from the SOSA ontologies but extends it further into the

technical domain. The Procedure Execution Ontology (PEP)[10] within the SEAS framework provides generic concepts to describe a procedure with inputs and outputs, which can be seen as an abstraction of a control actor. Simple means, to describe the assignment of a command to an actuator, are provided but no explicit description of control logic itself. The SEAS System ontology provides the concept `CommunicationConnectionPoint`, which can be used to model datapoints in an automation system. The specific semantics and hierarchical relationships among control actors cannot be specified.
The *oneM2M* consortium is an international initiative to support and enable interoperability in the IoT. A number of standards are published by the initiative in this regard. The oneM2M base ontology [TS 18] is a core ontology to describe IoT devices. It needs to be extended to respective domains to describe aspects of physical systems. Within the ontology concepts are defined to describe interface entities of IoT devices. With respect to the automatic control domain the logical topology and hierarchy of IoT entities cannot be defined. Moreover, the ontology designers stipulate to use textual descriptions to specify the actual conversion between variables, i.e. control logic in a control context. Hence, the explicit formal description of control logic is not possible.

2.4.6 Other Formats and Models

Apart from the application area of automation systems other approaches exist, which are related to the modelling of control logic.
The UML [Obj15a] offers a rich set of modelling options for the description of various systems and artefacts. It is not restricted to a specific domains, hence, own classes for these systems need to be defined. Its abstraction mechanism to specify behaviour, such as UML state machines and activity diagrams, may be used for the specification of control logic. In particular, UML state machines are a possibility to specify discrete control logic. The format *XML Metadata Interchange (XMI)* [Obj15b] is defined to enable the exchange of UML models among software tools. Interfaces of classes can be defined without detailed semantics. Control actors can only be specifed when defined, e.g., as a new class.
One of the first ontologies to describe UML state machines, implemented using OWL, is described by Dolog [Dol04]. UML state machines provide generic descriptive capabilities and are used in a number of applications to specify discrete behaviour of systems. This is reflected by the use case presented in Dolog [Dol04], which is related to model web services using the described ontology. The ontology is only capable of describing UML state machines without relating them to other control actors or physical systems.
The Ptolemy II tool [EJL+03] is a widely accepted, freely available and open-source tool for the hybrid simulation of CPS [Lee08]. Amongst others it allows to specify finite state machines

[10] `https://w3id.org/pep/`, Last accessed: 22 October 2018

as well as continuous control logic. To exchange and store models developed using the tool the *Modeling Markup Language (MoML)* [LN00] is utilised. The basic building blocks of simulation models in Ptolemy II are termed actors, which can be a model of a physical process using first principle equations or a discrete control algorithm. The tool supports the export of these actors with interfaces and additionally the actual control logic can be exported to the format. Despite its ability to specify the behaviour of CPS for modelling and simulation through modelling equations, the tool and format do not describe the specification of the aspects of the physical system or devices hosting the actual control logic. In Leung *et al.* [LML+09] an ontology-based approach is described implemented in the Ptolemy II tool, which allows to check the compatibility of interface variables of actors.
The models reviewed in this section form potential candidates for the formal explicit specification of control logic in automation systems. However, all of them focus on one type of control logic and do not consider how to link to adjacent domains, e.g. from mechanical and electrical engineering. The underlying technology mostly is XML with associated limitations with regard to formal methods. The results of the analysis are summarised in Table 2.12.

Table 2.12: Results from the analysis of other formats and ontologies related to the formal modelling of control logic. XML - [BPSM+06], OWL - [W3C12].

Criteria	XMI [Obj15b]	- [Dol04]	MoML [LN00]
Physical system	○	-	-
Physical devices	○	-	-
Interface entities	+	-	+
Interface semantics	+	-	+
Logical topology	+	-	+
Logical hierarchy	○	-	+
Control logic	+	○	+
Data format	XML	OWL	XML

2.5 Summary

Within this chapter the foundations of control logic and the basic building blocks of BAS and IAS are briefly summarised in Section 2.1. This is particularly of importance as knowledge of this domain of interest is supposed to be represented through formal modelling later in this thesis as defined in Objective 2. Consequently the foundational terms and concepts of KR are introduced in Section 2.2. This is important as the formal representation of domain knowledge on control logic is required for developing novel automated knowledge-based methods as demanded by Objective 1. The current state of the art of existing knowledge-based methods,

formats and models related to various aspects of control logic in automation systems are analysed in Section 2.3 and Section 2.4. The findings drawn from this analysis are summarised below and an overview is visualised in Figure 2.9.

Finding 2.1 *Ontology is appropriate to represent knowledge from the automation systems domain.*

The use of ontology has evolved into the de facto standard for KR (see also [ESS+17, BS16b]). Other means for KR exist, e.g. frame like formalisms (see Section 2.2.3), but have not been pursued in current research and standardisation efforts. In particular, ontology provides benefits over other vocabulary types as it supports inheritance to describe specialisation of concepts, binary relationships between entities and the specification of attributes and logical statements to encode knowledge [Dib13]. An interesting rather recent development is that ontologies are standardised by international standardisation bodies, e.g. the initial version of SAREF [ETS15] in 2015 with a revised version in 2017 [ETS17a] or oneM2M [TS 18].

Finding 2.2 *OWL is an appropriate implementation language to formalise domain knowledge from the automation systems domain.*

The numerous successful examples reported in literature (see Section 2.4.2 and Section 2.4.4), which use OWL [W3C12] as a knowledge modelling language provide a strong evidence for its appropriateness to formalise domain knowledge in engineering applications. In comparison to other technologies, such as relational databases or structured data formats, e.g. XML, its strength lies in its formal *syntax* [MPSP12] and *semantics* [MPSCG12]. The formal semantics allow to automatically draw conclusions from a knowledge base through inference by an algorithm, i.e. a reasoner. In addition, compared to knowledge representation languages such as Resource Description Framework Schema (RDFS) additional more expressive constructs can be defined in OWL such as DL-rules. Its sound theoretical foundations in DL (see Section A.1 in the appendix) offer the possibility for semantic search and reasoning. Moreover, the various existing ontologies implemented in OWL allow to easily link different domain models with each other and reuse existing, formal domain knowledge.

Finding 2.3 *SWT offer the possibility to integrate heterogeneous data formats and tools utilised for the engineering of complex systems.*

SWT provide a set of technologies to address a number of challenges in the engineering domain [BS16b, ESS+17]. In particular, their ability to integrate heterogeneous formats is interesting with respect to current engineering environments, which use various disparate formats and tools. By specifying the semantics of a respective format the integration on the ontology level allows linking and cross-domain reasoning, e.g. [COC+13, Mos16]. An additional benefit is

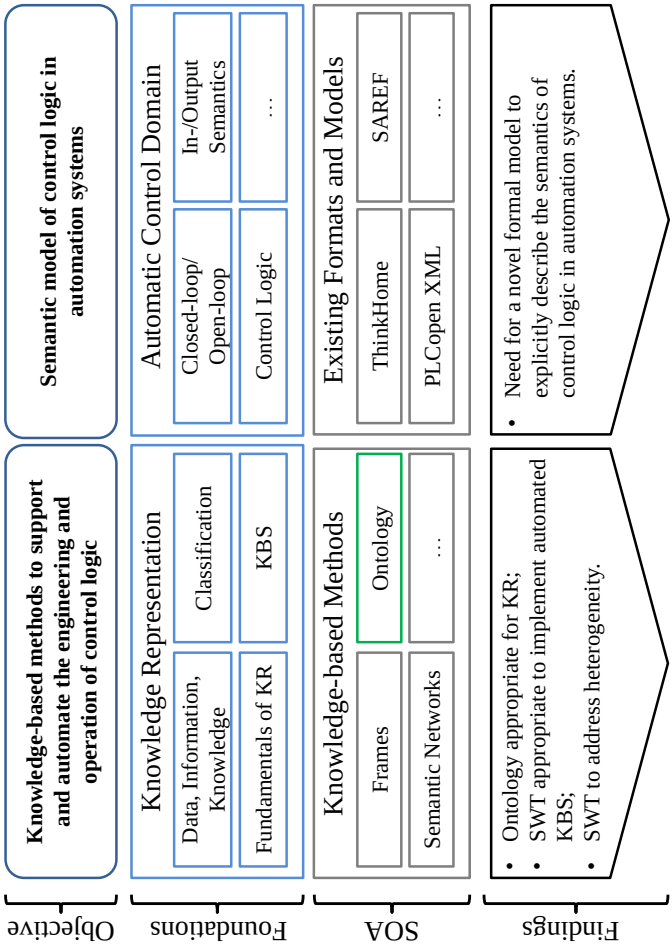


Figure 2.9: Overview on the findings drawn from the State Of the Art (SOA) analysis. KR - Knowledge Representation, KBS - Knowledge-Based System, SWT - Semantic Web Technologies, ThinkHome - [RKIK11], SAREF - [DdHR15], PLCopen XML - [PLC09]. Layout adapted from Wicaksono [Wic16].

that SWT are based on open standards and the underlying technology allows the exchange of domain knowledge on a web-scale.

Finding 2.4 *SWT support the development of automated KBS.*

SWT provide a number of characteristics and features, which provide a sound basis for their use for the development of automated KBS. SWT are open and standardised through the W3C. Several free commercial tools exist to host data expressed in the Resource Description Framework (RDF) format. Next, RDFS reasoning for light weight ontologies is possible with a good computational performance. More expressive knowledge modelling languages are available like OWL or SWRL and again OWL-DL reasoners are available, which allow to automate different task based on this knowledge, e.g. for abstract classification. The SPARQL query language allows to retrieve data from a triple store hosting the RDF data and allows in addition to specify complex and abstract graph patterns, e.g. to perform calculations. A number of successful applications of SWT-based KBS are summarised in Section 2.2.4. The foundations of SWT are summarised in Chapter A in the appendix of this thesis.

Finding 2.5 *SPARQL allows to query knowledge-bases and assert new knowledge.*

The SPARQL query language [PS17] is a versatile tool in the design and use of formal domain models and the design of KBS. Apart from retrieval of knowledge, also, knowledge can be specified in SPARQL notation and can be inserted through `SPARQL INSERT` ([PS17], see Section A.4) into the knowledge-base. It should be noted that knowledge encoded using SPARQL queries can be inserted in the knowledge base to be considered in OWL-DL reasoning, e.g. though the SPARQL Inferencing Notation (SPIN) [Knu11].

Finding 2.6 *Ontology reuse should be stipulated whenever possible.*

The proliferating development of novel ontologies has led to numerous ontologies, which significantly overlap. In his definition of ontology Studer *et al.* [SBF98] emphasise the *shared* term. Hence, ontologies need to be shared among experts and stakeholders need to agree on this. The reuse of existing ontologies is an inherent part of ontology engineering methods (see Section A.6) and should be stipulated thereof. An example for this is SAREF, which is an ETSI standard now and is based on extensively reviewing existing ontologies in the domain of smart appliances [ETS15].

Finding 2.7 *Lack of a formal explicit model to represent knowledge of the automatic control and control logic domains in automation systems and link this to knowledge on the physical system under control.*

The analysis in Section 2.4 shows that current modelling approaches lack the ability to explicitly and formally describe knowledge on control logic in automation systems and link these descriptions to adjacent domains of interest. The importance of this knowledge is recognised, e.g. for the BAS domain [DCVK16] and the related knowledge for BAS is summarised [Che15, CTM16]. Similar approaches exist in the industrial automation domain (see Section 2.4.4). Some approaches reported in literature allow to exchange information on control logic through data formats [Sch07, CTM16, PLC09, LEHM11] but lack a formal definition by means of a formal knowledge representation language. Isolated formal models exist to explicitly describe some kinds of control logic, such as UML state machines [Dol04, KP15, HWD+13] but the integration with adjacent domains and the integration with automation systems is not specified. The use of formal domain modelling to support control logic engineering in automation systems as described by Hästbacka & Kuikka [HK13] points in the direction of formally specifying the automatic control domain and its control logic. However, again the explicit modelling of control logic is not further detailed. Instead the approach relies on representing the actual control logic rather by describing function blocks representing 'parametrizable functionality provided by devices or subsystems' [HK13].
In essence, the *gap* identified is the absence of a model to explicitly and formally describe knowledge on the automatic control and control logic domains in automation systems and integrate this formal descriptions with adjacent domains such as electrical and mechanical engineering. In response to these shortcomings and with respect to the defined research objectives, Chapter 3 presents requirements in this regard and Chapter 4 a formal domain model.

3 Requirements

To accomplish the research objectives as defined in Section 1.3 and to overcome the limitations of existing approaches identified in Section 2.5, this Chapter provides requirements for the development of automated, knowledge-based methods and associated semantic models to support the engineering and operation of control logic in automation systems. An overview on the defined requirements is given in Figure 3.1 and the respective requirements are detailed in the subsequent sections. The requirements are grouped as general requirements, requirements related to the objective of developing knowledge-based methods for the support of the engineering and operation of control logic as well as to the objective for formally specifying the domain of automatic control and control logic.

3.1 General requirements

In Section 1.2 high level problems and challenges apparent in the engineering and operation of control logic in automation systems are defined. In response to these problems general requirements are defined in this section.

Requirement 3.1.1 *The intended solution needs to support the semantic integration of heterogeneous data formats and tools.*

This general requirement is motivated by the strong need to integrate various formats and tools [Dib13, BGL17, SH17] used during the course of engineering and operation of control logic in automation systems. These numerous formats can be considered as one of the most impeding obstacles in terms of the exchange for knowledge on control logic and in general automation systems [DCVK16]. Hence, the intended solution needs to enable the semantic integration of heterogeneous data formats and tools.

Requirement 3.1.2 *The intended solution should be developed independent of specific manufacturers.*

Manufacturers of automation systems often create their own ecosystems for the programming, configuration and execution of control. This partly causes the heterogeneity observed in the domain [DCVK16]. Often the vendors have their own programming environments, which are

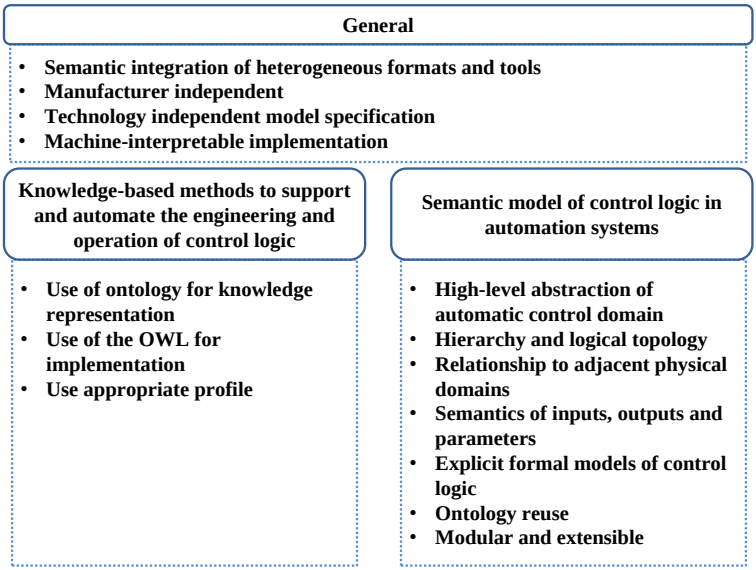


Figure 3.1: Overview of the defined general requirements, requirements related to the development of automated, knowledge-based methods and the development of semantic models to explicitly model domain knowledge on control logic. OWL - [W3C12].

one reason why it is cumbersome for the end user to switch between systems or hardware [Dib13]. However, for future automation systems it will be necessary to have open, standardised formats and interfaces to enable information exchange among stakeholders including machines [KDKO14]. In essence, it is required to develop and share conceptualisations of a domain openly and independently from manufacturers as defined in Requirement 3.1.2.

Requirement 3.1.3 *The intended models should be specified using an implementation independent modelling mechanism.*

The intended model should have a respective level of generality to be specified independently of the actual technology used for the implementation. As technology advances novel implementation tools and methods are available and novel programming and knowledge representation languages emerge. Moreover, the life-time of automation systems such as BAS easily can exceed 30 years. In conclusion, the intended models should be specified using a modelling

methodology, which is independent of its implementation to allow future reimplementation if necessary.

Requirement 3.1.4 *The intended models should be implemented using a machine-interpretable modelling language.*

It is important to use a machine-interpretable modelling language for the implementation. The strong benefit by using a machine-interpretable language over a machine-readable language is that, in the end, machines are able to understand the meaning of the specified artefacts in terms of KR (see Section 2.2). Hence, the automation of knowledge-intensive tasks associated to the engineering and operation of control logic in automation systems is enabled. Thus Requirement 3.1.4 is defined. This reflects Objective 1, which intends to leverage on the defined and formalised semantic model for knowledge-based support.

3.2 Requirements for Knowledge Representation and Knowledge-based Methods

The requirements defined in this section arise from Objective 1 defined in this thesis.

Requirement 3.2.1 *Ontology should be used for knowledge representation.*

As outlined in Section 2.2.3 various approaches and formalisms exist for the representation of knowledge in a knowledge-base. One of the findings summarised in Section 2.5 is that ontology-based KR provides some benefits over other approaches. In particular, it provides a formal syntax and semantics as well as a number of ontologies already exist, which can be reused.

Requirement 3.2.2 *The OWL should be used for implementation.*

Various knowledge representation languages exist (see Section 2.2 and Chapter A). For implementation OWL should be used as it provides a formal syntax [MPSP12] and semantics [MPSCG12] supporting semantic search and reasoning. Moreover, the numerous publications reviewed in Section 2.4 provide evidence that OWL and the associated SWT are suitable for the integration of heterogeneous data and formats as well as the development of automated KBSs. A benefit of using OWL is that the developed models can be easily linked to existing models of adjacent domains to stipulate reuse.

Requirement 3.2.3 *Use of an appropriate profile.*

A basic component of a KBS is a reasoner, which allows to infer new knowledge from the explicitly defined one (Section 2.2.4 and Chapter A). Within OWL different off-the-shelf implementations of reasoner exist. A trade-off exists between the complexity of the possible axioms and expression used in an ontology and the computational effort for a reasoner to process this knowledge. Profiles (see Section A.3.4) allow to restrict the modelling to certain axioms. This allows to tailor highly specified reasoners, which leverage on the restricted set of logical axioms and provide computationally efficient implementations. For the intended knowledge-based method an appropriate profile should be chosen.

3.3 Requirements for the Domain Model

To formally specify knowledge of the automatic control and control logic domain a number of requirements are defined in this section.

Requirement 3.3.1 *The model needs to enable the high-level description of the automatic control domain including the specification of concepts and relationships to describe control actors, inputs, outputs, parameters and control logic.*

The first requirement 3.3.1 relates to the Objective 2, which focusses on the specification of domain knowledge of the automatic control domain. An overview on the domain with respect to BAS and IAS is given in Section 2.1 and fundamental concepts are defined.

Requirement 3.3.2 *The model should allow to specify domain-specific relationships, such as hierarchy among control actors and logical topology.*

The requirement 3.3.2 addresses specifically the hierarchical relationships between control actors in automation systems. Structuring automation systems by a hierarchy is an established means of abstraction on a vertical level (see also Section 2.1). Horizontally the logical topology structures and connects control actors, which are at the same hierarchical level.

Requirement 3.3.3 *The model should allow to specify the relationship of control actors to adjacent domains of interest, such as the physical systems under control and execution platforms.*

This requirement reflects the fact that control logic is tightly integrated with the adjacent domains of interest, such as physical system under control and computation platform (see [SKK+12] and Section 2.1). For example, in BAS an input of a control logic can be related to a certain type of equipment or a controller is related to a space (Room controller). Requirement 3.3.3 reflects this fact.

Requirement 3.3.4 *The model should allow the specification of the semantics of inputs, outputs and parameters with respect to unit, medium, quantity and basic data type.*

When a logical connection is established often mismatches occur between connected inputs and outputs [DPK10, Dib13]. To enable knowledge-based analysis of these problems the semantics of inputs and outputs as well as parameters need to be defined (Requirement 3.3.4). This is also related to Objective 2.

Requirement 3.3.5 *The model should explicitly formally specify control logic.*

As highlighted in the summary of the analysis of the current state of the art (Section 2.5), no explicit formal specification of control logic in automation systems and their relationship to adjacent domains exists. However, the absence prevents the use and analysis of this information and related knowledge. Hence, Requirement 3.3.5 demands the explicit formal specification of control logic. In particular, this relates to the core of Objective 2.

Requirement 3.3.6 *Existing ontologies should be reused whenever possible.*

A fundamental step in knowledge engineering as described in Section A.6 is the reuse of existing ontologies. When demanding the exchange of domain knowledge via the web as required to enable the envisioned novel architectures in automation technology [KDKO14, VH14b] open, shared ontologies need to be available. As reviewed in Section 2.4 a number of ontologies do exist and should be considered for reuse if applicable.

Requirement 3.3.7 *The intended models should be designed in a modular extendible fashion to stipulate and enable its future reuse and adoption.*

Methods to support the engineering and operation of control logic in automation systems are a vibrant research area and novel methods are continuously developed. Hence, Requirement 3.3.7 ensures that the extension of the model is possible with limited effort. Additionally, as it can be seen from the review in Section 2.4.4 and 2.4.2, a number of models already exist for adjacent domains. Hence, the novel models should be designed to enable easy integration with these existing models.

3.4 Summary

From the initially defined objectives in Section 1.3 and to overcome the found limitations of existing work as summarised in Section 2.5 requirements are defined, which need to be fulfilled by possible methods and models. The requirements are related to general needs for a solution arising from the problems related to the engineering and operation of control logic and the development of knowledge-based methods to support and automate the engineering and operation of control logic in automation systems. As a prerequisite for the development of automated, knowledge-based methods to support and automate the engineering and operation of control logic in automation systems a novel semantic model is presented in the next chapter, which is designed with respect to the defined requirements.

4 Semantic Modelling of Control Logic in Automation Systems

Within this chapter a novel, semantic model is defined, which allows the explicit, formal description of knowledge of the automatic control and control logic domains in automation systems. In addition, the model is designed to allow the linking of this knowledge to adjacent domains relevant in the engineering and operation of automation systems, such as mechanical and electrical domains. The novel model particularly aims at filling the gap identified in the analysis of existing work as presented in Section 2.5. In addition, this contribution targets Objective 2.

This chapter is organised as follows. First, the modelling methodology utilised for the development of the semantic model is described in Section 4.1. In Section 4.2 the established layered model architecture is presented, which allows to extend the sub-model of the automatic control domain with the sub-models of the control logic domain. Next, a semantic model for the automatic control domain is presented in Section 4.3. Finally, semantic models for the explicit modelling of different types of control logic are presented in Section 4.4. The developed models allow the semantic modelling of generic control logic types and building blocks, such as algebraic expressions in Section 4.4.1, schedules in Section 4.4.2, sequence control in Section 4.4.3 and non-linear, discrete two-point control in Section 4.4.4. Moreover, semantic models for the explicit description of discrete formalisms of control logic are presented including UML state machines [Obj15a] in Section 4.4.6 and state graphs from VDI 3814-6 [VDI08] in Section 4.4.7. The presented models and ontologies represent a set of possible types of control logic typically available in automation systems of different vendors (see Req. 3.1.2), but are not meant to be exhaustive.

4.1 Modelling Methodology

Within this section the established modelling methodology for developing the semantic models and their formalisation is described. For the specification an Object-Oriented Modelling (OOM) methodology is utilised as described in Section 4.1.1. To formalise the developed semantic models in terms of knowledge representation, OWL is used and the respective mapping between both is described in Section 4.1.2.

4.1.1 Object-Oriented Modelling Methodology

The OOM [DR15] is a well-accepted modelling methodology, which is applicable in multiple disciplines. It has a proven track of records of being successfully applied to diverse application areas from engineering, computer science, software engineering, systems engineering and many more [DR15, RQd12]. The method is chosen for the modelling of the semantic model presented in this chapter as it provides appropriate modelling mechanisms for the respective tasks. The utilised mechanisms are summarised in the following paragraphs.
Central to OOM is the abstraction of an entity into a *class*. A class can be seen as a template of a certain type of entities, with attributes and methods for these entities defined once in a class. In OOM it is possible to specialise classes from other classes. This mechanism is termed *inheritance*. A class specialised from a super-class inherits all attributes and methods of this class. Attributes define the relationship of a class to a data type. The attribute types used in the modelling are listed in Table 4.1. *Instances* of a class are entities, which can interact with each other and differ for example by the actual values specified for the attributes of each instance.

Table 4.1: Attribute types and their description used for the definition of the semantic models.

Attribute	Description	Reference
`string`	Set of finite-length sequences of characters	[BM04]
`non-negative integer`	Any positive integer number including 0	[BM04]
`double`	Institute of Electrical and Electronics Engineers (IEEE) double-precision 64-bit floating point type	[BM04]
`literal`	Literal in RDF	[CWL14]

The OOM paradigm is supported by a number of modelling and programming languages. The UML [Obj15a] is an international standardised modelling language, which fully supports OOM and provides tools to support the modelling as well as diagrams for the visualisation of the model. An introduction to the UML is given in Rupp *et al.* [RQd12]. Here the UML concepts and class diagrams are used for the technology independent specification (Req. 3.1.3) of the semantic model. Hence, users can implement the model in the target language of their choice. Attributes are defined in a class, if a relationship of this class to a data type is expressed. The attribute types used in the modelling are listed in Table 4.1.

To model binary relationships among classes named directed associations are used. A *stereotype* is used to indicate special classes. The stereotype *MetaClass* is used to distinguish classes, which cannot be instantiated. However, in OOM it is possible to instantiate classes, which inherit from a *MetaClass*.

4.1.2 Ontology-based Formalisation

For the formalisation of domain knowledge in terms of KR (see Section 2.2.3) different technologies exist. As a result of the analysis of the state of the art[1] and, in particular, as it supports inheritance, binary relationships, attributes and formal logics [Dib13], all needed to implement the models developed following OOM, ontology is used for KR in this thesis.
In terms of acquiring knowledge structured ontology engineering methods are available, e.g. Ontology 101 [NM01] or METHONTOLOGY [FLGPJ97] (see Section A.6). In the definition process of the semantic model the structured approach presented by Noy & McGuinness [NM01] is followed whenever possible.
As described in Section 2.5 a conclusion drawn from the analysis of the state of the art is that the OWL knowledge modelling language should be used for the implementation of the developed models (see Req. 3.2.2). OWL offers several advantages in its application: OWL is an internationally standardised knowledge modelling language [W3C12]. It provides a well-defined syntax [MPSP12] and formal semantics [MPSCG12]. In particular, the formal semantics enable the use of reasoning on the specified knowledge and allow machines to interpret this knowledge (see Req. 3.1.4). Numerous examples can be found in the literature, where ontologies implemented in OWL are used for the development of KBSs, e.g. for automated interoperability verification [Dib13], verification of the compliance to standards [PVDV+11] or verification of plant engineering data [ALGM13] (see Section 2.3). Amongst others, these provide evidence that OWL is an appropriate knowledge modelling language. Moreover, the underlying SWT provide means for semantic integration of heterogeneous data formats and associated tools [Sab16, COC+13] (see Req 3.1.1). The foundations of OWL are summarised in Section A.3 in the appendix of this thesis.
In literature efforts are reported towards defining a mapping between OWL ontologies and the UML [FGP+14, BBČ+10]. However, there is no standardised mapping available and each effort has made its own assumptions. In particular, none of the approaches does support all features of OWL or UML. Hence, an own mapping is established in this work to describe the respective model unambiguously. The established mapping from the UML specification of the semantic models to their formalisation in OWL is summarised in Table 4.2. The chosen features of OWL aim at staying within the OWL RL profile [MCGH+12] to avoid OWL-DL reasoning (see Section A.3.4). Mainly this is motivated as OWL-DL reasoning is within

[1] See Finding 2.1

Table 4.2: The mapping of UML modelling concepts [Obj15a] to OWL [MPSP12] as established in this thesis.

UML	**OWL**
Classes and properties	
Class	`owl:Class`
MetaClass	`owl:Class`, `owl:unionOf`, `owl:disjointWith`
Inheritance	`rdfs:subClassOf`
Attribute	`owl:DataTypeProperty`
Named directed association	`owl:ObjectProperty`
Source of directed association	`rdfs:domain`
Target of directed association	`rdfs:range`
Cardinalities at target	
0..1	`owl:minQualifiedCardinality 0`, `owl:maxQualifiedCardinality 1`
0..∗	`owl:minQualifiedCardinality 0`
1	`owl:qualifiedCardinality 1`
1..∗	`owl:allValuesFrom`, `owl:someValuesFrom`
∗	`rdfs:range`

complexity class NP-hard [MCGH+12], whereas OWL RL profile offers PTIME-completeness [MCGH+12] with associated benefits in the computational effort needed for the execution of reasoning algorithms. The expressibility of this profile is found to be sufficient in the course of the implementation and validation of the models.
The defined classes are formalised in a straightforward manner as an `owl:Class`. OWL does not provide a built-in language construct to express classes, which cannot be instantiated. Hence, a UML Meta-Class is modelled as `owl:Class` and all subclasses are defined as disjoint and the respective meta class is defined as a union of all subclasses. Inheritance relationships between UML classes are expressed using the `rdfs:subClassOf` relationship in OWL. Binary relationships between classes described as named directed associations are modelled as object properties. When applicable the cardinality of this relationship is defined and modelled as cardinality restriction on the respective property (see Table 4.2). Attributes specified in a class are formalised as data type properties in OWL and the respective range is specified depending on the attribute type. The graphical nomenclature for the illustration of OWL ontologies is presented in Figure 4.1 and each presented ontology can be obtained from

Figure 4.1: Established nomenclature for the graphical representation of OWL [W3C12] implementations in this thesis [SPS17].

its web repository by following the specified Unique Resource Identifier (URI) and is provided open-source under a creative commons license.
Formal constructs, such as restrictions, are not depicted in illustrations of the semantic models but mentioned in the text. All prefixes used in this thesis are summarised in Chapter D. The statistics of the defined ontologies are given in Table 4.3, Section 4.5.

4.2 Layered Model Architecture

For the semantic modelling of the automatic control and control logic domains a layered model architecture as depicted in Figure 4.2 is established. The architecture consists of two layers: Layer 0 to define on an abstract level domain specific knowledge of the automatic control domain and Layer 1 to define specific knowledge of the control logic domain. Note, the models of the control logic domain extend (Req. 3.3.7) the automatic control domain model to allow interoperation among different types of control logic. In addition, the layered architecture allows to link the automatic control domain to adjacent domains (Req. 3.3.3) and, hence, a link can be established across the domains of control logic, automatic control and adjacent

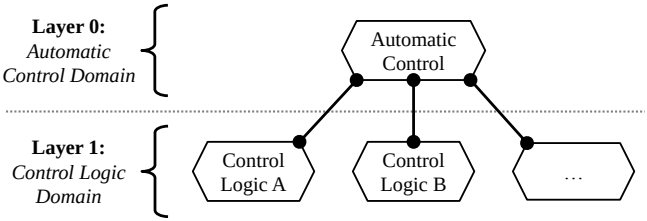


Figure 4.2: Layered architecture, where the automatic control domain (Layer 0) can be extended with domain descriptions of the control logic domain (Layer 1).

domains such as mechanical or electrical domains. An example for the use of this is presented in Section 5.2.
In Section 4.3 a semantic model is introduced for the automatic control domain and, subsequently, in Section 4.4 semantic models for the explicit modelling of a non-exhaustive set of control logic types are presented.

4.3 Semantic Model of the Automatic Control Domain

Within this section a semantic model of the automatic control domain is defined, which constitutes Layer 0 in the layered model architecture (see Figure 4.2). The model is depicted in Figure 4.3. The model can be linked to adjacent domains of interest and, hence, bridges the gap between explicit models of control logic introduced later in this chapter and adjacent domains, such as sensors, actuators, technical equipment, plants or spaces. As these links to adjacent domains are use case specific an example for the successful linking is presented in Section 5.2. The core part of the model is the description of the *Sense-Process-Actuate* pattern, a pattern recognised in the automatic control domain [SSKD11]. In automation systems this pattern typically requires some `ControlActor`[2] to sense the current state of the controlled system by accessing information via its `Inputs`. This information then is processed by some control logic[3], here modelled as `ApplicationLogic`. Finally, after processing, outputs are generated, which might be consumed by other `ControlActors` or be assigned to actuators installed in the system under control. This directional notion of an input to the processing unit by the control actor to an output is an important characteristic of automatic control in general and can be used, e.g. to improve the performance of simulation algorithms [Mod17a].

[2] See Definition 2.4.
[3] See Definition 2.6.

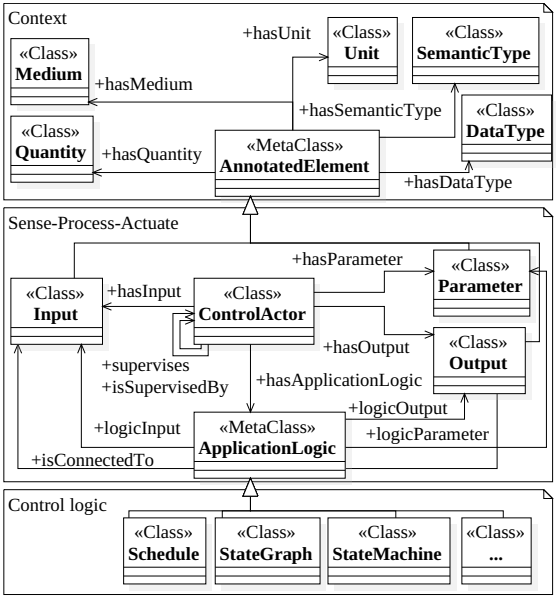


Figure 4.3: Class diagram of the semantic model of the automatic control domain.

`Parameters` used within the `ApplicationLogic` are time-invariant values influencing the behaviour of a `ControlActor`. The description of the sense-process-actuate pattern and the specification of fundamental concepts and relationships of a control actor are in response to Req. 3.3.1.

The relationship of a `ControlActor` to its inputs, outputs, parameters and application logic can be expressed via the `hasInput`, `hasOutput`, `hasParameter` and `hasApplicationLogic` relationships, respectively. To express a relationship between components of an application logic to inputs, outputs and parameters `logicInput`, `logicOutput` and `logicParameter` can be utilised, respectively. The use of the relationships to link explicit models of control logic to the interfaces of control actors is further detailed in the subsequent sections, where explicit models of different types of control logic are introduced.

Finally, the hierarchy between different `ControlActors` can be expressed using the `supervises` and `isSupervisedBy` relationships to distinguish between supervisory[4] and low-level[5] control logic. Note, `supervises` is defined as the inverse of `isSupervisedOf`.

[4] See Definition 2.9.
[5] See Definition 2.8.

Hierarchical relationships between different control actors can be observed in many cases of automation systems, e.g. supervisory control algorithms in building management systems or production scheduling systems in manufacturing. This is, in particular, in response to Req. 3.3.2. The connection (logical topology) of outputs to inputs can be specified using the `isConnectedTo` relationship.

The inputs, outputs and parameters of control actors can be further annotated, if applicable, by their `Medium` using the `hasMedium`, `Quantity` using the `hasQuantity` and `Unit` using the `hasUnit` relationships. To specify the basic data type of an `AnnotatedElement` via `hasDataType`, a relationship to `DataType` can be established. Example types are integer, boolean or floating point number and the in Section B.1 introduced Basic Datatype Ontology (BDO) can be utilised for this purpose. To further distinguish an annotated element it might be required to specify a `SemanticType` via the `hasSemanticType` relationship. For example, a control actor can have an input, which is a double value of the indoor air temperature measured in Kelvin. Here, for the given example the Medium is *Air*, the quantity is *Temperature*, the unit is *Kelvin*, the basic data type is a *double*, and its semantic type can be specified as *indoor* to distinguish it from, for example, the outdoor air temperature. The `SemanticType`, however, is domain specific. Specifying the semantics of the interfacing elements is in response to Req. 3.3.4.

The model is designed such that it can be easily extended with explicit specifications of control logic, e.g. with the ones presented in Section 4.4. This allows to easily exchange models of control logic and maintain and extend the proposed explicit specifications by custom ones. This is in response to Req. 3.3.7. In Figure 4.3 as an example the `Schedule`, `StateGraph` and `StateMachine` models are depicted.

The presented model is implemented using OWL according to the defined mapping (see Section 4.1.2). Classes and object properties are asserted as illustrated in Figure 4.4.

Beyond the described mapping methodology some additional restrictions are specified and added to the ontology as described in the following paragraphs.

Universal restrictions are embodied in the definition of the `ControlActor` and `AnnotatedElement` classes. Each universal restriction ensures that if an individual of one of these classes exists and it has a relationship via the restricted object property to another individual, then this individual needs to be of the specified class. Moreover, individuals, which do not have a relationship via the restricted property at all, are part of this class [HKR+04]. If these individuals should be excluded from the class a closure axiom is required as explained later in this section. The listing in Code 4.1 specified in Turtle syntax [BBLPC14] the universal restriction asserted in `CTRLont` for the `AnnotatedElement` class with respect to the `hasDataType` property. Additional restrictions are omitted for brevity of the documentation.

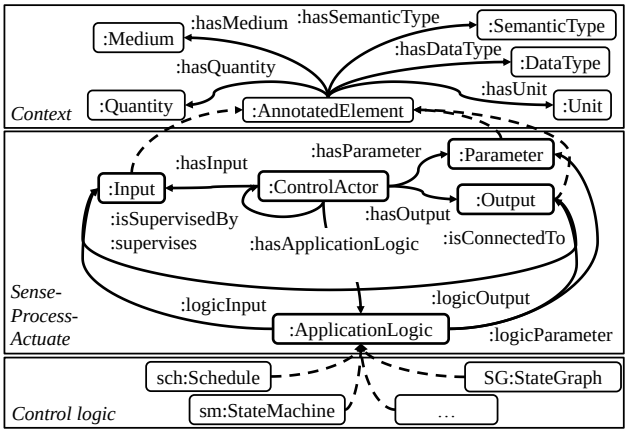


Figure 4.4: Concepts and relationships of Control Ontology (`CTRLont`) [SPS17, adapted].

Code 4.1: Universal restriction on property `hasDataType` in class `AnnotatedElement` defined in `CTRLont`. The other restrictions of the class are omitted here for brevity.

```
:AnnotatedElement rdf:type owl:Class ;
  rdfs:subClassOf [
    rdf:type owl:Restriction ;
    owl:onProperty :hasDataType ;
    owl:allValuesFrom :DataType
    ] .
```

To express that an individual A of the class `ControlActor` must have a relationship to at least one individual B via the `hasApplicationLogic` and `hasOutput` object properties, existential restrictions are formulated in the ontology. In particular, when combined with a universal restriction it is additionally specified that the individual B and alike need to be members of the `ApplicationLogic` or `Output` classes, respectively. Combining an existential and universal restriction on a property is referred to as closure axiom [HKR+04]. Code 4.2 shows the applied closure axiom for the class `ControlActor` on the `hasApplicationLogic` property.

Code 4.2: Closure axiom for the class `ControlActor` on property `hasApplicationLogic` defined in `CTRLont`.

```
:ControlActor rdf:type owl:Class ;
  rdfs:subClassOf [
    rdf:type owl:Restriction ;
    owl:onProperty :hasApplicationLogic ;
    owl:someValuesFrom :ApplicationLogic ] ,
    [
    rdf:type owl:Restriction ;
    owl:onProperty :hasApplicationLogic ;
    owl:allValuesFrom :ApplicationLogic ] .
```

For the `hasInput` and `hasParameter` object property no closure axiom is specified as it might be the case that a control actor does not have an input or a parameter.

To evaluate the proposed `CTRLont` ontology a scenario is considered, where two control actors `R1` and `R2` are modelled. Figure 4.5 presents the considered scenario as block diagram (above

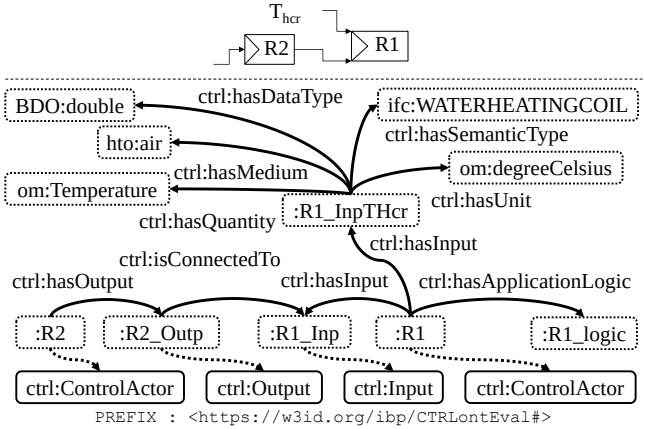


Figure 4.5: Excerpt of instances describing two control actors `R1` and `R2` (upper part), their logical topology and the annotation of T_{hcr}, a heating coil return temperature of an air handling unit (see Section 5.1).

dashed lined) and an excerpt of individuals representing the scenario (below dashed line).

The relationship of each control actor to its inputs and outputs can be modelled (`ctrl:hasInput, ctrl:hasOutput`) and the connection of an input to an output (`ctrl:isConnectedTo`). The control logic of each control actor can be further specified by detailing on the individual (`:R1_logic`), which is related to the control actor via the `ctrl:hasApplicationLogic` object property.

As an example the annotation of the input `:R1_InpTHcr` is presented. Its quantity and unit are specified by relating it using the respective object properties to individuals from the Ontology for units and Measures (OM) ontology [RWT11]. To specify the medium of the physical measurement an individual defined in HTO [CKAK15] is reused. Finally, to distinguish it further form other air temperatures measured in degree Celsius, its semantic type is specified by relating it to an individual from the ifcOWL [PT16] ontology for the buildings domain (`ifc:IFCWATERHEATINGCOIL`).
To specify the basic data type of a `ctrl:AnnotatedElement` a relationship is established to an individual of the `BasicDatatypeOntology`. This ontology is derived from the basic data types as specified in Biron and Malhotra [BM04]. This is necessary as the data types from the XML Schema Definition (XSD) specification [BM04] cannot be used in OWL object property assertions, as these belong to a set of reserved vocabulary in OWL [MPSP12, Table 3]. To enable the assertion of the `ctrl:hasDataType` object property to a respective individual the `BasicDatatypeOntology` is bootstrapped from the XSD specification [BM04] by creating a `owl:NamedIndividual` and a `owl:Class` for each data type and by keeping the defined hierarchy [BM04]. The generated concepts and individuals of the ontology are listed in Section B.1.

4.4 Semantic Models of the Control Logic Domain

Within this section models and ontologies are presented, which allow to describe explicitly and formally different types of control logic. The chosen types are typically available in automation systems of different vendors (see Req. 3.1.2), but are not meant to be exhaustive.
All semantic models are presented equally by first describing the object-oriented modelling using the UML and then detailing its ontology-based formalisation using OWL. Each model is studied in a simple use case for evaluation as stipulated by ontology-engineering methods (see Section A.6). A detailed validation of the proposed model in use-cases is elaborated in two studies documented in Chapter 5.

4.4.1 Algebraic Expressions

A basic component of control logic are mathematical expressions. For example, in continuous control the transfer function [Abe10] of a component exactly defines its behaviour and provides the input for further analysis, such as stability analysis [Abe10]. Another example are the boolean conditions defined in state-based control logic, which describe if a transition from one state to another should fire.

A model to describe mathematical expressions occurring in different types of control logic is presented in Figure 4.6. The model is based on reviewing existing approaches reported in the

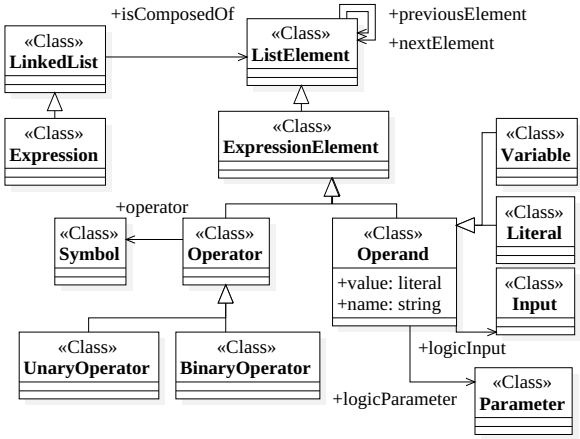


Figure 4.6: Class diagram of a model to describe mathematical expressions.

domain of simulation of chemical process systems [MMWY09] and the specification of mathematical knowledge on the web [WR12]. Precursors of the model developed simultaneously are published in Schneider *et al.* [SPS17] and Terkaj *et al.* [TSP17].

An `Expression` is modelled as a `LinkedList`, which `isComposedOf` `Expression-Elements`. As the order of expression elements in an expression matters, expression elements are subclasses of `ListElement`, where the relationships `previousElement` and `next-Element` allow to express the ordering. The order is defined to increase from left to write, e.g. in the equation $x < 1$ the first element would be the variable x.

Additionally, an expression element can be differentiated into `Operators` and `Operands`. Operators in an expression are mathematical symbols specifying what happens with the operands associated to it. Operands, in difference, are variables or values, which can be, for example, added or substracted. The `Symbol` of an operator can be specified using the relationship `operator` to instances of the symbol class. A `UnaryOperator` specifies operators, which are applied to a single `Operand` in contrast to a `BinaryOperator`, which can be only be applied between two `Operands`. The operand can be further specified either as a `Literal` or a `Variable`, where the literal connection between a parameter and the literal can be expressed using `logicParameter` and for a variable using `logicInput`. Finally,

an operand can be further specified as a `Variable` or a `Literal` to distinguish constant values and variables, which might change.
The presented model is formalised as an ontology as presented in Figure 4.7. The ontology is

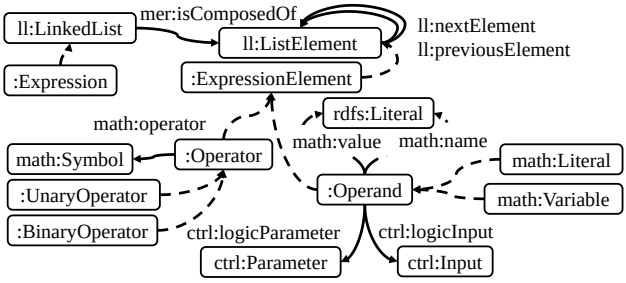


Figure 4.7: Classes and relationships as defined in the `ExpressionOntology` [SWO19].

based on a combination of reused concepts (`LinkedList`, `ListElement`, `Symbol`, `Literal`, `Variable`) and relationships (`nextElement`, `previousElement`, `operator`, `value`, `name`) as well as own, newly introduced ones (`Expression`, `ExpressionElement`, `Operator`, `UnaryOperator`, `BinaryOperator`, `Operand`). In order to describe the mentioned required ordering of elements in an equation, the linked list model available from the OntoCAPE family of ontologies [MMWY09] is reused. The respective concepts are specialised to describe the expression domain by introducing the subclasses `:Expression` and `:ExpressionElement`. The OpenMath ontology [WR12] is reused to specify mathematical knowledge on the web, such as the symbols of unary and binary operators and the meaning of operands in equations. In addition to the conversion specified in Section 4.1.2 no further formal statements are made.
For a simple evaluation the expression '$R1\_Inp <= 0.0$', originating from a condition in a transition of a state graph, is represented using the described ontology. In Figure 4.8 the respective triples are depicted.
The expression `:exp` is decomposed into three elements `:ele0`, `:ele1`, `:ele2` and the elements are related sequentially to each other via the `ll:nextElement` relationship. The decomposition starts from the first element `:ele0`, which is modelled as an `exp:Operand`. The next element of the expression `:ele1` is a binary operator. The operator is specified using the `math:operator` object property to the respective individual from OpenMath ontology [WR12]. Finally, the last element `:ele2` is modelled as a `math:Literal` and its value is specified via the `math:value` data type property. Both `:ele2` and `ele0` are related to the top level representation of the control actor via the respective `ctrl:logicParameter` and

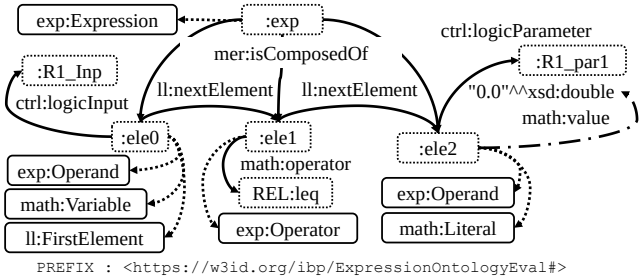


Figure 4.8: Excerpt of instances to describe the expression 'R1_Inp <= 0.0'.

`ctrl:logicInput` properties. For example, `:ele0` is related via the `ctrl:logicInput` property to the individual `R1_Input`. The `:ele0` is additionally typed as a `ll:FirstElement`.

4.4.2 Schedules

A common piece of control logic frequently occurring in BAS is a schedule. For example, a schedule limits the operation of technical equipment to times of the day, where an office building is occupied potentially reducing the operating hours and, hence, reduce significantly the associated costs and energy demand.

In Figure 4.9 a model to describe schedules is displayed. A `Schedule` is an `ApplicationLogic` and its execution is repeated with some `Periodicity`. Dependent on the current time, which is provided by an `Input` to the schedule, the current value of its `Output` is determined. To describe the mathematical relation between the time and the output a polynomial `MathematicalFunction` is modelled with a `PolynomialConstant`. The polynomial constant can be specified by the attribute `polyConstantDegree`, which is restricted to be a non-negative integer and a `polyConstantLiteralValue`, which is restricted to be a double precision floating point number. The intervals where a certain polynomial function is active can be specified using the classes `Interval`, `StartPoint` and `EndPoint` with their respective `Coordinates`.

The described semantic model is formalised using the established mapping methodology. The result is depicted in Figure 4.10. In addition, individuals to specify periodicity are defined, i.e. `Daily`, `Weekly`, `Monthly` and `Yearly`.

The defined ontology is evaluated in a simple use case as illustrated in Figure 4.11. A daily schedule restricts the operation time of some technical unit between 8am and 6pm. Three intervals (`sched_Interval1 - sched_Interval3`) can be identified, where the output has different values. From midnight until 8am its value is zero, from 8am until 6pm its value is

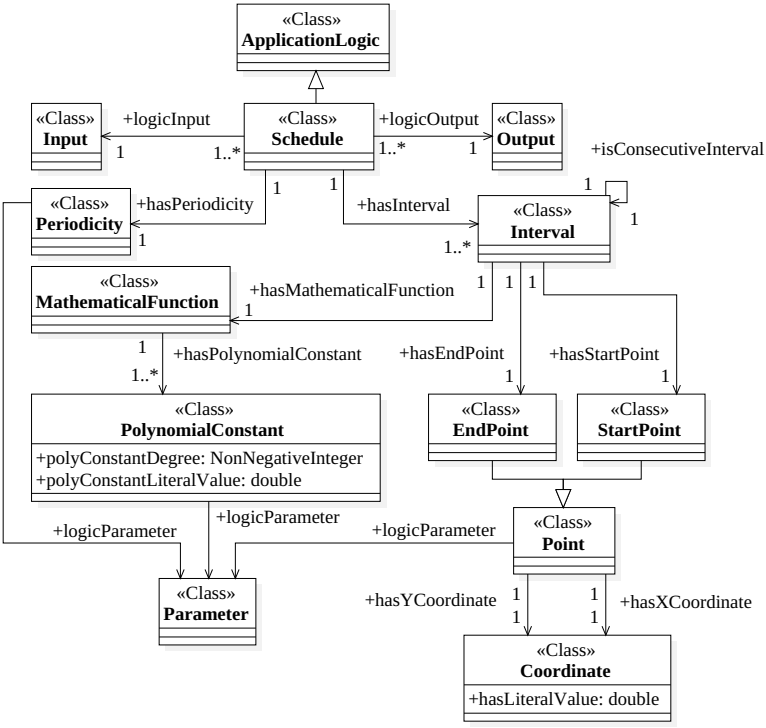


Figure 4.9: Class diagram of a model to describe schedules.

one and from 6pm until midnight its zero. The respective schedule is depicted in Figure 4.11 (below dashed line) and an excerpt of instances formalising the schedule are depicted (above dashed line). The interval of interest is the third one modelled using the start point of interval three (`_:b1`). The interval starts at 6pm and the x and y coordinates of the respective point are specified. For brevity, the mathematical function describing the behaviour of the schedule between the two points is not depicted here.

4.4.3 Sequence Control

Sequence control is a type of control logic often appearing when there are control actions within an automation system, which possibly contradict each other. For example, in an AHU

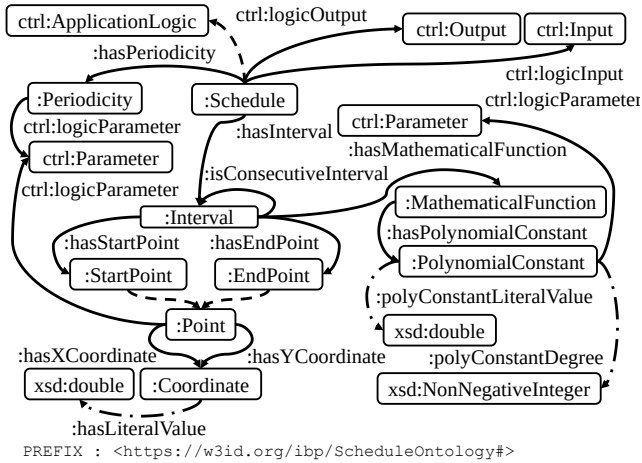


Figure 4.10: Class and relationships of the `ScheduleOntology` (adapted from [SPS17]).

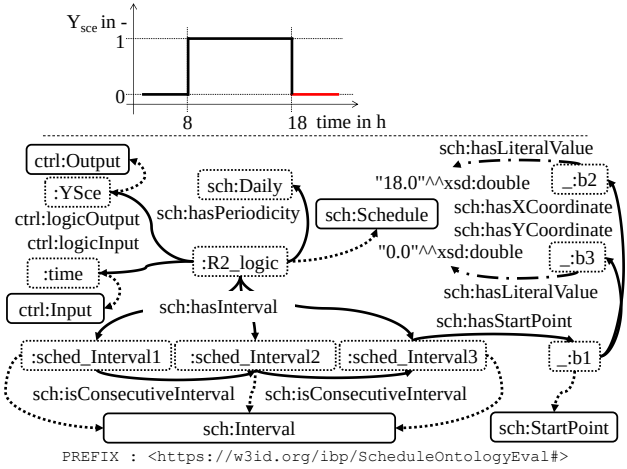


Figure 4.11: Excerpt of instances to represent a schedule.

air entering the unit can be either heated or cooled by sequentially arranged heating and cooling coils in the direction of air flow. Technically it can be the case that the respective coils are heated and cooled at the same time, thus wasting primary energy without any gain for the supplied air. In practical applications control logic, such as sequence control [ISO05], prevents this by only allowing cooling to start when heating ended, or vice versa.
In Figure 4.12 a semantic model to describe a sequence control is presented. It is similar

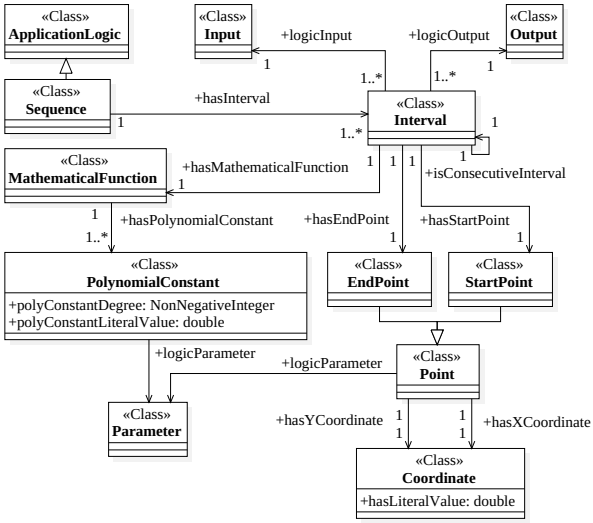


Figure 4.12: Class diagram of a model to describe a sequence control.

to the model developed for a schedule presented in Section 4.4.2, in fact a schedule can be modelled as a sequence controller, where the input is restricted to only the current time. In a sequence controller a `Sequence` has some `Intervals`, which describe for each `Input` how logically it should be converted into an `Output`. The order of the intervals can be specified by relating them using the `isConsecutiveInterval` relationship. To specify each interval the `Coordinates` of its `StartPoint` and `EndPoint` need to be defined using the respective relationships as depicted in Figure 4.12. The modelling of the actual calculation by means of a polynomial function can be specified exactly as for a schedule using the classes `MathematicalFunction` and `PolynomialConstant` using the respective binary relationships to relate them. Note, intervals need to be defined such that the mathematical function is continuous on the respective interval. The linking to adjacent domains using classes from

the CTRLont are defined in the class diagram using the logicInput, logicOutput and logicParameter relationships.
The SequenceControlOntology presented in Figure 4.13 is the result of the formalisa-

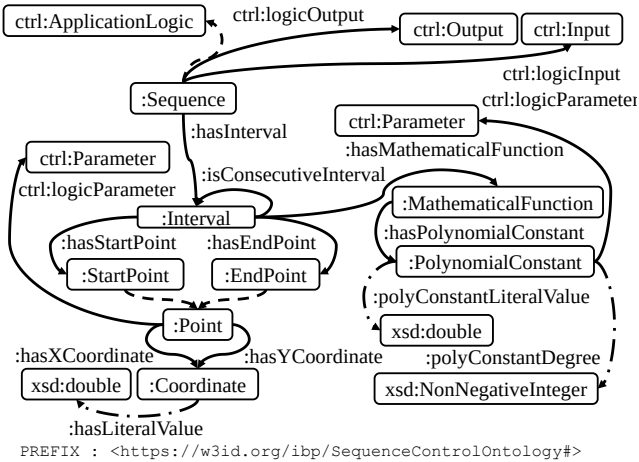


Figure 4.13: Concepts and relationships of the SequenceControlOntology.

tion of the described semantic model. Classes and relationships are defined as modelled. No additional statements are defined in the ontology.
To illustrate the use of the ontology an excerpt of instances representing the sequence control of an AHU is presented in Figure 4.14. Above the dashed line an example for a sequence control is depicted as reported in ISO 16484-3 [ISO05]. The graph should be understood as follows: Depending on the outdoor air temperature T_{oa} the normalised control signal for the heating coil valve Y_{hea} and cooling coil valve $Y_{\mathbf{coo}}$ should be changed. If the outdoor air temperature is below 10 degree Celsius Y_{hea} should be 1, Y_{coo} should be zero; if the temperature is between 10 and 16 degree Celsius Y_{hea} constantly decreases and Y_{coo} stays zero; if the temperature is between 16 and 18 degree Celsius Y_{hea} and Y_{coo} should be zero; if the temperature is between 18 and 24 degree Celsius Y_{hea} should be zero and Y_{coo} should constantly increase and, finally, if the temperature is above 24 degree Celsius Y_{hea} should be zero and Y_{coo} should be 1.
To evaluate the ontology the presented sequence control is modelled and for convenience the triples describing the interval (int2 in Figure 4.14) between 18 and 24 degree Celsius are displayed. In addition, the relationship to inputs, outputs and parameters is depicted.

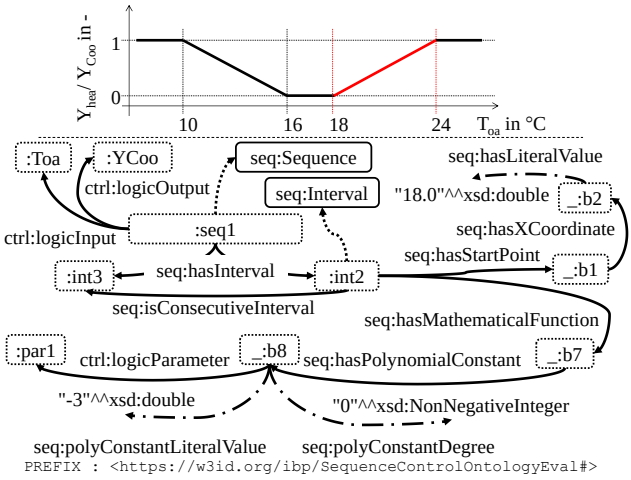


Figure 4.14: Excerpt of instances (below dashed line) of interval 18 - 24 degree Celsius (above dashed line).

4.4.4 Two-Point Discrete Control

Two point control of a process is a frequently applied type of control logic in feedback control. Examples are the heating control of a room by a heater, where the heater can be only operating or turned off. If the room temperature measured by some sensor is below its current set point the heater is turned on, if above the heater is turned of. Also, in open-loop control it is used, e.g. for a simple value conversion of a continuous value into another dependent on a threshold. In particular, the discrete variant considered here has a boolean input and a continuous output value.

Figure 4.15 presents a model to specify discrete two-point control logic. Each `TwoPoint-Discrete` is related to an `Input` and `Output` and `contains` a `LowerOutput` and an `UpperOutput`. The `LowerOutput` is the value associated to a Boolean `false` and the `UpperOutput` is the output value if the input is `True`. The model is formalised using the defined mapping methodology and the result is depicted in Figure 4.16.

The defined ontology is used to model a simple use case with two-point control as depicted in Figure 4.17. The behaviour of this type of control logic is depicted in a two-dimensional graph (left of dashed line in Figure 4.17). The input `u` is denoted on the x-axis and the output `y` on the y-axis. In this example, for instance, if the value of the input `u` is `false` the output `y` is 2; if the input `u` is `true` then the output `y` is 11. The use of this graphs aligns with the approach of transfer functions [Abe10], where a model for generic transfer function with multiple inputs

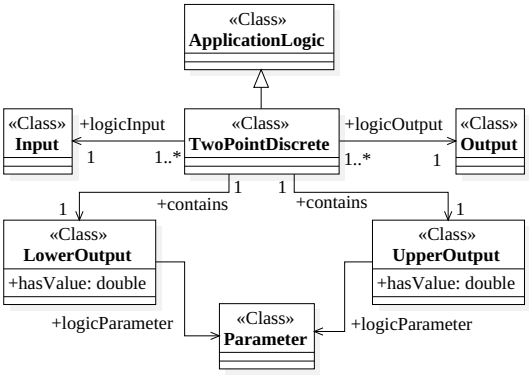


Figure 4.15: Class diagram of a model to describe discrete two point control.

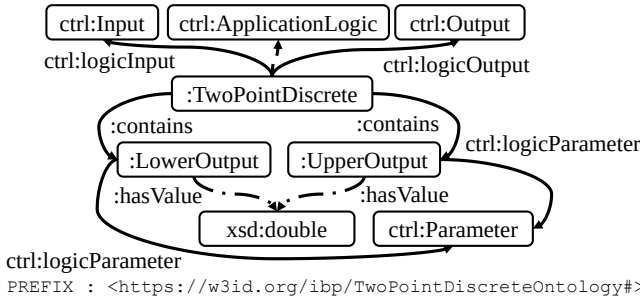
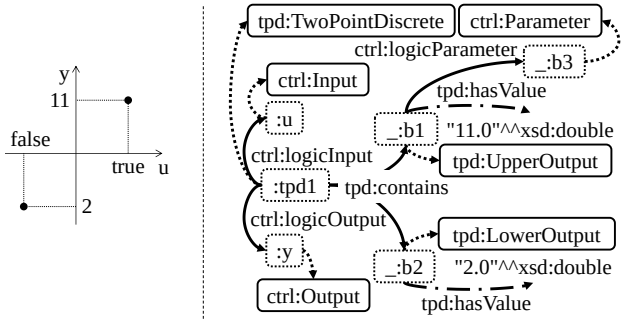


Figure 4.16: Concepts and relationships of the `TwoPointDiscreteOntology` [SWO19].

and single outputs is presented in Section 4.4.5. Another example of the use of this type of control logic is presented in Section 5.2.

4.4.5 Transfer Function Element

In linear control theory a transfer function element is a control actor with some inputs and outputs, where the relationship between these is mathematically defined by a transfer function [Abe10]. Examples for transfer function elements are proportional, integral and derivative elements, which in combination are frequently used in one of the most applied closed-loop controllers: PID-control [Abe10, ÅH01]. The semantic model and ontology described in the following allows to explicitly describe arbitrary transfer function elements through formally




```
PREFIX : <https://w3id.org/ibp/TwoPointDiscreteOntologyEval#>
```

Figure 4.17: Schema and instances of a two point discrete element.

representing their expression. This allows to describe a wide range and on a generic level control actors with control logic originating from linear automatic control domain.
In Figure 4.18 a semantic model to describe these elements is presented. The central concept

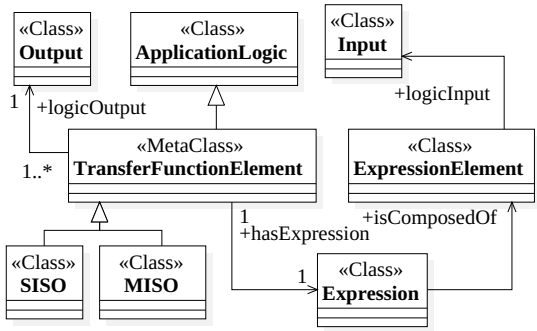


Figure 4.18: Class diagram of a semantic model to describe transfer function elements.

is a `TransferFunctionElement` to represent this kind of control logic. The expression describing mathematically how inputs are converted to outputs can be specified as an `Expression` following the model presented in Section 4.4.1 and related to by the `hasExpression` relationship. Each element has only a single expression related to it. For simplification the model restricts a transfer function element to either have a single input and a single output (`SISO`) or multiple inputs and a single output (`MISO`). The single `Output` can be related to

the element via the `logicOutput` relationship. Inputs relate to the respective element in the expression.
An ontology formalising the model is depicted in Figure 4.19. The mappings are applied as

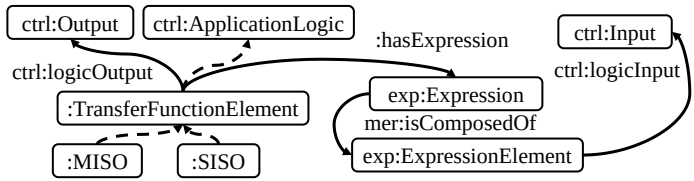


Figure 4.19: Concepts and relationships of the `TransferFunctionElementOntology` [SWO19].

defined in the methodology section of this chapter.
The defined ontology is used to represent a commonly known transfer function element: The proportional or P-Element. Its transfer function can be denoted as $y = K \cdot u$, where y is the output variable, K is a parameter specifying the proportional influence of the input u of the element. In Figure 4.20 an excerpt of instances to represent an arbitrary P-Element are given.

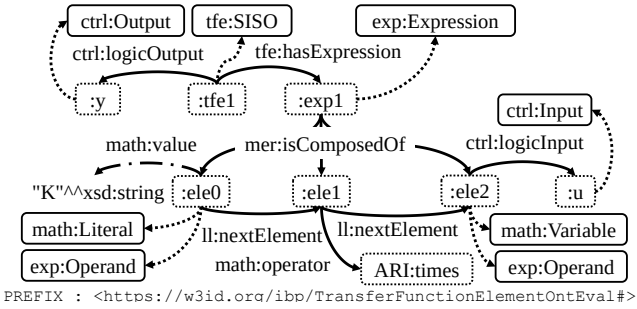


Figure 4.20: Excerpt of individuals to represent a P-element [Abe10] and its transfer function $y = K \cdot u$.

The element classifies as a `SISO` and its expression can be described via the concepts and relationships of the expression ontology (see Section 4.4.1).

4.4.6 UML State Machines

State machines as defined in the UML [Obj15a] are a generic way to describe discrete behaviour of actors. By this ability the UML state machines are a popular way to specify discrete behaviour of control logic. UML state machines are based on the finite state machine formalism and many precursors exist [Mea55, Har87, Lun09, LS17]. For the formal modelling of UML state machines in this thesis an earlier work presented by Dolog [Dol04] is taken into account. Significant improvements and changes have been applied to update the model to comply to version 2.4 [Obj15a] of the UML standard. Additionally, the ability to explicitly model expressions is added, which will be detailed later in this section. The class diagram illustrating the modelling is given in Figure 4.21. The taxonomy including children of the `PseudoState`

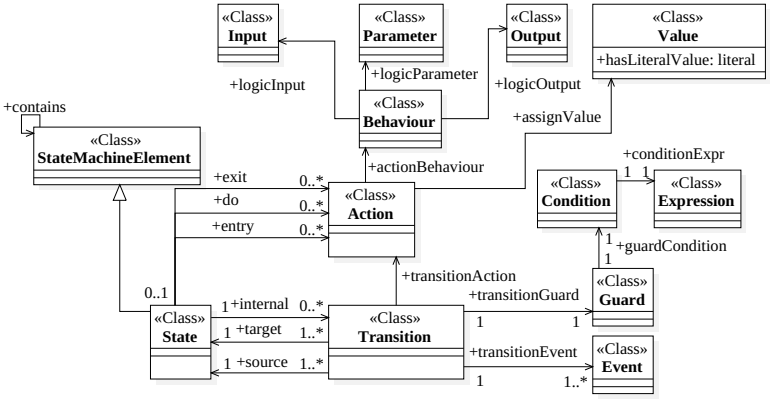


Figure 4.21: Class diagram of a model to describe UML state machines [Obj15a].

class and and children of class `State` is separately displayed in Figure 4.22.
Central to state machines is the concatenation of `State`s and `Transition`s to a network of alternating states and transitions. A transition has one state as `source` and one other or the same as a `target`. States in contrast might be connected to one or more transitions. In the UML `internal` transitions can be specified, which have as the source and target the same state. The `Action`s of a state can be related to it using the directed associations `entry`, `do` and `exit` similar for the transitions using the `transitionAction` association. The actual `Behaviour` of `Action` can be specified by any algorithm or even another state machine. The `Behaviour` class acts as a place holder in this regard. Connections to `Input`, `Parameter` and `Output` of `CTRLont` presented in Section 4.3 can be established using the specified relationships. For simple behaviours, where only a value is assigned to some output if the

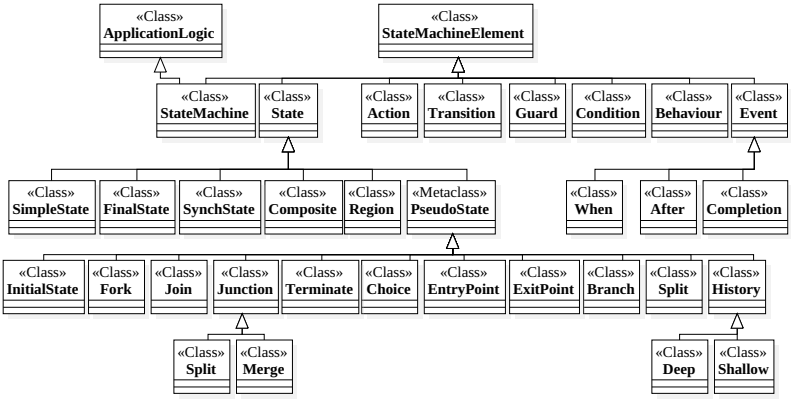


Figure 4.22: Class diagram of the taxonomy of a model to describe state machines from the UML.

state is active, the action can be related to its value via the `assignValue` relationship, which ultimately defines the value by the attribute `hasLiteralValue`.
Transitions are triggered by an `Event` and can be inhibited by a `Guard`. The `Condition` of this guard can be explicated by specifying the `Expression` associated to it via the `conditionExpr` relationship. The expression holds a boolean condition, which can be explicitly described using the expression model presented in Section 4.4.1. A transition its guard and its condition can be related to each other via the `transitionGuard` and `guardCondition` relationships. To describe that a `StateMachineElement` has other state machine elements as parts the `contains` relationship can be used.
A state can be further specified in `SimpleState`, `FinalState`, `SynchState`, `Composite`, `Region` and a number of `PseudoStates`. They share the common semantics of a state but each have their specific meaning in the UML. The pseudo states `InitialState`, `Fork`, `Join`, `Junction`, etc. refer to specific concepts in a UML state machine. Their specific use can be found as described in the standard [Obj15a] or in Rupp *et al.* [RQd12]. Also, an event can be further specified into `When`, `After` and `Completion`, respectively.
The presented semantic model to explicitly model UML state machines is formalised using the defined mapping methodology (see Section 4.1.2). The result is depicted in Figure 4.23.
The defined ontology is evaluated in a simple example to model a UML state machine. The state machine is depicted in Figure 4.24a and an excerpt of the resulting triples are depicted in Figure 4.24b. The described behaviour involves, after being initialised, the transition to a simple state `Init` and then if the condition $R1\_Inp <= 0.0$ evaluates to true the transition

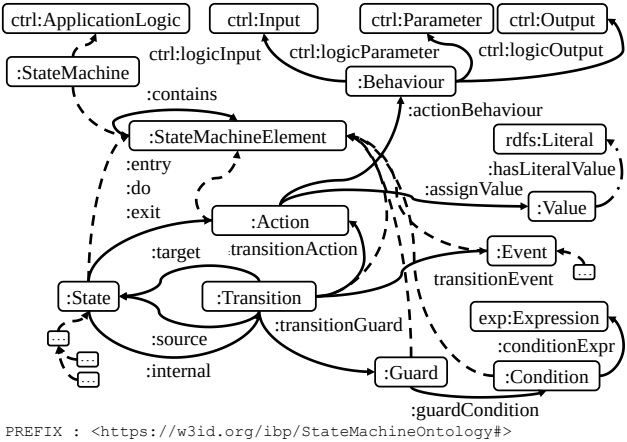
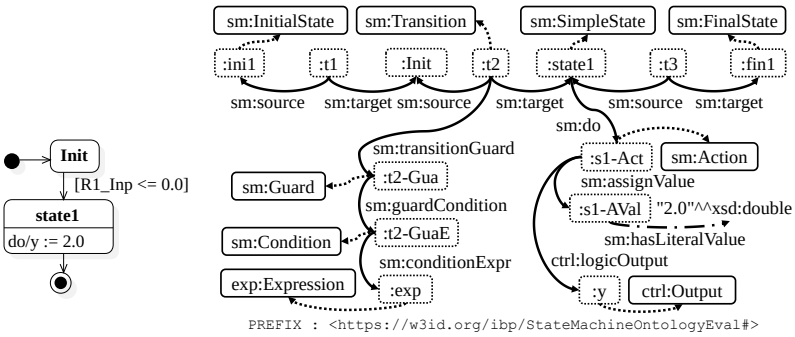


Figure 4.23: Concepts and relationships of the `StateMachineOntology` [SPS17, SWO19]. Subclasses of the `State` and `Event` classes omitted.



(a) UML state machine. (b) Excerpt of instances.

Figure 4.24: Excerpt of instances of a UML statemachine [Obj15a] described using the `StateMachineOntology`.

to `state1`. The condition of the transition can be explicitly described using the expression ontology presented in Section 4.4.1 and the respective triples of the expression are additionally visualised in Figure 4.8. `state1` has one `do` action, which causes the output `y` to be set to the value `2.0`. Then a transition to the final state takes place, when the default `:Completion` event triggers the transition (not depicted). The described ontology is able to describe the concepts of the defined state machine. In particular, the description of the expression of the boolean condition and the relationship to the inputs and outputs of the respective control actor can be explicitly defined. An evaluation in two elaborate use cases is undertaken in Section 5.2.

4.4.7 State Graphs from VDI 3814-6

A special kind of state graphs is specified in the German national standard VDI 3814-6 [VDI08], which allows the specification of state-based control logic and it has been specifically designed for its application in BAS. Compared to state machines it is less complex as it only provides a limited amount of modelling options. For instance, it cannot be specified whether an action is an entry or exit action and the behaviour of an action only involves the assignment of a value to an output.
A semantic model to describe state graphs is presented in Figure 4.25. The design of the model

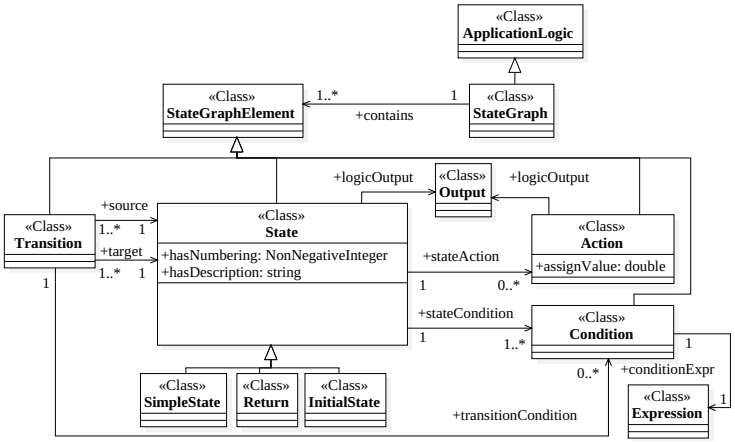


Figure 4.25: Class diagram of a model to describe state graphs according to VDI 3814-6 [VDI08].

considers modelling patterns of state machines from the UML presented in Section 4.4.6.

In a state graph a `Transition` always has as one `State` as a `source` and one or the same as its `target`. A `Condition` specified by an `Expression` determines, whether a transition from one state to another is conducted. In state graphs conditions are associated to a state, however, if several transitions are possible the condition additionally needs to be related to its associated transition via the `stateCondition` and `transitionCondition` relationships, respectively. The `Actions` associated to a state via the `stateAction` relationship assign a value to an output (`assignValue`), if the respective state is active. The `logicOutput` relationship establishes a connection between the action and the respective `Output` as well as the state to determine its activity.
The state graph itself `contains` other `StateGraphElements`, where no sub-state graphs are eligible. Specialisations of a state are a `SimpleState` describing a simple state, an `InitialState` describing the single, initial state of the state graph and a `Return` state, which acts as a placeholder to forward from one state to another. Properties of a state are its `hasNumbering` and `hasDescription` as illustrated.
The ontology formalising the presented model is illustrated in Figure 4.26. The ontology im-

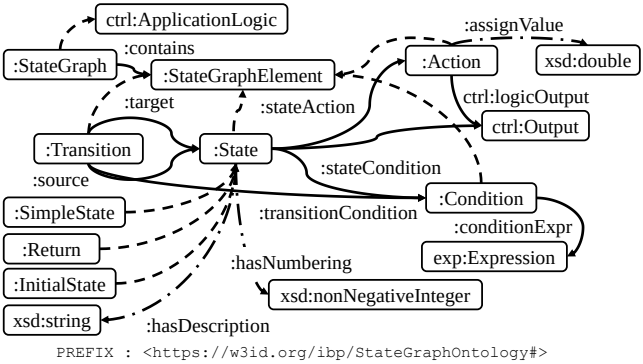


Figure 4.26: Classes and relationships to formalise state graphs using the web ontology language according to the standard VDI 3814-6 [VDI08] [SPS17].

plemented in OWL is created in compliance to the initially in this chapter defined mapping. An additional value restriction is set towards classifying automatically an initial state of a state graph by the data value associated to its numbering. The restriction is listed in Code 4.3.

Code 4.3: Value restriction on the data type property `hasNumbering` in turtle syntax [BBLPC14] to automatically classify an initial state in a state graph as defined in the `StateGraphOntology`.

```
0 :InitialState rdf:type owl:Class ;
```

```
owl:equivalentClass [ rdf:type owl:Restriction ;
                      owl:onProperty :
                          hasNumbering ;
                      owl:hasValue 0
                    ] .
```

For a simple evaluation instances are created, which explicitly describe the state graph presented in Figure 4.27 (left of dashed line). In a state graph one and only one state can be the

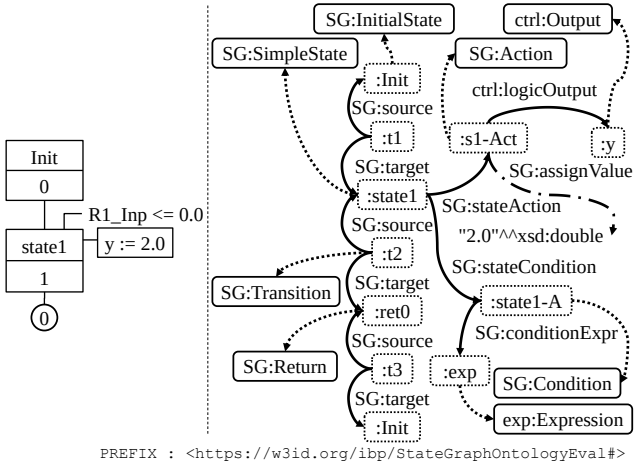


Figure 4.27: A simple state graph (left of dashed line) represented by an excerpt of instances (right of dashed line). The modelling of the expression is illustrated in Figure 4.8.

initial state. This state needs to be related to the value `0` via the `hasNumbering` data type property. A transition from the initial state `Init` to the simple state `state1` is conducted if the initial state is active and the condition associated to `state1`, $R1\_Inp <= 0.0$ evaluates to true. When `state1` is active the output `y` is set to a constant value of `2.0`. The simple evaluation shows that the concepts of the state graph can be described. Moreover, the boolean condition indicating, whether a transition may fire if the respective previous state is active can be explicitly defined and the respective inputs of the control actor can be related to it explicitly. Additionally, the actions including the assignment of values to outputs can be described. A more detailed evaluation of formally describing state graphs is conducted in Section 5.1, where a use case related to the automation of an AHU using a state graph is studied.

4.5 Summary

The layered, semantic model presented in this chapter closes the gap of the absence of a semantic model for the explicit, formal description of knowledge of the automatic control and control logic domains in automation systems and the integration thereof with adjacent domains. The layered model is designed with respect to the in Chapter 3 defined requirements.
Initially, the modelling methodology is presented in Section 4.1. All models are specified using the UML [Obj15a, RQd12] and implemented using OWL [W3C12], which are both manufacturer independent (see Req. 3.1.2) open standards. Additionally, the implementations are accessible freely on the web via their URIs from an open-source repository[6] to stipulate reuse, sharing and revision. Through the specification in UML the model can be implemented in the technology of choice (see Req. 3.1.3). By using OWL (see Req. 3.2.2) for implementation, Req. 3.1.4 is satisfied as it is a machine-interpretable language. It allows to implement an ontology as required by Req. 3.2.1. The underlying DL enable to choose an appropriate profile for the deployment of knowledge-based methods as demanded by Req. 3.2.3.
A layered model architecture is established as described in Section 4.2, where on the upper layer a high-level abstraction of the automatic control domain is separated from domain specific extensions for the explicit, formal description of control logic on the underlying layer. The layered architecture allows to link the automatic control domain to adjacent domains (Req. 3.3.3) and, hence, a link can be established across the domains of control logic, automatic control and adjacent domains. An example for this is presented in Section 5.2.
The `CTRLont` and its implementation as an OWL ontology presented in Section 4.3 provide the possibility for a formal and semantically precise definition of the automatic control domain (see Req. 3.3.1). In particular, it is possible to specify the hierarchy among control actors and their logical topologies (see Req. 3.3.2). The defined concepts for inputs and outputs can be used to establish a relationship to adjacent physical domains (see Req. 3.3.3). As this is use case specific no further details are provided. Instead, the reader is referred to Chapter 5, where the domain descriptions of the automatic control and control logic domains are linked to adjacent domains. Additionally, the semantics of inputs, outputs and parameters of control actors can be specified (see Req. 3.3.4, see Section 4.3) and `CTRLont` is designed in a modular fashion to enable extension (see Req. 3.3.7) and further addition of explicit formal descriptions of control logic in automation systems (see Req. 3.3.5).
A non-exhaustive set of explicit formal descriptions of diverse types of control logic is presented in Section 4.4. The formal descriptions can be used simultaneously, where a one-to-one mapping holds between each control actor and its control logic. This includes algebraic expressions in control logic (Section 4.4.1), schedules (Section 4.4.2), sequence control (Section 4.4.3), discrete two-point control (Section 4.4.4) and generic transfer function elements

[6] `https://w3id.org/ibp`, Last accessed: 22 October 2018

(Section 4.4.5). Additionally, explicit formal specifications of the following state-based control logic types are presented: UML state machines (Section 4.4.6) and state graphs from VDI 3814-6 (Section 4.4.7).
Table 4.3 provides an overview on the OWL implementations of the modules of the layered, semantic model. Only the `ExpressionOntology` concepts and properties are found suitable for reuse (see Req. 3.3.6).
The semantic models presented in this chapter are the basis for two knowledge-based methods, which are presented in Chapter 5.

Table 4.3: Statistics of the OWL implementations of the semantic models presented in Chapter 4 reported by the Protégé tool [Mus15], DL - Description Logics.

Section	Ontology	Logical axioms	Classes	Individuals	Object Properties	Data type properties	Ontology Reuse	DL Expressivity
4.3	CTRLont	44	13	1	15	0	No	$\mathcal{ALCI}$
4.4.1	ExpressionOntology	36	13	16	6	2	Yes	$\mathcal{ALCI}$
4.4.2	ScheduleOntology	53	15	5	12	3	No	$\mathcal{ALCOQ}(\mathcal{D})$
4.4.3	SequenceControl-Ontology	44	14	1	11	3	No	$\mathcal{ALEQ}(\mathcal{D})$
4.4.4	TwoPointDiscrete-Ontology	4	8	0	4	1	No	$\mathcal{ALC}(\mathcal{D})$
4.4.5	TransferFunctionEle-mentOntology	6	10	0	4	1	No	$\mathcal{AL}(\mathcal{D})$
4.4.6	StateMachineOntology	69	28	1	13	2	No	$\mathcal{ALEQ}(\mathcal{D})$
4.4.7	StateGraphOntology	46	13	1	9	3	No	$\mathcal{ALCQ}(\mathcal{D})$
B.1	BasicDatatype-Ontology	93	47	47	0	0	No	$\mathcal{AL}$

5 Validation

Within this chapter two separate knowledge-based methods are presented, which support and automate the engineering and operation of control logic in automation systems. Each method is tested in a use case. A prerequisite for both methods is the formal, explicit representation of knowledge of the automatic control and control logic domains in automation systems. This is enabled by the semantic model presented in Chapter 4. Hence, by utilising the proposed model in the use cases, its ability to fulfil the knowledge requirements of the respective knowledge-based methods is validated.

In Section 5.1 the utilisation of the described models is studied in a use case originating from building automation. The defined ontologies are used to formalise control logic from BAS. The formal knowledge is then used to enable a method for the automated rule-based verification of designed control logic [SPS17]. The method allows to check in monitoring data, whether the control logic is implemented as originally specified in the design. The method is evaluated in a simulation-based study related to the automation on an AHU.

The next use case presented in Section 5.2 studies the support and enhancement of the engineering of control logic in industrial automation. A knowledge-based method is investigated to allow automated, formal verification across control logic types and plant knowledge, incremental verification of the control logic and bidirectional exchange of control logic knowledge as designs and implementation evolve over the project life cycle [SWO19]. The developed method is deployed in a scenario-based evaluation related to the automation of a batch process available from literature [KSB01].

The findings from the validation in the two use cases are summarised in Section 5.3 with reference to the in Chapter 3 defined requirements.

5.1 Automated Rule-Based Verification of Designed Control Logic in Building Automation Systems

5.1.1 Problem Description

The design process of an automation solution for BAS as defined in the standard ISO 16484-1 [ISO10] incorporates the planning, technical design, installation, commissioning and test op-

eration. In particular, in public construction projects the planning/technical design and the installation/commissioning is executed from different project teams as this is stipulated by the procurement process. This requires the documentation of the designed automation solutions, such that it can be implemented at a later point in time. Often the team taking over does not have the full information and knowledge to understand all design decisions and a loss of information and knowledge at every information hand over is recognised. This documentation typically constitutes of spreadsheets, drawings and textual descriptions as mandated by the current standardisation bodies [ISO11]. It is almost impossible for the designer of a certain control logic to verify if the designed control logic has been implemented as originally specified. Additionally, the actual source code implementing the control logic is usually concealed out of intellectual property reasons by the manufacturers of BAS equipment.
A semi-automatic method to automatically detect mismatches between the actual designed control logic and its implementation is proposed in Plesser *et al.* [PFP+10] and Plesser [Ple13]. The method requires to manually mine the specification of the designed control logic from its documentation by trained experts. Subsequently the expert defines verification rules depended on the control design. For large installations the documentation can easily constitute of thousands of pages with textual descriptions, tables and drawings. By using the defined rules the method checks monitoring data, if the observed behaviour corresponds to the designed one.
The process to manually acquire and process the knowledge on the actual design of the control logic from its documentation is cumbersome, time and cost-consuming. Within the next section a method is presented, which provides means to automate the approach by Plesser [Ple13].

5.1.2 Methodology

In the following a methodology is proposed as illustrated in Figure 5.1, where the mentioned approach by Plesser [Ple13] is automated by formalising the control logic design knowledge from its documentation and store the knowledge in a knowledge-base. The stored knowledge is utilised to automatically configure and deploy parametrised queries encoding knowledge to verify specific control logic types.
Initially the as-designed control logic of a BAS installation is formally specified and stored in a knowledge-base. Next, a query is executed, which retrieves necessary information to configure previously encoded archetype rules. The rules are deployed to evaluate monitoring data and return as a result points in time where the system detects a symptom of a fault [TVM+09]. The detected faults are stored in the knowledge base for further post-processing and fault isolation [KB05]. The execution of the method can be implemented, and hence, automated.

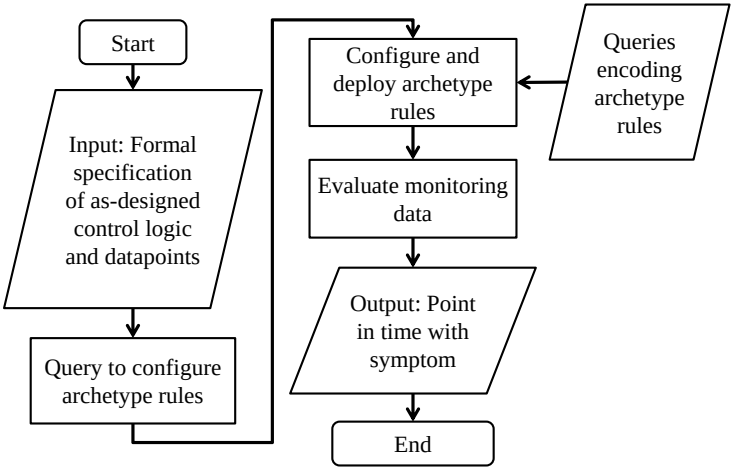


Figure 5.1: Flowchart to describe the procedure to execute the automated rule-based verification of designed control logic in building automation systems [SPS17].

5.1.3 Use Case and Implementation

The described knowledge-based method for automated rule-based verification is implemented as a prototype using the Python programming language. In the following, the studied use case related to an AHU is described. Then, two scenarios are presented, where the method is used to detect two typical faults occurring frequently in the operation of control logic in BAS.

Description of the AHU

The technical equipment of the AHU (fans, heating coil, pump, temperature sensors, etc.), the logical topology of the designed control logic for the AHU and the control logic itself are depicted in a control diagram presented in Figure 5.2.

The AHU retrieves outdoor air and heats, if required, this air through a heating coil before supplying it to the served building. The air returning from the building can be either recirculated via a mixing box or disposed to the exhaust. As sensor inputs the outdoor air temperature T_{oa}, the heating coil return water temperature T_{hcr} and the current time are used. As outputs the normalised output signal of the mixing box damper Y_{Mix}, the normalised output signal of the supply and return fans Y_{Fan} and the normalised circulation pump output signal Y_{Pump} are utilised.

A simple control logic is designed for the described AHU consisting out of two control actors being a state graph (R1) and a schedule (R2). The state graph [VDI08] operates the AHU according to the inputs of the heating coil water return temperature, the outdoor air temperature

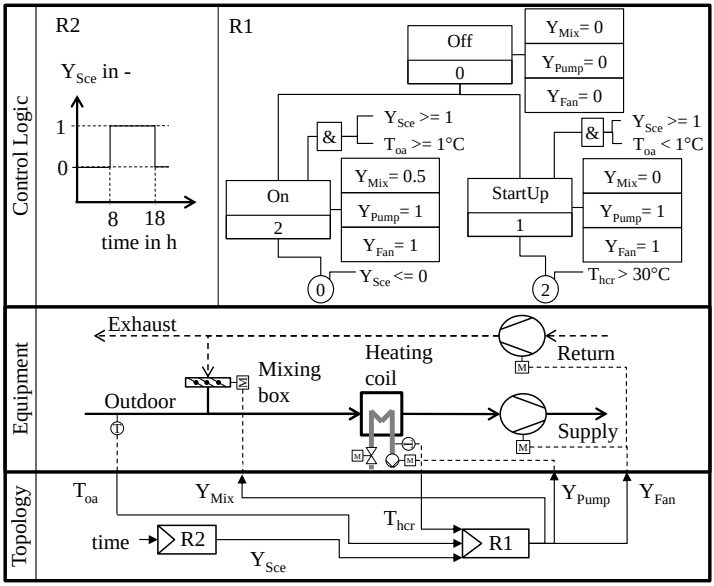


Figure 5.2: Control diagram of the studied air handling unit according to ISO 16484 [ISO11]. Y_{Sce}, Y_{Mix}, Y_{Fan}, Y_{Pump} - Normalised output signal from schedule, mixing box damper flap, fan and pump, respecitvely; T_{hcr} - Heating coil return water temperature, T_{oa} - Outdoor air temperature [SPS17].

and the normalised output signal of the schedule. The schedule allows operation only between 8am and 6pm using the current time as an input and giving a normalised output signal Y_{Sce}.
The state graph specifies the following behaviour. Initially the state `Off` is activated. If the schedule signal is 1 and the outdoor air temperature is above 1 degree Celsius a transition from state `Off` to state `On` is executed. The state graph turns back to state `Off`, if the schedule signal turns back to zero. A different behaviour appears when the outdoor air temperature is below 1 degree Celsius. To avoid the freezing and potential damage of the heating coil, first the `StartUp` state is active. Within this state the mixing box damper is kept closed (Y_{Mix} := 0) such that only return air recirculates in the system while fans (Y_{Fan} := 1) and pump (Y_{Pump} = 1) are operating. If the heating coil water return temperature rises above 30 degree Celsius the state graph continues in normal operation mode (state `On`).
The described AHU is modelled using the Modelica modelling language [Mod17a] and models from the `Buildings` library [WZNP14]. The implementation uses the `Modelica`

`Standard Library` version 3.2.2[1] and `Buildings` library version 4.0.0[2]. The state-based control logic of the state graph is implemented using the `Modelica.StateGraph` [OÅD05] library available from the Modelica standard library. Code snippets from the implementation as referenced in the text can be found in Section C.1.

Description of the Scenarios

Two scenarios are considered for the testing of the knowledge-based method:

1. Faulty operation of the state graph by artificially creating a fault in its operation. It is assumed that for some maintenance task the pump was manually forced[3] to operate. However, the signal Y_{Pump} was not unforced and thus the pump is operating constantly after finishing the maintenance actions.
2. Faulty operation of the schedule by artificially creating a fault. It is assumed that the time of the BAS installation did not change after the change from winter to summer time, thus, the operation of the schedule is shifted by exactly one hour.

The presented control logic design as illustrated in Figure 5.2 is formally represented using the ontologies defined in Chapter 4. In addition, the SEAS System ontology [LKGZ17] is reused to describe the connection of inputs and outputs of control actors to sensors and actuators of the BAS. An excerpt of triples formally representing the control logic design is given in Figure 5.3. The triples are manually formalised, however, the knowledge can be extracted automatically by implementing adapters to domain expert tools, such as TRIC[4] or WEBPROJECT[5].

Verification Knowledge of State Graphs

For BAS implementing a state graph according to VDI 3814-6 following knowledge can be defined to verify if the implemented state-graph operates as-designed:

Definition 5.1 *If a state is active, then its actions need to be executed [Ple13].*

In Figure 5.4 the deployment to verify the correct behaviour of a state graph with the generic procedure described in Figure 5.1 is illustrated. The design of the state graph is formalised compliant to the `StateGraphOntology` presented in Section 4.4.7 and stored in a triple store. The knowledge as defined in Definition 5.1 is implemented in a parametrisable SPARQL

[1] `https://github.com/modelica/Modelica/releases/`, Last accessed: 22 October 2018

[2] `https://github.com/lbl-srg/modelica-buildings/releases`, Last accessed: 22 October 2018

[3] Forcing variable values in PLCs is a common practice, where a human operator can manually set values in the implementation. The forced variables can remain at the forced value even when returning into automated operation mode.

[4] `https://www.tric.de/`, Last accessed: 22 October 2018

[5] `http://www.webproject-portal.de`, Last accessed: 22 October 2018

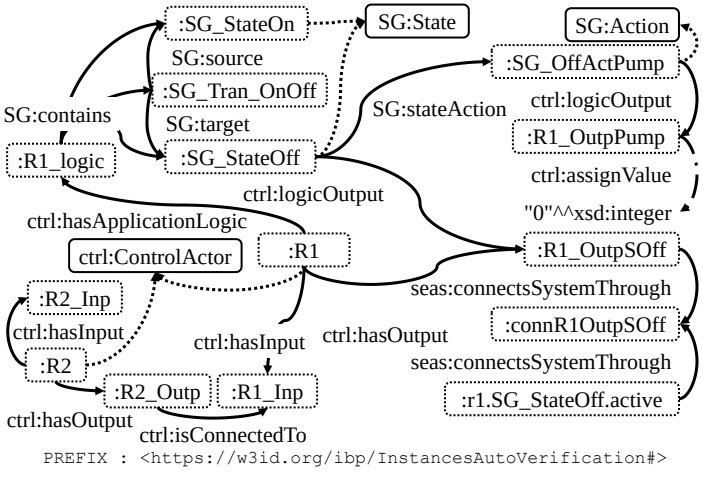


Figure 5.3: Excerpt of instances as considered in this use case (see Figure 5.2). Adapted from Schneider *et al.* [SPS17].

query. An additional SPARQL query retrieves the knowledge to configure instances for a specific state graph and these instances then can be deployed to check in monitoring data if the state graph performs as specified in the design. Points in time where a mismatch is detected are reported for further analysis.

The respective query encoding the archetype rule using SPARQL is given in Code C.1. The query to retrieve the respective knowledge for configuring the archetype rule for arbitrary state graphs stored in the triple store is given in Code C.2 and example results of this query for this use case are denoted in Table C.1.

Verification Knowledge of Schedules

For schedules the following rule can be defined to verify their correct behaviour:

Definition 5.2 *If an interval of a schedule is active then the output of the schedule should be equal to the specified output of this interval.*

Figure 5.5 illustrates the deployment of the method for the verification of a schedule. The schedule design is formalised compliant to the `ScheduleOntology` and stored in a triple store. This knowledge is retrieved by a query to configure and deploy a parametrised query. The instantiated queries then are utilised to check monitoring data if the implemented schedule behaves as designed. Points in time where a fault is detected are reported for further analysis.

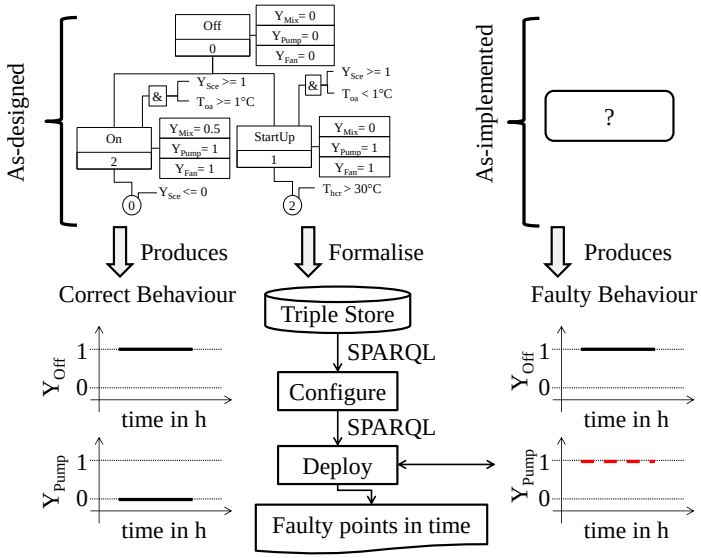


Figure 5.4: Visualisation of the procedure to detect faulty behaviour in state graphs. The as-designed state graph is formalised and stored in a triple store. This knowledge can then be used for the detection of faulty points in time. Y_{Pump} - Normalised output signal from pump, Y_{Off} - Normalised output signal of state off, SPARQL - [PS17].

The knowledge as defined in Definition 5.2 to verify schedules can be formalised as a SPARQL query. A parametrised query implementing this is given in Code C.3. The query to retrieve the respective knowledge for configuring the archetype query listed in Code C.3 is presented in Code C.4.

The configured queries for both rules are compiled into one resulting query by concatenating them via `UNION` statements [PS17]. Code C.5 lists an example query, which is run against the knowledge base hosting the monitoring data. All queries are inserted as sub-queries to ensure efficient query execution by the SPARQL end point. In this example the readings of the simulated monitoring data and the formal description of the control logic are hosted in one triple store as the amount of data is small (˜100000 triples). For large scale deployment of the method it is recommended to host the time series data in a relational data base and access it via ontology-based access of relational data bases, e.g. R2RML [DSC12].

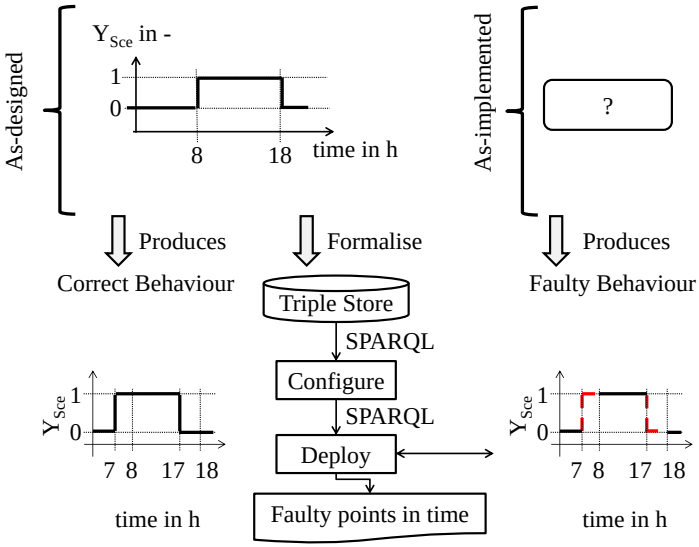


Figure 5.5: Visualisation of the procedure to detect faulty behaviour in schedules. The as-designed schedule is formalised and stored in a triple store. This knowledge can then be used for the detection of faulty points in time. Y_{Sce} - Normalised output signal from schedule, SPARQL - [PS17].

5.1.4 Results

The described simulation model of the AHU is simulated for 48 hours to generate the respective observations for analysis by the implemented method.

In Figure 5.6 results from deploying the described method to the first scenario with the erroneous, continuous operation of the pump is presented. In the following the depicted results starting from the top of the figure to the bottom of the inclined plot are presented. The activity of state *Off* over the duration of the 48 hours is depicted in the top graph. Below for comparison the correct outputs of the normalised control signals of the mixing box damper flap Y_{Mix}, the pump Y_{Pump}, the fans Y_{Fan} and the schedule Y_{Sce} are presented. Next, the respective signals, which are the respective inputs for the analysis by the automated rule-based verification method are given, where the pump signal Y_{Pump} indicates the erroneous operation of the pump by staying at a constant value during the full 48 hours. Finally, in the bottom graph the results of evaluating the monitoring data with the implementation of the automated rule-based verification method are presented. Three different results are presented here: First, points in time

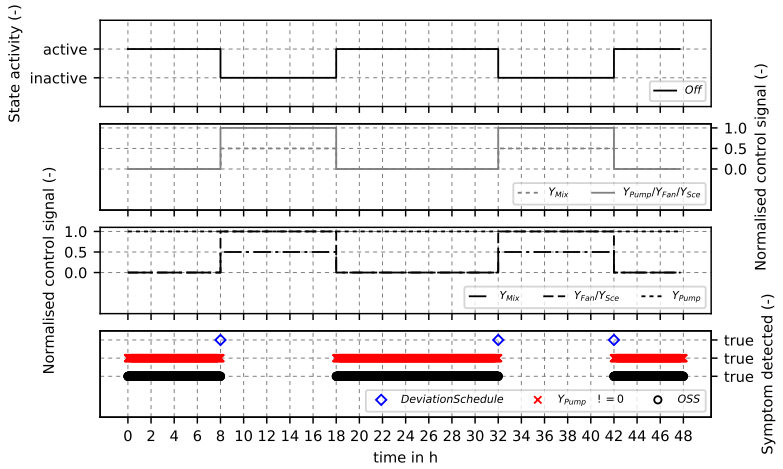


Figure 5.6: Results from scenario with faulty state graph behaviour; Subplots numbered from the top to bottom of the graph: (1) Activity of state *Off*; (2) Correct normalised control signals of the pump Y_{Pump}, mixing box damper flap Y_{Mix}, fans Y_{Fan} and schedule Y_{Sce}; (3) Faulty normalised signals as analysed in the scenario; (4) Faulty points in time for the schedule (*DeviationSchedule*), the state graph (Y_{Pump} != 0) and the Accumulated Weighted Operational Quality (OSS) (*OSS*).

are given where a deviation of the schedule is detected (*DeviationSchedule*); Second, results indicating if there is a mismatch for the state actions detected (Y_{Pump} != 0); and Finally, the OSS [Ple13] determined from aggregating the previous results are reported.

Points in time detecting deviations of the schedule are found in some cases (e.g. at 8h). The reason for this is that within the simulated data two values at one point in time occur when discontinuities in the simulation force the solver to reinitialise the equation system. This occurs at the switching times of the schedule and is detected by the rule. The mismatch of the state actions with the constant operation of the pump is detected by the system. Note, fault detection is one step in FDD [KB05]. Further evaluation to determine the root cause of this fault need to be put into place. Automating this process is proposed in Dibowski *et al.* [DHR16a].

In Figure 5.7 results are presented for the simulation of the AHU according to the second scenario where the schedule is operating not as specified from 8am to 18am but from 7am to 17am. In the top graph the correct output $Y_{Sce,Corr}$ and the faulty signal of the schedule as simulated in the second scenario $Y_{Sce,F}$ are depicted. The lower graph in Figure 5.7 presents the results from evaluating a deviation of the schedule (*DeviationSchedule*), the verification of the correct execution of state off (Y_{Pump} != 0) and the *OSS*. As expected, no faulty behaviour is

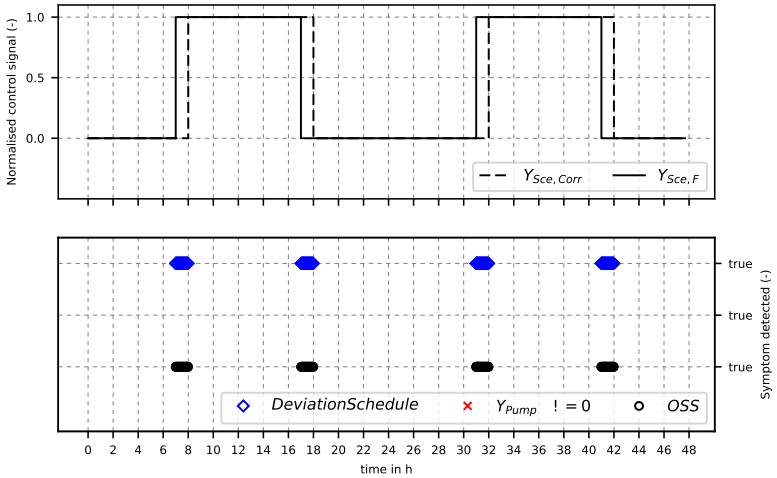


Figure 5.7: Results from the scenario with faulty schedule behaviour; Top graph: Correct $Y_{Sce,Corr}$ and faulty $Y_{Sce,F}$ normalised control signal of schedule. Bottom graph: Faulty points in time for deviation of schedule (*DeviationSchedule*), the verification of the correct execution of state off (Y_{Pump} != 0) and the OSS (*OSS*).

detected from verifying the designed state graph. The rules verifying the schedule behaviour correctly identifies the points in time where the schedule is not operating as specified (Between hours 7-8, 17-18, 31-32 and 41-42).

5.2 Knowledge-Enhanced Engineering of Control Logic in Industrial Automation

This section presents a knowledge-based method for the support and enhancement of knowledge-intensive tasks related to engineering of control logic in IAS. To specify domain knowledge the ontologies presented in Chapter 4 are utilised. The addressed, specific problems related to the engineering of control logic are presented in Section 5.2.1. Next, in Section 5.2.2 the knowledge-based methodology and the developed KBS is detailed. Finally, in Section 5.2.3 the implementation of the KBS and in Section 5.2.4 results from deploying the methodology in a scenario-based evaluation related to the engineering of the control logic of a batch plant available from literature [KSB01] is presented.

5.2.1 Problem Definition

The engineering of control logic in automation systems is a challenging task and requires methods and tools from software, systems and automation engineering [KRS12]. The current state of the art as well as open research questions are defined and summarised by the reviews presented by Vyatkin [Vya13] and Vogel-Heuser *et al.* [VHDF+14, VHFST15]. The method and tools presented here are in response to some of the demands defined by the cited reviews.
Figure 5.8 illustrates three commonly identified problems in the engineering of control logic in

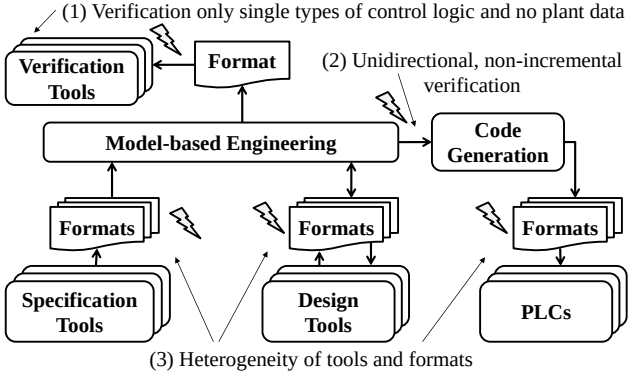


Figure 5.8: Common identified problems in the engineering of control logic in automation systems [Vya13, VHDF+14, VHFST15]. (1) Verification of single types of control logic without plant data; (2) unidirectional code generation in model-based engineering; (3) Heterogeneity of tools and formats.

automation systems, which are specifically addressed within this use case.
The first problem (1) relates to the missing abilities of existing automated verification mechanisms. Automated verification is demanded as a key method to ensure safe and reliable execution of control logic in automation systems [Vya13, VHFST15]. A number of approaches exist for the formal verification of control logic [SWO19], e.g. formal verification of UML models [LMM99, HWD+13, GGA14, KP15], petri nets [Fre02, GWGW17] or programming of reactive systems for simulation and control [HLR92, Hal93]. The described automated verification methods are an important achievement for both academia and industry, however, a common downside is that the mentioned methods only work for one single type of control logic and do not integrate information of plant data into the verification mechanisms as demanded by Vyatkin [Vya13] and Vogel-Heuser *et al.* [VHFST15]. The ties and interdependencies of the physical plants and its control logic are multi fold, hence, verification mechanisms need to reflect this characteristic by taking both the control logic and the plant into account.

The second problem (2) addressed here relates to the unidirectional generation of code frequently occurring in model-based engineering. This problem is observed in model-based engineering of control logic, where in the final step source code is generated to be executed on a PLC, e.g. [HVA16, JSSF17]. However, during the commissioning often changes in this source code happen to fix errors and tune the designed control logic to fit the real plant [VHFST15]. The commissioning staff often consists of experts for the respective plants and PLC programming languages but not for the languages and tools used in model-based engineering [VHFST15]. For documentation purposes and for continuous verification it is required to establish a feedback from the implemented code back to the model in an iterative, incremental fashion. To ensure seamless operation of the designed system and also to keep model-based documentation up to date, the bidirectional exchange of knowledge from the executed source code back to the model is required. Additionally, the newly added knowledge needs to be verified as well.
Finally, the last problem (3) relates to the heterogeneity prevalent in the engineering tool chains in terms of formats and tools. Stakeholders from various disciplines use multiple tools and formats to specify, design and in the end commission and operate an automation system [SH17]. Different machine-readable formats are available, which support this exchange (see Section 2.4), however, the exchange using paper-based formats, PDF and spreadsheets is still very common [SH17]. The negative effects of this heterogeneity needs to be eliminated through semantic integration of tools and formats. Additionally, this aspect is of relevance to enable the envisioned two-layered architectures of future automation systems (see Section 1.2, [KDKO14, VHDB13]), where intelligent components of an automation system communicate from machine to machine.
In essence, to overcome the mentioned problems a solution method should support:

1. 'Interoperability between heterogeneous formats and tools through semantic integration' [SWO19];
2. 'Automated formal verification of the control logic design including different types of control logic [VHFST15] and inclusion of plant information [Vya13, VHFST15]' [SWO19];
3. 'Incremental verification at changes and updates [VHFST15]' [SWO19];
4. 'Bidirectional flow of information from model to target format and vice versa [VHFST15]' [SWO19].

5.2.2 Methodology

In response to the problems outlined in the previous section a knowledge-based methodology is developed in this section. The methodology is realised through the design of a KBS (see Section 2.2.4) illustrated in Figure 5.9.

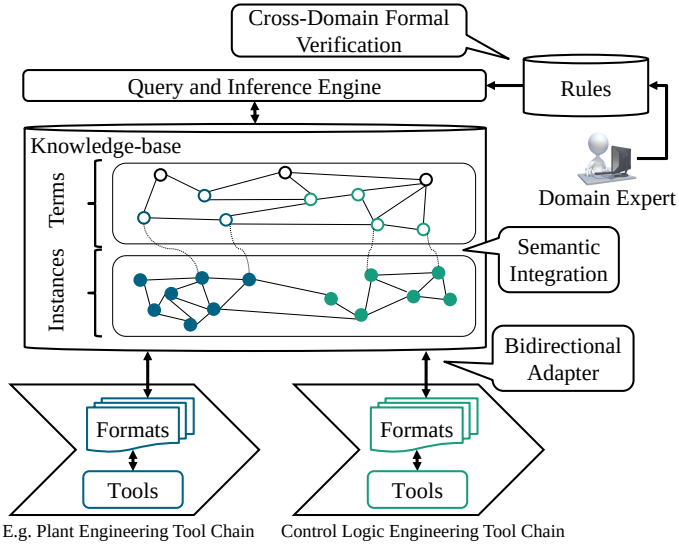


Figure 5.9: Knowledge-based system as designed for this use case to enable automated knowledge-based methods in response to the identified problems (see Figure 5.8).

The core component of the KBS is its knowledge-base. Here three types of knowledge can be identified:

1. Assertional knowledge (Instances) constitutes of project or use case specific knowledge. This knowledge is derived from acquiring knowledge from the various data formats provided by the tools used in engineering tool chains. In particular, *Bidirectional adapters* are implemented to enable the acquisition of knowledge as well as the feedback of improvements to the source formats and tools;
2. Terminological knowledge (Terms) of control logic in automation systems is represented through a domain model, which uses the ontologies developed in this thesis combined with reused ontologies of adjacent domains and;
3. Procedural knowledge (Rules) constitutes of formalised knowledge acquired from domain experts. In particular, this is the case when rule-like statements are formalised, e.g. to allow the *cross-domain formal verification* of control logic including different types of control logic and plant knowledge.

In the described approach technologies from the semantic web are utilised for the implementation as detailed in the following sections. Two main benefits can be identified in this regard:

SWT allow to integrate the heterogeneous formats (*semantic integration*) and tools apparent in engineering tool chains of control logic in automation systems. A review summarising related work in this regard is published by Ekaputra *et al.* [ESS+17]. Furthermore, assertional and terminological domain knowledge from adjacent domains, such as mechanical and electrical engineering of a plant, can be seamlessly integrated through the reuse of respective ontologies. Most importantly the formal nature of SWT with their basis in DL (see Section A.1) allow for the rigorous verification of control logic as desired. For reference, it may be noted that the use of SWT for integrating heterogeneous data formats in engineering tool chains and for the development of KBS related to the engineering of automation systems is described by different authors [Run11, MB12, NSMŠ15, Mos16]. However, the novelty of the approach resides in the use and integration of explicit formal specification of knowledge on control logic in automation systems enabled by the novel model.

The approach presented here can be *automated* as the described modules and tools, such as the *query and inference engine*, can be automatically invoked either on a periodic level or through triggers, such as model update events. This also supports the demanded *incremental verification* capabilities, the verification can be triggered after every update.

5.2.3 Implementation

Plant Description

The 'AST Batch plant' [KSB01] is an experimental batch plant, which is set up to investigate and study control problems in the process automation domain. Its design provides a good trade-off between complexity of the plant to be interesting for research as well as simplicity to be still easily to grasp by practitioners, scholars and students. The physical plant, its components, the processes and its control is described in detail in Kowaleski *et al.* [KSB01]. Moreover, an open-source simulation model of the process is available [PRPO06], which is implemented using the Modelica modelling language [Mod17a].

The Pipes and Instrumentation Diagram (PI&D) showing the components of the process, sensors and actuators is depicted in Figure 5.10.

The components are mounted to a wall, hence, fluids flow from the top (e.g. B1) to the bottom (e.g. B7), if the respective valves are opened. The process realises the desalination of a water sodium mixture (brine) through evaporation. Initially, the brine is stored in tank B1 and fresh water in B2. By opening valve V8 a fixed amount of brine flows in to tank B3. Fresh water is added from tank B2 by opening valve V9 until a certain concentration is reached. The resulting mixture is buffered in tank B4 until it moves forward to tank B5 for evaporation. To evaporate freshwater from the water sodium mixture, tank B5 is heated and the evaporated steam is condensed in a cooled condenser and gathered in tank B6. Both the condensed steam and the resulting high concentrated brine from tank B5 are cooled in tank B6 and B7, respectively. After cooling the resulting fluids are pumped back to the initial tanks through pump P1 and

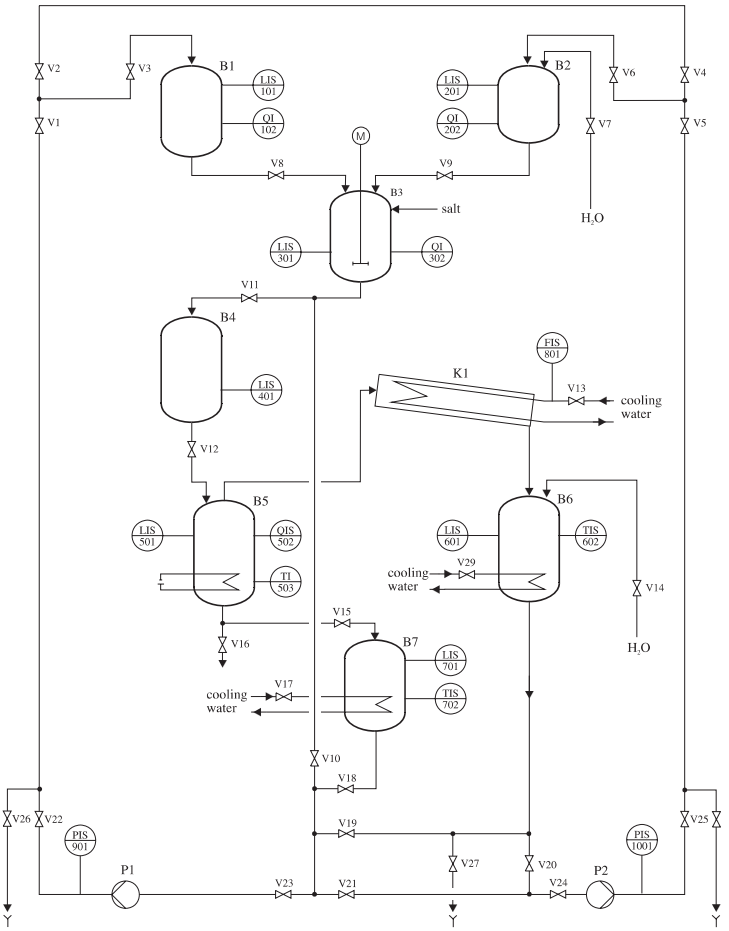


Figure 5.10: Pipes and instrumentation diagram of the studied batch plant [KSB01]. Reprinted from European Journal of Control, Vol. 7, No. 4, Kowaleski, S., Stursberg, O., Bauer, N., An Experimental Batch Plant as a Test Case for the Verification of Hybrid Systems, pp. 366-381, Copyright 2001, with permission from Elsevier under license number 4354061510201.

P2. A detailed description including the available sensors, actuators and time constants of the (sub-)processes is provided in Kowaleski *et al.* [KSB01].
The control logic of the process including some start up processes and the automated operation mode are textually described and an implementation using the SFC formalism [IEC14a] is provided [KSB01]. In this work, a UML state machine presented in Figure 5.11 is specified,

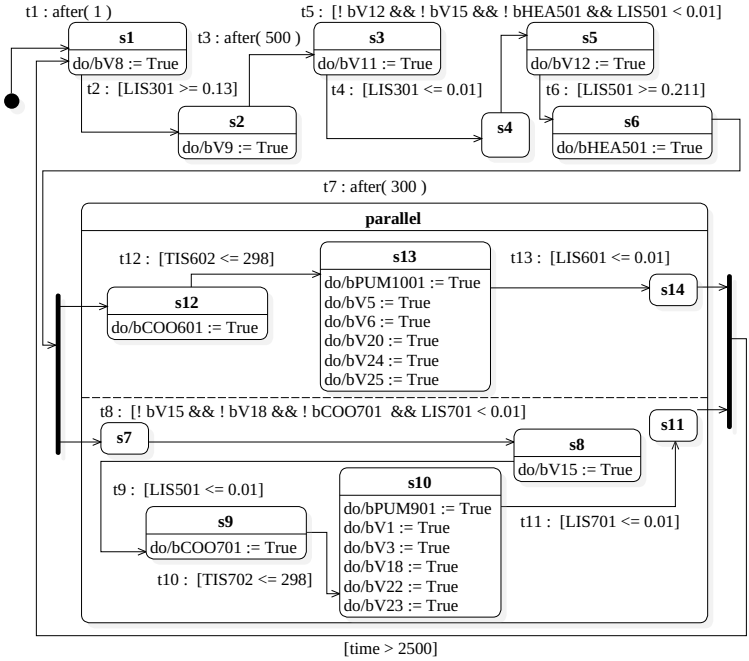


Figure 5.11: UML state machine [Obj15a] specifying the automated state-based control of the AST batch plant [KSB01] (adapted from Poschlad *et al.* [PRPO06]) [SWO19].

which is derived from the automated operation mode of the plant model as defined in Poschlad *et al.* [PRPO06].

Formal Domain Model

A prerequisite for the development of the envisioned knowledge-based methods is the availability of a formal model to describe domain knowledge. In Figure 5.12 a modular ontology is presented, which serves this purpose. Core part of the modular ontology is the `CTRLont` as presented in Section 4.3, which allows to describe high-level domain knowledge on control logic as well as to link explicit specifications of control logic to specifications of the adjacent

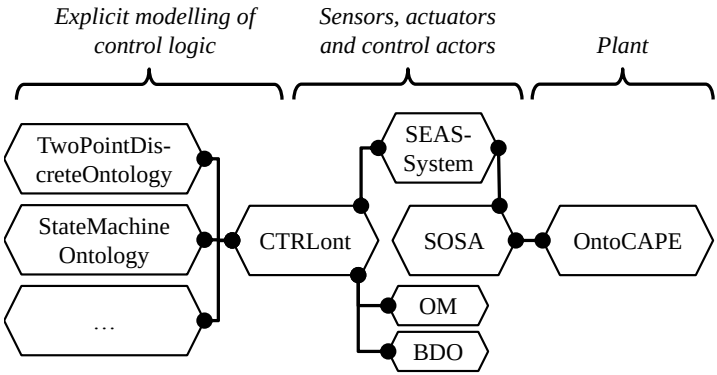


Figure 5.12: Ontology modules to integrate explicit formal models of control logic with adjacent domains, such as sensors, actuators and plants from chemical process engineering [SWO19].

physical system ontologies. To explicitly represent control logic the ontologies presented in Chapter 4 are used. To describe the tangible components of the automation system, such as sensors and actuators, the SOSA [HKC+17] is reused. The connection between systems, e.g. connection of datapoints in an automation system via a bus to a control actors is modelled by using the SEAS System ontology [LKGZ17]. Fragments of the OntoCAPE family of ontologies [MMWY09] are used to describe the physical plant domain. For the modelling of quantities and units of sensors, actuators, inputs, outputs and parameter the Ontology of units of Measure (OM) [RWT11] is used. Basic data types are described using BDO, an ontology bootstrapped from the XSD [BM04] specification.

The use of the modular ontology is illustrated in Figure 5.13 by presenting an excerpt of instances to formally describe a temperature sensor of tank B5. The tank `:B5` is annotated as both a `OC-equ:HeatedTank` from OntoCAPE ontology as well as a `sosa:FeatureOfInterest`. The temperature `:B5-T` of tank `B5` is modelled as a `sosa:ObservableProperty` and related to the tank via the `ssn:hasProperty` object property. The actual sensor `:B5-T-Sensor` `sosa:observes` this property. The connection of the sensor via an automation system is modelled using the `seas:connectsAt` object property. Using object properties and concepts provided by the SEAS System ontology the sensor can be connected to a `ctrl:Input` of the individual `:ctrl1-CA`. Finally, the control logic of this `ctrl:ControlActor` can be specified using one of the described ontologies and is related to it via the `ctrl:hasApplicationLogic`.

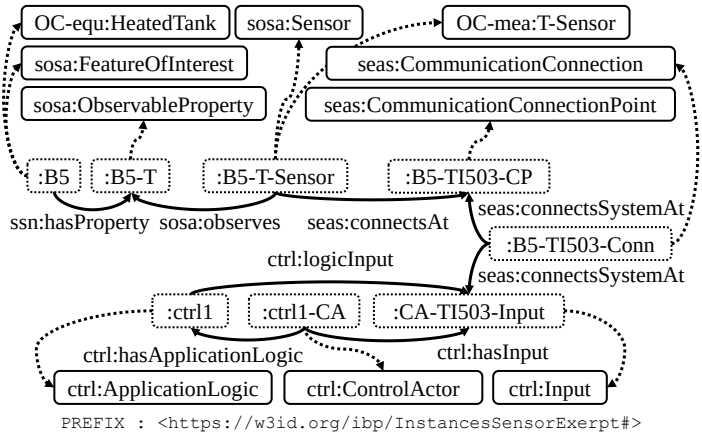


Figure 5.13: Excerpt of instances for describing temperature sensor TI503 in tank B5 of the studied batch plant (see piping and instrumentation diagram in Figure 5.10) [SWO19].

Control Logic Engineering Tool Chain

The knowledge-based support of control logic engineering is studied by deploying the described method in a tool chain used for the engineering of the control logic for the described batch plant. The utilised engineering tools as well as the tools and languages used for the implementation of the described method and KBS (see Section 5.2.2) are depicted in Figure 5.14.

The chosen tools are aligned with the life cycle of control logic engineering (e.g. Julius *et al.* [JSSF17]) spanning the specification of the desired behaviour, the simulation-based testing and prototyping and, finally, the implementation and commissioning on a real PLC. The utilised tools support diverse formats for the exchange of the control logic.

To specify the discrete behaviour of the plant the implementation neutral UML [Obj15a] is utilised via the StarUML tool [MKL18]. The tool supports the export of the designed UML state machine to the XMI format [Obj15b]. For hybrid simulation and prototyping of the initially designed control logic in combination with a plant model the simulation environment Dymola [3ds17] is used, which supports the Modelica modelling language [Mod17a]. Beneficial in this regard is the availability of a plant model provided by Poschlad *et al.* [PRPO06]. For hybrid simulation the discrete control logic is implemented using the `StateGraph` Modelica library [OÅD05]. Finally, for the implementation of the actual control logic in one of the programming languages from IEC 61131-3 [IEC14a] the PLCopen Editor tool [Ber18] is

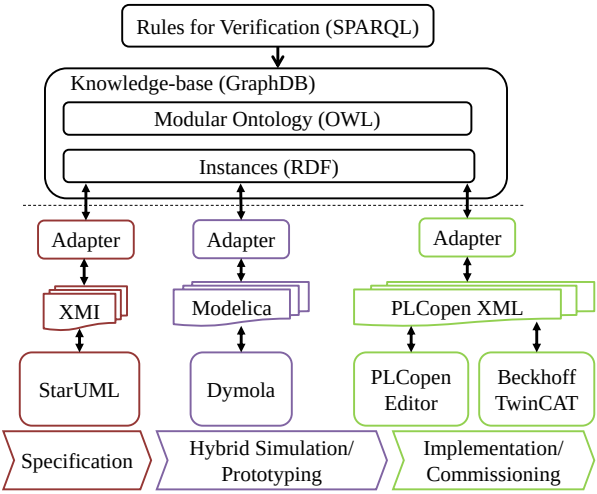


Figure 5.14: Overview on the tools, formats and services utilised in this use case [SWO19]. The utilised tools and formats are explained in the text.

used, which supports the PLCopen XML format [PLC09]. The resulting code is compiled and executed using the TwinCAT 3.1 PLC [Bec18]. To formalise the knowledge contained in the formats provided by the tools custom adapters are implemented using the Python programming language. The implementation is based on the built-in `ElementTree` package for XML parsing, the `rdflib`[6] and `jinja2`[7] packages to generate RDF and the `OMPython`[8] package for handling Modelica source code. The developed code allows the batch processing of the bidirectional conversion between the formats and their formal representation in RDF [CWL14]. The KBS is implemented through hosting the modular ontology and the instances created by the custom adapters in a triple store (GraphDB[9]). The chosen triple store supports both reasoning as well as processing of queries implemented in SPARQL [PS17]. Rule-like expert knowledge used for advanced verification is implemented as SPARQL queries and can be executed on the query engine of the respective triple store. It may be noted that SPARQL is only one possibility to implement the described rule-like expert knowledge. However, in the example

[6] `https://rdflib.readthedocs.io/`, Last accessed: 22 October 2018
[7] `http://jinja.pocoo.org`, Last accessed: 22 October 2018
[8] `https://www.openmodelica.org/doc/OpenModelicaUsersGuide/latest/ompython.html`, Last accessed: 22 October 2018
[9] `https://ontotext.com/products/graphdb/`, Last accessed: 22 October 2018

implementations it has proven to be a versatile tool and, moreover, it is widely supported by tools as a standard. The support of the control logic engineering process through automated knowledge-based verification is detailed in the following section.

5.2.4 Scenario-Based Evaluation

The proposed method is evaluated by investigating a number of scenarios related to the engineering of the control logic of the described batch plant. The expert knowledge for the verification is implemented as SPARQL queries and working code examples are provided in Section C.2, p. 168 and are referenced within the text.

Formal verification of single types of control logic

When specifying control logic using the UML automated verification is possible (see also Grobelna *et al.* [GWGW17] and Khan & Porres [KP15]). From the UML standard [Obj15a] the knowledge specified in Definition 5.3 can be used for verification of UML state machines.

Definition 5.3 *An initial state has one and only one outgoing transition.*

This knowledge can be implemented as a SPARQL query (Code C.6) and if executed by the query engine of the KBS used to identify faulty designs of UML state machines. For instance, Figure 5.15 presents a simple UML state machine using the StarUML tool, which is formally

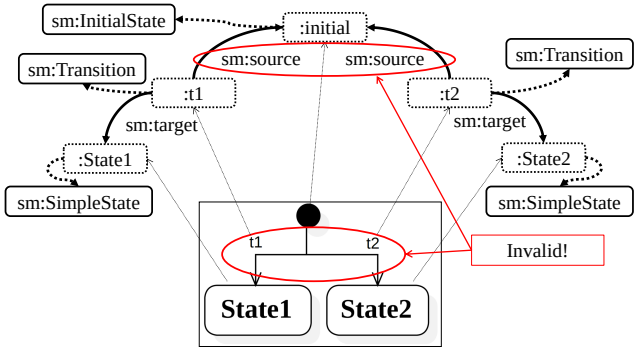


Figure 5.15: Excerpt of instances of a faulty UML state machine design, where more then one transition leaves the initial state.

represented by the depicted RDF statements. The mentioned SPARQL query allows to detect the invalid two outgoing transitions from the initial state.

The UML standard provides additional expert knowledge, which can be used to verify a UML state machine:

Definition 5.4 *Outgoing transitions from a fork must not have a guard.*

Figure 5.16 shows a faulty design from the UML editor and its formal representation using the

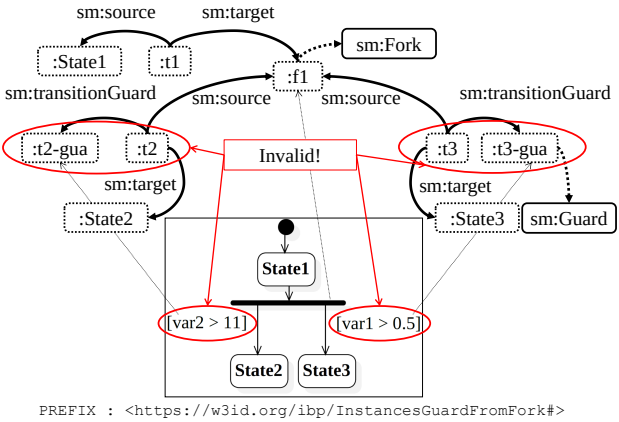


Figure 5.16: Excerpt of instances of a faulty UML state machine, where guards are associated to transitions leaving a fork.

`StateMachineOntology` (see Section 4.4.6). Again, the knowledge to identify this invalid design, where guards are associated to the two transitions leaving the fork can be implemented as a SPARQL query, which is listed in Code C.7.
The formal verification is not restricted to control logic, such as UML state machines, but can additionally be applied to other control logic types, such as transfer function elements. For instance, in expressions of transfer function elements often a variable and a parameter are compared. A common fault is that the parameter and variable represent a different quantity, e.g. mass fraction and volume fraction.

Definition 5.5 *If a parameter and a variable or a variable and a variable are compared in an expression, then the quantities of both needs to match.*

The importance of this knowledge in the described use case is illustrated in Figure 5.17 by presenting an excerpt of instances, which formally represent a transfer function element. It compares the value of the concentration measured in tank B3 `QI302` with a parameter `Conc_B3_high` [KSB01]. Here, the quantity of the two elements in the expression do not match and contradict the rule-like knowledge (Definition 5.5). A SPARQL query encoding this knowledge is given in Code C.9.

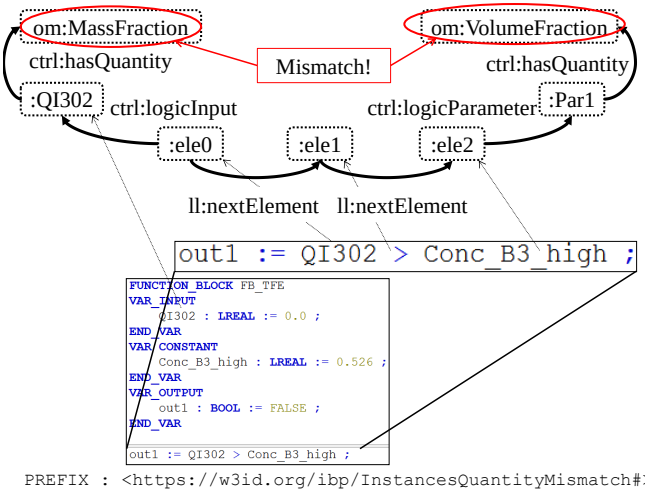


Figure 5.17: Excerpt of instances in a transfer function element, where a variable and a parameter with different quantities are compared.

Formal verification of control logic across types

A common downside of existing formal verification methods is that typically only one type of control logic can be verified. The method and approach presented here supports the verification across control logic types by the terminological knowledge defined in the modular ontology as well as specifying expert knowledge as demonstrated in the following scenario. A value with a basic data type is assigned in an action of a state machine to an output of this state machine. This output is connected to zero or more inputs of downstream control actors. If an input connected to the output of the action is compared to a parameter in an expression, then the parameter basic data type needs to match the basic data type of the assignment in the state machine. The scenario is depicted in Figure 5.18 including an excerpt of instances formally representing the described scenario. Two control actors are considered here: (1) the control actor ensuring the save operation of the pump `fB_PUM901_Safety` and (2) the state machine `fB_StateMachine`, which is also presented in Figure 5.11. Within the specification of the UML state machine in state `s10` the value `True` is assigned to the output `bPUM901` with a literal data type `xsd:string`. Downstream of the state machine a control actor `fB_PUM901_Safety` receives this output as an input and compares it to a parameter. The basic data type of this parameter (`:fB_PUM901_Safety-ExpEle-2-Par`) is defined as `BDO:boolean`. The knowledge to detect this defect can be implemented as a SPARQL query and is used to

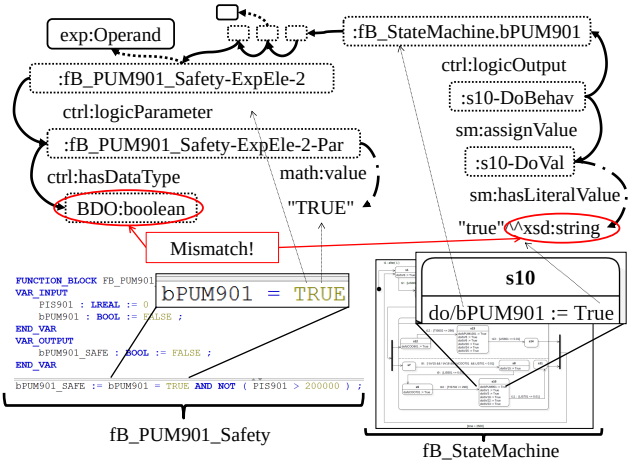


Figure 5.18: Verification across control logic types, where the basic data type of a value assigned in action to an output is compared to a parameter in the control logic specification of a down stream control actor.

check whether this occurs in the stored knowledge (see Code C.8). In the regarded scenario the verification includes knowledge on UML state machines as well as transfer function elements, thus verifies across control logic types.

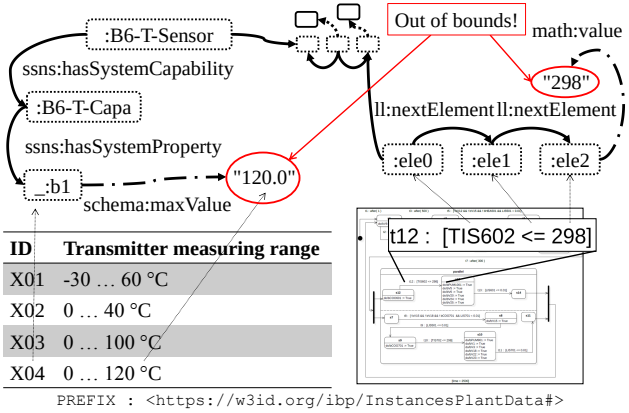
Verification of control logic and plant data

The verification of control logic and plant data is of particular interest to scholars [Vya13, VHFST15]. The strong ties between the control and the actual physical plant lead to tight interdependencies. In the following scenario the benefit from verifying control logic and plant data is demonstrated.

A real sensor[10] is chosen to realise the measurement of the temperature in tank B6 of the batch plant. From the data sheet of the sensor the measurement range is extracted and added to the knowledge base. The respective triples are presented in Code C.10.

Now this knowledge can be used to verify the control logic design as illustrated in Figure 5.19. Within the state machine the respective input `TIS602` of the sensor is compared to a parameter value of ”298”. When comparing this value to the maximum measurement range, i.e. ”120” it is possible to detect this defect. The SPARQL query listed in Code C.11 specifies knowledge, which allows to detect faulty designs, where a mismatch occurs between the upper bound of the measurement range of a sensor and a parameter compared to it.

[10] ABB, Model: SensyTemp TSP121 [ABB18]



ID	Transmitter measuring range
X01	-30 … 60 °C
X02	0 … 40 °C
X03	0 … 100 °C
X04	0 … 120 °C

Figure 5.19: Based on the formal domain description it is possible to verify whether a parameter compared to a sensor measurement is within the bounds of its measurement range.

Bidirectional flow of information and incremental verification

In the following a scenario is studied, which uses the abilities of the developed method and approach for bidirectional flow of information and incremental verification. The setup of the scenario in the designed KBS is depicted in Figure 5.20 for reference. For the considered batch plant a control solution is designed and virtually tested through simulation in Dymola (1). The design is extracted and stored in the knowledge base (2) and the verification mechanisms are applied. Then the design is exported to the PLCopen XML format (3) and tested on a real hardware PLC. The commissioning staff adds an additional piece of control logic to protect pumps from running if valves are still closed. This feature is added by introducing two control actors, which compare the signals of pressure signals of sensors `PIS901` and `PIS1001` against a threshold (200000 Pa) and evaluate the boolean output of the state machine indicating if a pump should run (`bPUM901`) (see Code C.12). The changed PLC project is exported again (4) and verified (5) in the KBS (next increment). Finally, the new version is exported to the Modelica format as listed in Code C.14 to allow simulation-based methods in the engineering, e.g. virtual commissioning. For reference, the source code of the novel control actor is listed in Code C.12, its RDF representation is listed in Code C.13 and the generated Modelica source code after verification is given in Code C.14.

The scenario illustrates the ability of the designed KBS to support incremental verification. It should be noted that the model-based transformation between different formats is reported by other model-driven engineering approaches, e.g. Alvarez *et al.* [ASB+18] or Lüder *et*

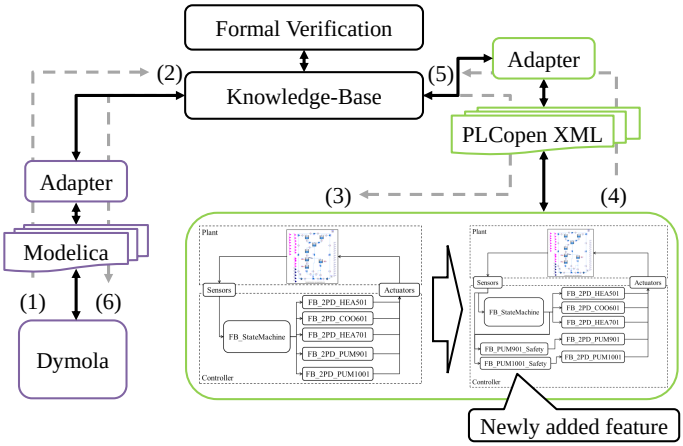


Figure 5.20: 'Schema of the scenario to evaluate the support of bidirectional information exchange and incremental verification. The designed control logic (1) is exported to the knowledge-base, verified (2) and then exported to the target PLC (3). New features can be added and are exported (4) to the knowledge base and (next increment) verified (5) until again Modelica code (6) is generated' [SWO19]

al. [LEHM11], however, in this approach the formal verification can be applied each time an iteration happens and, thus, the integrity and the consistency of the generated code is ensured.

5.3 Summary

Within this chapter the model defined in Chapter 4 is validated in its ability to support novel knowledge-based methods for the engineering and operation of control logic in automation systems by serving their knowledge needs. The models are successfully utilised to support the engineering and operation of control logic in two domains of automation: building automation and industrial automation. The presented use cases are part of two publications [SPS17, SWO19].

In the use case presented in Section 5.1 the semantic model developed in this thesis is used to explicitly and formally represent control logic in BAS. The models provide the basis for a novel, knowledge-based method [SPS17]. This method uses the represented knowledge to automatically configure and deploy archetype rules to verify from monitoring data if the control logic in a BAS implementation is implemented as-designed. The verification of discrete control

logic formulated as state graph and of schedules is validated in a simulation-based scenario related to the automation of an AHU.
In addition, the developed models are validated in a second use case from the industrial automation domain, which is reported in Section 5.2. The use case addresses some problems identified in the automation domain including:

- the heterogeneity of formats and tools prevalent in engineering tool chains;
- the missing ability of current automated formal verification mechanisms to verify different types of control logic;
- the missing ability of current automated formal verification methods to verify control logic and plant data simultaneously;
- the need for incremental verification of control logic designs over the life cycle and the bidirectional flow of information from the final PLC code back to the model.

In response to the defined problems a KBS is designed, which supports the engineering of control logic in automation systems through novel knowledge-based methods. The core component of the KBS is a modular domain ontology, which uses the models developed in Chapter 4. Knowledge on adjacent domains is described by reusing existing OWL ontologies. In a scenario-based evaluation approach the developed system is used to enable the formal verification of control logic across types and including plant data as well as investigating the bidirectional exchange from and to the target formats while incrementally verifying the model. The scenarios are motivated and tested for an automation solution designed to automate a batch plant available from literature [KSB01].
In essence, the contributions made in this chapter to reach the defined objectives and fulfil the defined requirements are detailed in the following. An overview is given in Figure 5.21.

- Req. 3.1.1 *Semantic integration of heterogeneous formats and tools:* In particular, the use case presented in Section 5.2 shows the successful integration of heterogeneous formats, such as PLC open XML [PLC09] and Modelica [Mod17a] source code. The reported approach shows the integration of tools for the specification, hybrid simulation and actual deployment of control logic on a PLC;
- Req. 3.1.2 *Manufacturer independent:* The presented solutions and methods are implemented and used in absence of specific software ecosystems of manufacturers. For instance, open standardised formats, such as XMI [Obj15b], Modelica [Mod17a] and PLCopen XML [PLC09] are used for demonstration in the use cases. The prototypical implementations (see Chapter 5) of applications developed in this thesis are carried out using Python programming language and for implementing the semantic models (Chapter 4) the open and free OWL knowledge modelling language [W3C12] is used;

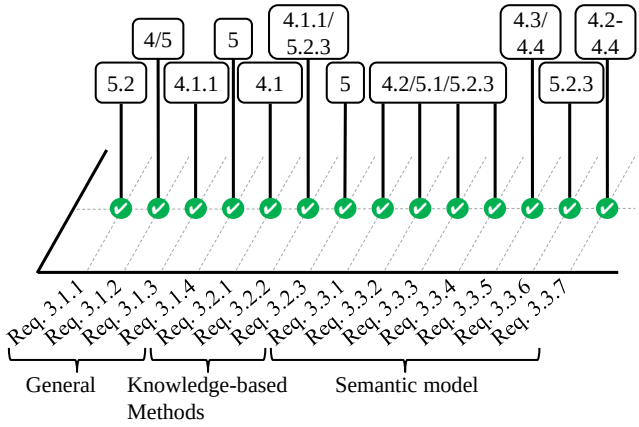


Figure 5.21: Overview on the alignment of requirements and contributions provided in this thesis.

- Req. 3.1.3 *Technology independent model specification:* To ensure future reusability of the developed models an object-oriented modelling paradigm is followed as documented in Section 4.1.1. Moreover, the models are specified using the UML [Obj15a] and, hence, are specified independent from the actual implementation technology;
- Req. 3.1.4 *Machine-interpretable implementation:* The semantic models are implemented using OWL. OWL is a formal knowledge representation language with a formal syntax [MPSP12] and semantics [MPSCG12]. Based on OWL KBS can be implemented, which allow machines to interpret represented domain knowledge. The successful implementation of knowledge-based methods is described in Section 5.1 and Section 5.2;
- Req. 3.2.1 *Use of Ontology:* The use of ontology for knowledge representation is found to be appropriate through out the use cases and from the cited references (see Section 2.5). In particular, with respect to the modelling method, ontology is used as specified in Section 4.1 and Chapter 4;
- Req. 3.2.2 *Use of OWL:* The semantic models developed in this thesis are implemented using OWL. Some benefits are arising from this. The ontologies seamlessly integrate with adjacent domain models (Section 5.2.3 and Figure 5.12). An example of the already successful reuse of `CTRLont` ontology by other researchers is presented in Balaji *et al.* [BBF+18]. The respective ontologies are reused (see also Req. 3.3.3, Req. 3.3.7 and Req. 3.3.6) as successfully demonstrated in Section 5.2;

- Req. 3.2.3 *Use of appropriate profile:* During the course of this investigation the reasoning capabilities of off-the-shelf OWL reasoners are found to provide sufficient means for reasoning during early stages of the ontology engineering phase [GHM+14, SPG+07]. Later, a triple store, GraphDB[11], is utilised, which supports both, high performant OWL 2 RL [MCGH+12] reasoning as well as execution of SPARQL queries and updates. The capabilities of this combination are found to be sufficient with respect to the inference needs in the two use cases and the computational performance. The reasoning tasks have been completed in less then a second;
- Req. 3.3.1 *High-level abstraction of automatic control domain:* The `CTRLont` as reported in Section 4.3 aims at providing sufficient means for the high level abstraction of the automatic control domain. The model constitutes the core of the domain models utilised in the use cases as described in Section 5.1 and Figure 5.12;
- Req. 3.3.2 *Hierarchy and logical topology:* The Control Ontology model and ontology (`CTRLont`, Section 4.3) allow to express hierarchy and logical topology for control actors in automation systems;
- Req. 3.3.3 *Relation to adjacent physical domains:* The formal explicit models of control logic are integrated with adjacent domains of information through linking the `CTRLont` with adjacent domain models (Section 4.3, Section 5.2.3 and Figure 5.12);
- Req. 3.3.4 *Semantic of inputs, outputs and parameters:* The `CTRLont` allows to annotate inputs, outputs and parameters of control actors with their unit, quantity, medium, basic data type and a semantic data type (Section 4.3, Figure 4.3;
- Req. 3.3.5 *Explicit formal models of control logic:* A major contribution of this thesis is the formal and explicit modelling of control logic in automation systems. In particular, in Section 4.4 semantic models and their ontology-based implementation for generic control logic frequently appearing in automation systems and for discrete control logic, which can be described using a state based formalism, are presented. The benefit of the formal explicit modelling is highlighted in the use cases presented in Chapter 5;
- Req. 3.3.6 *Ontology reuse:* A number of ontologies have been successfully reused. In particular, for relating the control logic specification and the automatic control domain to adjacent mechanical and electrical domains (see Section 5.2.3 and Figure 5.12);
- Req. 3.3.7 *Modular and extensible:* The defined model is abstracted into two layers. The Control Ontology (Section 4.3) describes the automatic control domain on an abstract level. It may be extended separately with explicit formal specifications of control logic, e.g. the ones presented in Section 4.4.

[11] `https://ontotext.com/products/graphdb/`

6 Conclusion and Outlook

6.1 Conclusion

In many technical processes automation systems are a key technology for the successful deployment of these processes. In particular, automation technology comes into play when processes are repetitive, or dangerous to humans and often machines can perform these tasks with higher precision, faster, more efficient and at lower cost [SSKD11]. The successful application of automation technology is associated with solving some of the existing global challenges, such as the sustainable supply of primary energy [VDI15]. Hence, an ultimate goal is to lower the barriers for the engineering and operation of automation technologies to pursue further their widespread adoption.
The actual control logic operating in an automation system significantly effects the overall performance of the system. A number of problems related to the engineering and operation of control logic in automation systems are identified in Chapter 1. These problems are related to the increasing complexity arising from the size of the systems, their cyber and physical nature and the paradigm change in automation through the introduction of advanced ICT. Additionally, problems arise from the heterogeneity of tools and data formats utilised by various stakeholders during the engineering and operation of control logic in automation systems. This heterogeneity conceals the incorporated knowledge for further reuse and exploitation.
In response to these problems, objectives for this thesis are defined as stated in Section 1.3 and are repeated here for reference:

- **Objective 1:** *Development of knowledge-based methods to support and automate the engineering and operation of control logic in automation systems.*
- **Objective 2:** *Development of a semantic model of control logic in automation systems.*

Based on the needs arising from the defined research objectives, a brief description of the automatic control and control logic domains in automation is provided as well as the foundations of KR are presented in the first part of Chapter 2. Next, the current state of the art with respect to knowledge-based methods for the engineering and operation of control logic in BAS and IAS and existing data formats and ontologies for the description of the respective domains are analysed in the second part of Chapter 2. The analysis reveals a *gap* in the current state of the art: The absence of a model to explicitly and formally describe knowledge on the automatic

control and control logic domains in automation systems and integrate this formal descriptions with adjacent domains, such as electrical and mechanical engineering. This gap prohibits the development of novel knowledge-based methods to support and automate the engineering and operation of control logic.
To overcome the identified shortcomings in existing work and close the identified gap, Chapter 3 presents requirements in this regard. The requirements are differentiated into general requirements, requirements related to Objective 1 for the development of knowledge-based methods and requirements related to Objective 2 for the definition of a domain model.
The remaining chapters, Chapter 4 and Chapter 5, incorporate the *contributions* of this thesis, which are summarised in the following. The sequence of this presentation is oriented according to the defined objectives, hence, the contributions made in Chapter 5 are presented first.
Objective 1 is reached in Chapter 5. Two novel knowledge-based methods are developed, which support the engineering and operation of control logic in automation systems.
Section 5.1 presents a novel method, which enhances the operation of control logic in BAS. The method allows the automated, rule-based verification of designed control logic with monitoring data obtained from a building management system. The method is prototypically implemented using the free python programming language and has been tested in a use case related to the control of an AHU.
In Section 5.2 a novel knowledge-based method is developed to support the engineering of control logic in IAS. The method allows to automate formal verification of the designed control logic for different types of control logic and plant data, incrementally verify changes and updates to the control logic design and support the bidirectional flow of information from the model to the target format and vice versa. The method is tested in a use case related to a real, lab-scale plant from process automation.
With respect to Objective 1 this thesis makes the following scientific and technical contributions by addressing identified problems as well as it produces the following outcomes for industry:

- *Complexity from size:* The sheer size of automation systems as highlighted in Section 1.2.1 creates the need for automating associated processes during the engineering and operation. So far, control logic is not the main focus area for such methods (see Section 2.5). The methods presented in Chapter 5 allow to automate verification tasks related to the engineering and operation of control logic in automation systems. The method presented in Section 5.1 shows how formalising design knowledge of control logic can be utilised to automatically instantiate rules to verify, whether the designed control logic is implemented in a BAS. The correct implementation has significant impact on the overall performance of a building, e.g. with respect to its energy demand;
- *Complexity from CPS*: Knowledge-based methods are identified to offer the possibility to enhance the engineering of CPS, e.g. [NSMŠ15, Mos16]. A contribution of this thesis is the inclusion of domain knowledge on control logic in this effort. An implic-

ation of this is, for instance, as described in Section 5.2, the possibility for concurrent verification of control logic and plant data. Hence, the integration of cyber and physical domains is successfully performed;

- *Complexity from paradigm change through the introduction of ICT:* The described knowledge-based methods show how automated applications can further increase the quality of control logic in automation systems by leveraging on knowledge stored in a common semantic integration layer;
- *Heterogeneity of tools, formats and stakeholders:* This thesis contributes to successfully addressing the prevalent heterogeneity of tools, formats and stakeholders involved in the engineering process of automation systems. The study conducted in Section 5.2 highlights the superior ability of the chosen ontology-based knowledge representation approach and SWT for implementation by integrating heterogeneous formats and tools used by different stakeholders in the engineering process, such as the XML-based XMI format used for specification, Modelica source code used for hybrid simulation and the PLCopen XML format used for the actual deployment of control logic on a PLC;
- *Openness of tools and implementation:* The described knowledge-based methods are implemented using mostly open-source software (Python, Protégé [Mus15], etc.) and are based on open standards (OWL, SPARQL, Modelica, PLCOpen XML, etc.). In addition they are published and documented for reference and reuse by practitioners from industry as well as scholars from academia [SPS17, SWO19].

Objective 2 is reached in Chapter 4 by presenting a novel, layered semantic model. The model allows to formally describe knowledge acquired from the automatic control and control logic domains and link this knowledge to adjacent domains, such as mechanical or electrical engineering. An object-oriented modelling methodology is adopted and the models are specified using the UML modelling language. This allows the technology independent specification and is particularly interesting as, for instance, BAS operate for decades and implementation technologies most likely change. The model is implemented using OWL and forms the basis for the mentioned, novel knowledge-based methods described in Chapter 5. In the layered model, extensions are defined to explicitly describe knowledge of the control logic domain. The presented explicit models of control logic reflect a non-exhaustive set of control logic types, which is not restricted to a certain manufacturer of automation systems.
With respect to Objective 2 this thesis makes the following scientific and technical contributions by addressing identified problems as well as it produces the following outcomes relevant for industry:

- *Fundamental deficit:* This thesis contributes to science by solving the fundamental deficit of the missing availability of an explicit formal specification of control logic in automation systems and their linkage to adjacent domains. The Control Ontology

defined in Section 4.3 allows to link explicit formal models of control logic to physical domain description. For the adjacent domains respective domain models need to be developed or can be reused as shown in Chapter 5. As indicated in the introduction of this thesis, automation systems have a significant impact on global challenges, such as energy, and, hence, the outcome of this thesis can help to pursue this path in future by enabling the integration of formal knowledge on the control logic domain in methods and applications related to further improve the quality of automation systems and the performance of automated processes;

- *Ontology-based KR and SWT for implementation:* The presented model and its validation contributes to science by providing additional evidence for the superior capabilities of ontology-based KR using SWT for their implementation. A particular strength is the possibility to integrate heterogeneous knowledge sources. Moreover, the underlying DL support the automated analysis of the represented knowledge and deduce novel, implicitly stated insights;
- *Reusability and openness:* To stipulate reuse and extension of this research a technical contribution of this thesis is the openness established of the developed models and documentation. All models are specified technology independent using the UML. All implemented ontologies are available open-source from an online repository, which is accessible through their respective persistent ontology URI. The measures are found to be successful as, for instance, `CTRLont` is considered for reuse in the context of formally modelling monitoring data in building management systems as presented by Balaji *et al.* [BBF+18];
- *Manufacturer independence:* The developed formal models of control logic are chosen to be manufacturer independent. Hence, practitioners from industry may reuse and adapt to their own software eco-systems without the need to buy additional soft- or hardware.

In addition to the highlighted contributions open questions for future research remain and are discussed in the next section.

6.2 Outlook

This thesis aims at contributing towards filling the identified gaps. However, open questions for future research can be defined.
Ontologies are by definition 'a formal, explicit specification of a shared conceptualization.' [SBF98], where the term *shared* is an important part of the definition implying that a body of experts agrees on a conceptualisation. Hence, the presented models in this thesis have to be considered as a starting point towards formalising the domain of control logic. They need to be

discussed in a broad community and revised until reaching an appropriate level of maturity. In particular, the connection to national and international standardisation efforts needs to be made [ETS15, PLC09, BWL+07, IEC08]. As the modelling effort is a starting point it should be evaluated in future how existing ontologies can be extended or revised to support the explicit description of control logic. Interesting are the efforts related to SAREF [DdHR15], which is an open standard in the smart appliances domain [ETS15], or AutomationMLOWL [KGG18], which is an open standard in the IAS domain.

A field for further research relates to the specification of engineering knowledge related to mathematical calculations and generic mathematical knowledge. The in Section 4.4.1 developed `ExpressionOntology` provides the means to formally describe expressions. Beyond this focussed view work exists towards formally describing generic mathematical knowledge on the web, such as proofs and theorems [WR12, Ope16, Lan13]. For instance, it would be interesting to develop knowledge-based approaches to identify if the transition between two intervals is continuous. Other examples include the automated calculation of unit conversions [RWT11].

The field of control logic offers an enormous breadth of methods for the control of physical systems (e.g. see [Hon97, Sal05, Abe10]). The approach presented in this thesis requires the explicit modelling of each control logic type, where there certainly exists a trade-off between the modelling effort required and the complexity of the respective control logic types. Potential candidates to be added as explicit formal descriptions might be the Petri Net formalism [Fre02, Lun09, LS17] or fuzzy control logic, where a good starting point is the work by De Maio *et al.* [DMFF+12]. In addition, an interesting research item is to investigate if the conversion between similar control logic types can be automated. An initial contribution in this regard is described in Schneider *et al.* [SPT19].

A promising research path to follow emerges at the intersection of formal modelling approaches of static knowledge and data driven approaches, such as machine learning for quantitative analysis. For example, it would be interesting to correlate performance characteristics of plants with their control parametrisation to identify well-performing control logic. Interesting work on the topic of learning knowledge is published by Wicaksono *et al.* [WJRO14] and Wicaksono [Wic16].

In terms of tackling open challenges in the engineering of control logic in automation systems detecting code clones [VHFST15] is an interesting research area. Again quantitative approaches might be interesting in this regard [TRPE17] and a connection to formal descriptions to allow reasoning and semantic search could be a promising research area.

The validation performed in this thesis is restricted to simulation-based evaluation of plants available from literature in absence of the access to a real plant and automation system. Future research and evaluation activities should aim at obtaining data and applying the respective models to real world use cases for further validation and verification.

Bibliography

[3ds17] 3ds. http://www.3ds.com, 2017. Dassault Systemes AB, Lund, Sweden, Last accessed: 22 October 2018.

[ABB18] ABB. Temperature sensors SensyTemp TSP111, TSP121, TSP131. http://bit.ly/2Eln615, 2018. Last accessed: 22 October 2018.

[Abe10] D. Abel. *Regelungstechnik*. Verlagsgruppe Mainz GmbH, Aachen, Germany, 34th edition, 2010.

[AGE17] AGEB e.V. Auswertungstabellen zur Energiebilanz Deutschland, September 2017. Arbeitsgemeinschaft Energiebilanzen (AGEB), Berlin, Germany.

[ÅH01] K. J. Åström and T. Hägglund. The future of PID control. *Control Engineering Practice*, 9(11):1163–1175, 2001.

[ALGM13] L. Abele, C. Legat, S. Grimm, and A. W. Müller. Ontology-based validation of plant models. In *Proceedings of the 11th International Conference on Industrial Informatics (INDIN)*, pages 236–241, Bochum, Germany, July 29-31 2013. IEEE.

[ASB+18] M. L. Alvarez, I. Sarachaga, A. Burgos, E. Estévez, and M. Marcos. A Methodological Approach to Model-Driven Design and Development of Automation Systems. *IEEE Transactions on Automation Science and Engineering*, 15(1):67–79, 2018.

[BBČ+10] J. Bārzdiņš, G. Bārzdiņš, K. Čerāns, R. Liepiņš, and A. Sproģis. OWLGrEd: a UML Style Graphical Editor for OWL. In *Proceedings of the 1st Workshop on Ontology Repositories and Editors for the Semantic Web (ORES)*, pages 21–25, Hersonissos, Greece, May 31 2010. CEUR-WS.org.

[BBF+16] B. Balaji, A. Bhattacharya, G. Fierro, J. Gao, J. Gluck, D. Hong, A. Johansen, J. Koh, J. Ploennigs, Y. Agarwal, M. Berges, D. Culler, R. Gupta, M. B. Kjærgaard, M. Srivastava, and K. Whitehouse. Brick: Towards a unified metadata schema for buildings. In *Proceedings of the 3rd ACM International Conference on Systems for Energy-Efficient Built Environments (BuildSys)*, pages 41–50, Palo Alto, USA, November 16-17 2016.

[BBF+18] B. Balaji, A. Bhattacharya, G. Fierro, J. Gao, J. Gluck, D. Hong, A. Johansen, J. Koh, J. Ploennigs, Y. Agarwal, M. Berges, D. Culler, R. K. Gupta, M. B. Kjærgaard, M. Srivastava, and K. Whitehouse. Brick : Metadata schema for portable smart building applications. *Applied Energy*, 226:1273–1292, 2018.

[BBLPC14] D. Beckett, T. Berners-Lee, E. Prud'hommeaux, and G. Carothers. RDF 1.1 Turtle Terse RDF Triple Language. Permanent URI https://www.w3.org/TR/turtle/, 2014. Recommendation, World Wide Web Consortium (W3C), Cambridge, USA.

[BC08] D. Bonino and F. Corno. DogOnt - Ontology Modeling for Intelligent Domotic Environments. In *Proceedings of the 7th International Semantic Web Conference (ISWC)*, pages 790–803, Karlsruhe, Germany, October 26-30 2008.

[BCM+03] F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, and P. F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, New York, USA, 2003.

[Bec18] Beckhoff Automation GmbH. TwinCAT 3.1. `http://www.beckhoff.de`, 2018. Verl, Germany, Last accessed: 22 October 2018.

[Ber18] Beremiz. PLCopen Editor. `http://www.beremiz.org`, 2018. Last accessed: 22 October 2018.

[BG14] D. Brickley and R. V. Guha. RDF Schema 1.1. Permanent URL: `https://www.w3.org/TR/rdf-schema/`, 2014. Recommendation, World Wide Web Consortium (W3C), Cambridge, USA.

[BGL17] S. Biffl, D. Gerhard, and A. Lüder. *Multi-Disciplinary Engineering for Cyber-Physical Production Systems*. Springer International Publishing, Cham, Switzerland, 2017.

[BGT17] B. Butzin, F. Golatowski, and D. Timmermann. A survey on information modeling and ontologies in building automation. In *43rd Annual Conference of the IEEE Industrial Electronics Society (IECON)*, pages 8615–8621, Beijing, China, October 29 - November 1 2017.

[BHS07] I. Boersch, J. Heinsohn, and R. Socher. *Wissensverarbeitung: Eine Einführung in die Künstliche Intelligenz für Informatiker und Ingenieure*. Elsevier Spektrum Akademischer Verlag, Heidelberg, Germany, 2nd edition, 2007.

[BKI03] C. Beierle and G. Kern-Isberner. *Methoden wissensbasierter Systeme: Grundlagen, Algorithmen, Anwendungen*. Springer, Wiesbaden, Germany, 2003.

[BL04] R. Brachman and H. Levesque. *Knowledge Representation and Reasoning*. Elsevier, Amsterdam, The Netherlands, 2004.

[BLHL01] T. Berners-Lee, J. Hendler, and O. Lassila. The Semantic Web. *Scientific American*, 285(5):28–37, 2001.

[BM04] P. V. Biron and A. Malhotra. XML Schema Part 2: Datatypes Second Edition. Permanent URL: `https://www.w3.org/TR/xmlschema-2/#built-in-datatypes`, 2004. Recommendation, World Wide Web Consortium (W3C), Cambridge, USA.

[BPSM+06] T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler, F. Yergeau, and J. Cowan. Extensible Markup Language (XML) 1.1 (Second Edition). Permanent URL: `https://www.w3.org/TR/xml11/`, 2006. Recommendation, World Wide Web Consortium (W3C), Cambridge, USA.

[BRCR17] G. Benndorf, N. Rehault, M. Clairembault, and T. Rist. Describing HVAC controls in IFC - method and application. *Energy Procedia*, 122:319–324, 2017.

[BS16a] S. Biffl and M. Sabou. Introduction. In S. Biffl and M. Sabou, editors, *Semantic Web Technologies for Intelligent Engineering Applications*, pages 1–13. Springer International Publishing, Cham, Switzerland, 2016.

[BS16b] S. Biffl and M. Sabou, editors. *Semantic Web Technologies for Intelligent Engineering Applications*. Springer International Publishing, Cham, Switzerland, 2016.

[BWL+07] R. Batres, M. West, D. Leal, D. Price, K. Masaki, Y. Shimada, T. Fuchino, and Y. Naka. An upper ontology based on ISO 15926. *Computers & Chemical Engineering*, 31(5):519–534, 2007.

[Che15] Y. Chen. *Building Control Knowledge Information Modeling & Control Self-Configuration*. PhD thesis, Pennsylvania State University, State College, USA, 2015.

[CKAK15] V. Charpenay, S. Käbisch, D. Anicic, and H. Kosch. An ontology design pattern for IoT device tagging systems. In *Proceedings of the 5th International Conference on the Internet of Things (IOT)*, pages 138–145, Seoul, South Korea, October 26-28 2015. IEEE.

[COC+13] E. Curry, J. O'Donnell, E. Corry, S. Hasan, M. Keane, and S. O'Riain. Linking building data in the cloud: Integrating cross-domain building data using linked data. *Advanced Engineering Informatics*, 27(2):206–219, 2013.

[CTM16] Y. Chen, S. J. Treado, and J. I. Messner. Building HVAC control knowledge data schema – towards a unified representation of control system knowledge. *Automation in Construction*, 72, Part 2:174–186, 2016.

[CWL14] R. Cyganiak, D. Wood, and M. Lanthaler. Resource Description Framework (RDF): Concepts and Abstract Syntax. Permanent URL: `http://www.w3.org/TR/rdf11-concepts/`, 2014. Recommendation, World Wide Web Consortium (W3C), Cambridge, USA.

[DCTTDK11] K. Dentler, R. Cornet, A. Ten Teije, and N. De Keizer. Comparison of reasoners for large ontologies in the OWL 2 EL profile. *Semantic Web*, 2(2):71–87, 2011.

[DCVK16] P. Domingues, P. Carreira, R. Vieira, and W. Kastner. Building automation systems: Concepts and technology review. *Computer Standards & Interfaces*, 45:1–12, 2016.

[DdHR15] L. Daniele, F. den Hartog, and J. Roes. Created in Close Interaction with the Industry: The Smart Appliances REFerence (SAREF) Ontology. In *Proceedings of the 7th International Workshop Formal Ontologies Meet Industries (FOMI)*, pages 100–112, Berlin, Germany, August 5 2015. CEUR-WS.org.

[DFH11] J. Domingue, D. Fensel, and J. A. Hendler. *Handbook of semantic web technologies*. Springer, Berlin, Germany, 2011.

[DHR16a] H. Dibowski, O. Holub, and J. Rojíček. Knowledge-based Fault Propagation in Building Automation Systems. In *Proceedings of the International Conference on Systems Informatics, Modelling and Simulation (SIMS)*, pages 124–132, Riga, Latvia, June 1-3 2016. IEEE.

[DHR16b] H. Dibowski, O. Holub, and J. Rojíček. Ontology-based automatic setup of virtual sensors in building automation systems. In *Proceedings of the International Congress on Ultra Modern Telecommunications and Control Systems and Workshops (ICUMT)*, pages 375–381, Lisbon, Portugal, October 18-20 2016. IEEE.

[Dib13] H. Dibowski. *Semantischer Gerätebeschreibungsansatz für einen automatisierten Entwurf von Raumautomationssystemen*. PhD thesis, TU Dresden, Dresden, Germany, 2013.

[DMFF+12] C. De Maio, G. Fenza, D. Furno, V. Loia, and S. Senatore. OWL-FC: an upper ontology for semantic modeling of Fuzzy Control. *Soft Computing*, 16(7):1153–1164, 2012.

[Dol04] P. Dolog. Model-Driven Navigation Design for Semantic Web Applications with the UML-Guide. In *Proceedings of the 4th International Conference on Web Engineering (ICWE)*, pages 75–86, Munich, Germany, July 28-30 2004.

[DPK10] H. Dibowski, J. Ploennigs, and K. Kabitzsch. Automated Design of Building Automation Systems. *IEEE Transactions on Industrial Electronics*, 57(11):3606–3613, 2010.

[DR15] B. Dathan and S. Ramnath. *Object-Oriented Analysis, Design and Implementation*. Springer International Publishing, Cham, Switzerland, 2015.

[DSC12] S. Das, S. Sundara, and R. Cyganiak. R2RML: RDB to RDF Mapping Language. Permanent URL: `https://www.w3.org/TR/r2rml/`, 2012. Recommendation, World Wide Web Consortium (W3C), Cambridge, USA.

[DSS93] R. Davis, H. Shrobe, and P. Szolovits. What Is a Knowledge Representation? *AI magazine*, 14(1):1–17, 1993.

[Dub11] A. Dubey. Evaluating software engineering methods in the context of automation applications. In *9th International Conference on Industrial Informatics (INDIN)*, pages 585–590, Caparica, Portugal, July 26-29 2011. IEEE.

[DuC15] B. DuCharme. *Learning SPARQL: querying and updating with SPARQL 1.1*. O'Reilly, Sebastopol, USA, 2015.

[DVC11] E. Della Valle and S. Ceri. Querying the semantic web: SPARQL. In J. Domingue, D. Fensel, and J. A. Hendler, editors, *Handbook of Semantic Web Technologies*, pages 299–363. Springer, Berlin, Germany, 2011.

[DVHR16] H. Dibowski, J. Vass, O. Holub, and J. Rojíček. Automatic setup of fault detection algorithms in building and home automation. In *Proceedings of the International Conference on Emerging Technologies and Factory Automation (ETFA)*, pages 1–6, Berlin, Germany, September 6-9 2016. IEEE.

[DW14] P. De Wilde. The gap between predicted and measured energy performance of buildings: A framework for investigation. *Automation in Construction*, 41:40–49, 2014.

[EA07] P. C. Evans and M. Annunziata. Industrial internet: Pushing the boundaries of minds and machines. In *General Electric Reports*, 2007.

[EJL+03] J. Eker, J. W. Janneck, E. A. Lee, J. Liu, X. Liu, J. Ludvig, S. Neuendorffer, S. Sachs, and Y. Xiong. Taming heterogeneity - the Ptolemy approach. *Proceedings of the IEEE*, 91(1):127–144, 2003.

[EN 17] EN 15232. Energy performance of buildings – Impact of Building Automation, Controls and Building Management, 2017. European Committee for Standardization (CEN), Brussels, Belgium.

[ESS+17] F. J. Ekaputra, M. Sabou, E. Serral, E. Kiesling, and S. Biffl. Ontology-based data integration in multi-disciplinary engineering environments: A review. *Open Journal of Information Systems*, 4(1):1–26, 2017.

[ETS15] ETSI TS 103 264. SmartM2M Smart; Appliances; Reference Ontology and oneM2M Mapping, 2015. V1.1.1, European Telecommunications Standards Institute, Sophia Antipolis, France.

[ETS17a] ETSI TS 103 264. SmartM2M Smart; Appliances; Reference Ontology and oneM2M Mapping, 2017. V2.1.1, European Telecommunications Standards Institute, Sophia Antipolis, France.

[ETS17b] ETSI TS 103 410-1. SmartM2M; Smart Appliances Extension to SAREF; Part 1: Energy Domain, 2017. European Telecommunications Standards Institute, Sophia Antipolis, France.

[ETSL11] C. Eastman, P. Teicholz, R. Sacks, and K. Liston. *BIM handbook: A guide to building information modeling for owners, managers, designers, engineers and contractors*. John Wiley & Sons, Hoboken, USA, 2nd edition, 2011.

[Eur08] European Commission. *Energy Efficiency in Manufacturing – The Role of ICT*. Office for Official Publications of the European Communities, Luxembourg, Luxembourg, 2008.

[FD04] M. Fedai and R. Drath. CAEX - ein neutrales Datenaustauschformat für Anlagendaten. *Automatisierungs Technische Praxis*, 46(2):52–56, 2004.

[FGP+14] R. Falco, A. Gangemi, S. Peroni, D. Shotton, and F. Vitali. Modelling OWL Ontologies with Graffoo. In *Proceedings of European Semantic Web Conference (ESWC) Satellite Events*, pages 320–325, Anissaras, Greece, May 25-29 2014.

[FLGPJ97] M. Fernández-López, A. Gómez-Pérez, and N. Juristo. Methontology: from ontological art towards ontological engineering. In *Proceedings of the American Association for Artificial Intelligence (AAAI) Spring Symposium Series*, pages 33–40, Stanford, USA, March 24-26 1997.

[FR14] R. Fielding and J. Reschke. Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing. Permanent URL: `https://www.rfc-editor.org/rfc/rfc7230.txt`, 2014. Request for Comments, Internet Engineering Task Force (IETF), Fremont, USA.

[Fre02] G. Frey. *Design and formal Analysis of Petri net based Logic Control Algorithms*. PhD thesis, TU Kaiserslautern, Kaiserslautern, Germany, 2002.

[FSUT17] K. F. Früh, D. Schaudel, L. Urbas, and T. Tauchnitz, editors. *Handbuch der Prozessautomatisierung: Prozessleittechnik für verfahrenstechnische Anlagen*. Vulkan-Verlag, Essen, Germany, 6th edition, November 2017.

[FVC+17] M. Foehr, J. Vollmar, A. Calà, P. Leitão, S. Karnouskos, and A. W. Colombo. Engineering of Next Generation Cyber-Physical Automation System Architectures. In S. Biffl, A. Lüder, and D. Gerhard, editors, *Multi-Disciplinary Engineering for Cyber-Physical Production Systems*, pages 185–206. Springer International Publishing, Cham, Switzerland, 2017.

[GBMP13] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami. Internet of Things (IoT): A vision, architectural elements, and future directions. *Future Generation Computer Systems*, 29(7):1645–1660, 2013.

[GF95] M. Grüninger and M. S. Fox. Methodology for the design and evaluation of ontologies. In *Proceedings of the Workshop on Basic Ontological Issues in Knowledge Sharing (IJCAI)*, pages 1–10, Montreal, Canada, July 1995. CEUR-WS.

[GGA14] I. Grobelna, M. Grobelny, and M. Adamski. Model Checking of UML Activity Diagrams in Logic Controllers Design. In *Proceedings of the 9th International Conference on Dependability and Complex Systems (DepCoS-RELCOMEX)*, pages 233–242, Brunów, Poland, June 30 - July 4 2014.

[GHM+14] B. Glimm, I. Horrocks, B. Motik, G. Stoilos, and Z. Wang. HermiT: An OWL 2 Reasoner. *Journal of Automated Reasoning*, 53(3):245–269, 2014.

[GR15] H. Gimpel and M. Röglinger. Digital transformation: changes and chances-insights based on an empirical study. Permanent link: `http://publica.fraunhofer.de/dokumente/N-391990.html`, 2015. Fraunhofer Institute for Applied Information Technology FIT, Augsburg/ Bayreuth, Germany.

[Gru93] T. R. Gruber. A translation approach to portable ontology specifications. *Knowledge acquisition*, 5(2):199–220, 1993.

[GS14] F. Gandon and G. Schreiber. RDF 1.1 XML Syntax. Permanent URL: `https://www.w3.org/TR/rdf-syntax-grammar/`, 2014. Recommendation, World Wide Web Consortium (W3C), Cambridge, USA.

[GWGW17] I. Grobelna, R. Wiśniewski, M. Grobelny, and M. Wiśniewska. Design and Verification of Real-Life Processes With Application of Petri Nets. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 47(11):2856–2869, 2017.

[Hal93] N. Halbwachs. *Lustre program verification: the tool Lesar*, pages 139–147. Springer, Boston, USA, 1993.

[Har87] D. Harel. Statecharts: A visual formalism for complex systems. *Science of computer programming*, 8(3):231–274, 1987.

[HE08] T. Horn and J. Ebert. Ein Referenzschema für die Sprachen der IEC 61131. Technical Report 13, University of Koblenz-Landau, Koblenz, Germany, 2008.

[Hil03] E. F. Hill. *Jess in action: Java rule-based systems*. Manning Publications Co., Greenwich, USA, 2003.

[HK12] D. Hästbacka and S. Kuikka. Semantics and reasoning for control application engineering models. In *Proceedings of the 17th International Conference on Artificial Intelligence and Soft Computing (ICAISC)*, pages 647–655, Zakopane, Poland, June 3-7 2012.

[HK13] D. Hästbacka and S. Kuikka. Semantics enhanced engineering and model reasoning for control application development. *Multimedia Tools and Applications*, 65(1):47–62, 2013.

[HKC+17] A. Haller, J. Krzystof, S. Cox, D. le Phuoc, K. Taylor, and M. Lefrançois. Semantic Sensor Network Ontology. Permanent URL: `https://www.w3.org/TR/vocab-ssn/`, 2017. Recommendation, World Wide Web Consortium (W3C), Cambridge, USA.

[HKR+04] M. Horridge, H. Knublauch, A. Rector, R. Stevens, and C. Wroe. A practical guide to building OWL ontologies using the Protégé 4 and CO-ODE tools. Technical Report 1.3, University of Manchester, Manchester, UK, 2004.

[HKR10] P. Hitzler, M. Krötzsch, and S. Rudolph. *Foundations of Semantic Web technologies*. CRC Press, Boca Raton, USA, 2010.

[HLR92] N. Halbwachs, F. Lagnier, and C. Ratel. Programming and verifying real-time systems by means of the synchronous data-flow language LUSTRE. *IEEE Transactions on Software Engineering*, 18(9):785–793, 1992.

[HMS11] A. Harth, J. Maciej, and S. Staab. Semantic web architecture. In J. Domingue, D. Fensel, and J. A. Hendler, editors, *Handbook of Semantic Web Technologies*, pages 43–75. Springer, Berlin, Germany, 2011.

[Hon97] Honeywell. Honeywell Engineering Manual of Automatic Control for Commercial Buildings. Technical Report 97-72971, Honewell Inc., Minneapolis, USA, 1997.

[HPS11] I. Horroks and P. F. Patel-Schneider. KR and Reasoning on the Semantic Web: OWL. In J. Domingue, D. Fensel, and J. A. Hendler, editors, *Handbook of Semantic Web Technologies*, pages 365–398. Springer, Berlin, Germany, 2011.

[HPSB+04] I. Horrocks, P. F. Patel-Schneider, H. Boley, S. Tabet, B. Grosof, and M. Dean. SWRL: A Semantic Web Rule Language Combining OWL and RuleML. Permanent URL: `https://www.w3.org/Submission/SWRL/`, 2004. Member Submission, World Wide Web Consortium (W3C), Cambridge, USA.

[HTE15] M. Hydeman, S. T. Taylor, and B. Eubanks. Control sequences & controller programming. *ASHRAE Journal*, 57:58–62, 2015.

[HVA16] R. Harrison, D. Vera, and B. Ahmad. Engineering Methods and Tools for Cyber–Physical Automation Systems. *Proceedings of the IEEE*, 104(5):973–985, 2016.

[HWD+13] H. He, Z. Wang, Q. Dong, W. Zhang, and W. Zhu. Ontology-Based Semantic Verification for UML Behavioural Models. *International Journal of Software Engineering and Knowledge Engineering*, 23(2):117–145, 2013.

[HZ16] D. Hästbacka and A. Zoitl. Towards semantic self-description of industrial devices and control system interfaces. In *Proceedings of International Conference on Industrial Technology (ICIT)*, pages 879–884, Taipei, Taiwan, March 14-17 2016. IEEE.

[IEC08] IEC 61512-3. Batch control - Part 3: General and site recipe models and representation, 2008. International Electrotechnical Commission, Geneva, Switzerland.

[IEC12] IEC 61499-1. Function blocks - Part 1: Architecture, 2012. International Electrotechnical Commission, Geneva, Switzerland.

[IEC13a] IEC 60050-351. International electrotechnical vocabulary - Part 351: Control technology, 2013. International Electrotechnical Commission, Geneva, Switzerland.

[IEC13b] IEC 60050-351. International electrotechnical vocabulary - Part 441: Switchgear, controlgear and fuses, 2013. International Electrotechnical Commission, Geneva, Switzerland.

[IEC13c] IEC 60848. GRAFCET specification language for sequential function charts, 2013. International Electrotechnical Commission, Geneva, Switzerland.

[IEC13d] IEC 62264. Enterprise-control system integration, 2013. International Electrotechnical Commission, Geneva, Switzerland.

[IEC14a] IEC 61131-3. Programmable controllers - Part 3: Programming languages, 2014. International Electrotechnical Commission, Geneva, Switzerland.

[IEC14b] IEC 62714-1. Engineering data exchange format for use in industrial automation systems engineering - Automation Markup Language - Part 1: Architecture and general requirements, 2014. International Electrotechnical Commission, Geneva, Switzerland.

[IEC16] IEC 62424. Representation of process control engineering - Requests in P&I diagrams and data exchange between P&ID tools and PCE-CAE tools, 2016. International Electrotechnical Commission, Geneva, Switzerland.

[ISO94] ISO/IEC 7498-1. Information technology – Open Systems Interconnection – Basic Reference Model: The Basic Model, 1994. International Standardisation Organisation/ International Electrotechnical Commission, Geneva, Switzerland.

[ISO03] ISO 15926-2. Industrial automation systems and integration – Integration of life-cycle data for process plants including oil and gas production facilities – Part 2: Data model, 2003. International Standardisation Organisation, Geneva, Switzerland.

[ISO04a] ISO 10303-11. Industrial automation systems and integration - Product data representation and exchange - Part 11, 2004. International Standardisation Organisation, Geneva, Switzerland.

[ISO04b] ISO 16484-2. Building automation and control systems (BACS) - Part 2: Hardware, 2004. International Standardization Organization, Geneva, Switzerland.

[ISO05] ISO 16484-3. Building automation and control systems (BACS) - Part 3: Functions, 2005. International Standardization Organization, Geneva, Switzerland.

[ISO07] ISO/IEC 14543-3-1. Information Technology - Home Electronic System (HES) Architecture - Part 3-1: Communication Layers - Application Layer for Network Based Control of HES Class 1, 2007. International Standardization Organization/ International Eletrotechnical Commission, Geneva, Switzerland.

[ISO10] ISO 16484-1. Building automation and control systems (BACS) - Part 1: Project specification and implementation, 2010. International Standardization Organization, Geneva, Switzerland.

[ISO11] ISO 16484. Building automation and control systems (BACS), 2011. International Standardization Organization, Geneva, Switzerland.

[ISO12] ISO/IEC 14908-1. Information technology - Control network protocol - Part 1: Protocol stack, 2012. International Standardization Organization/ International Eletrotechnical Commission, Geneva, Switzerland.

[ISO13] ISO 16739. Industry Foundation Classes (IFC) for data sharing in the construction and facility management industries, 2013. International Standardisation Organisation, Geneva, Switzerland.

[ISO16] ISO/IEC 9075. Information technology – Database languages – SQL, 2016. International Standardization Organization/ International Electrotechnical Commission, Geneva, Switzerland.

[ISO17a] ISO 16484-5. Building automation and control systems (BACS) - Part 5: Data communication protocol, 2017. International Standardization Organization, Geneva, Switzerland.

[ISO17b] ISO/IEC 10646. Information technology – Universal Coded Character Set (UCS), 2017. International Standardization Organization/ International Electrotechnical Commission, Geneva, Switzerland.

[JSSF17] R. Julius, M. Schürenberg, F. Schumacher, and A. Fay. Transformation of GRAFCET to PLC code including hierarchical structures. *Control Engineering Practice*, 64:173–194, 2017.

[KB05] S. Katipamula and M. R. Brambley. Review Article: Methods for Fault Detection, Diagnostics, and Prognostics for Building Systems - A Review, Part I. *HVAC&R Research*, 11(1):3–25, 2005.

[KB13] M. Kifer and H. Boley. RIF Overview (Second Edition). Permanent URL: `https://www.w3.org/TR/rif-overview/`, 2013. Recommendation, World Wide Web Consortium (W3C), Cambridge, USA.

[KD11] A. Kiryakov and M. Damova. Storing the semantic web: Repositories. In J. Domingue, D. Fensel, and J. A. Hendler, editors, *Handbook of Semantic Web Technologies*, pages 231–298. Springer, Berlin, Germany, 2011.

[KDKO14] M. W. Krueger, R. Drath, H. Koziolek, and Z. M. Ouertani. A new era: ABB is working with the leading industry initiatives to help usher in a new industrial revolution. *ABB Review*, 4:70–75, 2014.

[KGG18] O. Kovalenko and I. Grangel-Gonzalez. AutomationML Ontology. Permanent URL: `https://w3id.org/i40/aml#`, 2018. Last accessed: 22 October 2018.

[KK08] H. Knublauch and D. Kontokostas. Shapes Constraint Language (SHACL). Permanent URL: `https://www.w3.org/TR/shacl/`, 2008. Recommendation, World Wide Web Consortium (W3C), Cambridge, USA.

[KKGR16] K. Katsigarakis, G. Kontes, G. Giannakis, and D. Rovas. Sense-think-act framework for intelligent building energy management. *Computer-Aided Civil and Infrastructure Engineering*, 31(1):50–64, 2016.

[Kle14] M. Kleinemeier. Von der Automatisierungspyramide zu Unternehmenssteuerungsnetzwerken. In T. Bauernhansl, M. Ten Hompel, and B. Vogel-Heuser, editors, *Industrie 4.0 in Produktion, Automatisierung und Logistik: Anwendung, Technologien und Migration*, pages 571–580. Springer Vieweg, Wiesbaden, Germany, 2014.

[KLW95] M. Kifer, G. Lausen, and J. Wu. Logical foundations of object-oriented and frame-based languages. *Journal of the ACM*, 42(4):741–843, 1995.

[Knu11] H. Knublauch. SPIN - SPARQL Syntax. Permanent URL: `https://www.w3.org/Submission/spin-sparql/`, 2011. Member Submission, World Wide Web Consortium (W3C), Cambridge, USA.

[KP15] A. H. Khan and I. Porres. Consistency of UML class, object and statechart diagrams using ontology reasoners. *Journal of Visual Languages & Computing*, 26:42–65, 2015.

[KRK12] M. J. Kofler, C. Reinisch, and W. Kastner. A semantic representation of energy-related information in future smart homes. *Energy and Buildings*, 47:169–179, 2012.

[KRS12] S. Kowalewski, B. Rumpe, and A. Stollenwerk. Cyber-Physical Systems - eine Herausforderung für die Automatisierungstechnik? In *Proceedings of Automation*, pages 113–116, Baden-Baden, Germany, June 13-14 2012.

[KS15] J. Kaiser and P. Stenzel. eeEmbedded D4.2: Energy System Information Model - ESIM, 2015. *ee*Embedded Consortium, Brussels, Belgium.

[KSB01] S. Kowalewski, O. Stursberg, and N. Bauer. An Experimental Batch Plant as a Test Case for the Verification of Hybrid Systems. *European Journal of Control*, 7(4):366–381, 2001.

[KSH12] M. Krötzsch, F. Simancik, and I. Horrocks. A Description Logic Primer, 2012. abs/1201.4089.

[KWH13] H. Kagermann, W. Wahlster, and J. Helbig. *Recommendations for implementing the strategic initiative INDUSTRIE 4.0: Securing the future of German manufacturing industry; final report of the Industrie 4.0 Working Group*. Forschungsunion, acatech, Frankfurt/Main, Germany, 2013.

[Lan13] C. Lange. Ontologies and languages for representing mathematical knowledge on the semantic web. *Semantic Web*, 4(2):119–158, 2013.

[Lee08] E. A. Lee. Cyber Physical Systems: Design Challenges. In *11th International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC)*, pages 363–369, Orlando, USA, May 5-7 2008. IEEE.

[LEHM11] A. Lüder, E. Estévez, L. Hundt, and M. Marcos. Automatic transformation of logic models within engineering of embedded mechatronical units. *The International Journal of Advanced Manufacturing Technology*, 54(9):1077–1089, 2011.

[LKGZ17] M. Lefrançois, J. Kalaoja, T. Ghariani, and A. Zimmermann. D2.2: The SEAS Knowledge Model, 2017. ITEA2 SEAS, Brussels, Belgium.

[LML+09] M.-K. Leung, T. Mandl, E. A. Lee, E. Latronico, C. Shelton, S. Tripakis, and B. Lickly. Scalable semantic annotation using lattice-based ontologies. In *Proceedings of the 12th International Conference on Model Driven Engineering Languages and Systems*, pages 393–407, Denver, USA, October 4-9 2009.

[LMM99] D. Latella, I. Majzik, and M. Massink. Automatic Verification of a Behavioural Subset of UML Statechart Diagrams Using the SPIN Model-checker. *Formal Aspects of Computing*, 11(6):637–664, 1999.

[LN00] E. A. Lee and S. Neuendorffer. MoML - A Modeling Markup Language in XML - Version 0.4. Technical Report 0.4, University of California, Berkeley, USA, 2000.

[LS17] E. A. Lee and S. A. Seshia. *Introduction to Embedded Systems: A Cyber-Physical Systems Approach*. MIT Press, Cambridge, USA, 2nd edition, 2017.

[LSD17] A. Lüder, N. Schmidt, and R. Drath. Standardized Information Exchange Within Production System Engineering. In S. Biffl, A. Lüder, and D. Gerhard, editors, *Multi-Disciplinary Engineering for Cyber-Physical Production Systems*, pages 235–257. Springer International

Publishing, Cham, Switzerland, 2017.

[LTV+16] L. Luyen, A. Tireau, A. Venkatesan, P. Neveu, and P. Larmande. Development of a knowledge system for big data: Case study to plant phenotyping data. In *Proceedings of the 6th International Conference on Web Intelligence, Mining and Semantics*, pages 1–9, Nimes, France, June 13-15 2016.

[Lun09] J. Lunze. *Ereignisdiskrete Systeme: Modellierung und Analyse dynamischer Systeme mit Automaten, Markovketten und Petrinetzen*. Oldenbourg, Munich, Germany, 2009.

[MB12] T. Moser and S. Biffl. Semantic Integration of Software and Systems Engineering Environments. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 42(1):38–50, 2012.

[MCGH+12] B. Motik, B. Cuenca Grau, I. Horrocks, Z. Wu, A. Fokoue, and C. Lutz. OWL 2 Web Ontology Language Profiles (Second Edition). Permanent URL: `http://www.w3.org/TR/owl2-profiles/`, 2012. Recommendation, World Wide Web Consortium (W3C), Cambridge, USA.

[Mea55] G. H. Mealy. A method for synthesizing sequential circuits. *Bell Labs Technical Journal*, 34(5):1045–1079, 1955.

[MEP10] E. Muñoz, A. Espuña, and L. Puigjaner. Towards an ontological infrastructure for chemical batch process management. *Computers & Chemical Engineering*, 34(5):668–682, 2010.

[MHH09] H. Merz, T. Hansemann, and C. Hübner. *Building automation: communication systems with EIB/KNX, LON und BACnet*. Signals and communication technology. Springer, Berlin, Germany, 2009.

[Min74] M. Minsky. A framework for representing knowledge. Technical Report AIM-306, A.I. Laboratory, Massachusetts Institute of Technology, Cambridge, USA, 1974.

[MKL18] MKLab. StarUML. `http://staruml.io`, 2018. Last accessed: 22 October 2018.

[MMWY09] W. Marquardt, J. Morbach, A. Wiesner, and A. Yang. *OntoCAPE: A re-usable ontology for chemical process engineering*. Springer, Berlin, Germany, 2009.

[Mod17a] Modelica Association. Modelica - A Unified Object-Oriented Language for Systems Modeling, April 10 2017. Modelica Association, Linköping, Sweden.

[Mod17b] Modelica Association. Modelica Standard Library, 2017. URL: `https://github.com/modelica/Modelica`, Last accessed: 22 October 2018.

[Mos16] T. Moser. The Engineering Knowledge Base Approach. In S. Biffl and M. Sabou, editors, *Semantic Web Technologies for Intelligent Engineering Applications*, pages 85–103. Springer International Publishing, Cham, Switzerland, 2016.

[MPSCG12] B. Motik, P. F. Patel-Schneider, and B. Cuenca Grau. OWL 2 Web Ontology Language Direct Semantics (Second Edition). Permanent URL: `https://www.w3.org/TR/owl2-direct-semantics/`, 2012. Recommendation, World Wide Web Consortium (W3C), Cambridge, USA.

[MPSP12] B. Motik, P. F. Patel-Schneider, and B. Parsia. OWL 2 Web Ontology Language Structural Specification and Functional-Style Syntax (Second Edition). Permanent URL: `https://www.w3.org/TR/owl2-syntax/`, 2012. Recommendation, World Wide Web Consortium (W3C), Cambridge, USA.

[Mus15] M. A. Musen. The protégé project: a look back and a look forward. *AI matters*, 1(4):4–12, 2015.

[MvH04] D. L. McGuiness and F. van Hermelen. OWL Web Ontology Language. Permanent URL: https://www.w3.org/TR/owl-features/, 2004. Recommendation, World Wide Web Consortium (W3C), Cambridge, USA.

[NK98] I. Nonaka and N. Konno. The concept of "ba": Building a foundation for knowledge creation. *California management review*, 40(3):40–54, 1998.

[NM01] N. F. Noy and D. L. McGuinness. Ontology Development 101: A Guide to Creating Your First Ontology. Technical Report SMI-2001-0880, Stanford University, Stanford, USA, 2001.

[Nor98] K. North. *Wissensorientierte Unternehmensführung: Wertschöpfung durch Wissen*. Gabler, Wiesbaden, Germany, 1998.

[NSMŠ15] P. Novák, E. Serral, R. Mordinyi, and R. Šindelář. Integrating heterogeneous engineering knowledge and tools for efficient industrial simulation model support. *Advanced Engineering Informatics*, 29(3):575–590, 2015.

[OÅD05] M. Otter, K.-E. Årzén, and I. Dressler. StateGraph - a Modelica library for hierarchical state machines. In *Proceedings of the 4th international Modelica conference*, pages 569–578, Hamburg, Germany, March 7-8 2005.

[Obj15a] Object Management Group. OMG Unified Modeling Language (OMG UML), 2015. Needham, USA.

[Obj15b] Object Management Group. XML Metadata Interchange (XMI) Specification, 2015. Needham, USA.

[Ope16] OpenMath. OpenMath. http://www.openmath.org/, 2016. Last accessed: 22 October 2018.

[PA16] L. Petnga and M. Austin. An ontological framework for knowledge modeling and decision support in cyber-physical systems. *Advanced Engineering Informatics*, 30(1):77–94, 2016.

[PFP+10] S. Plesser, M. N. Fisch, C. Pinkernell, T. Kurpick, and B. Rumpe. The Energy Navigator-A Web based Platform for functional Quality Mangement in Buildings. In *Proceedings of the 10th International Conference for Enhanced Building Operations (ICEBO)*, Kuwait City, Kuwait, October 26-28 2010.

[PHDK12] J. Ploennigs, B. Hensel, H. Dibowski, and K. Kabitzsch. BASont - A Modular, Adaptive Building Automation System Ontology. In *Proceedings of the 38th Annual Conference on IEEE Industrial Electronics Society (IECON)*, pages 4827–4833, Montreal, Canada, October 25-28 2012.

[PLC09] PLCopen. XML Formats for IEC 61131-3, 2009. Gorinchem, The Netherlands.

[Ple13] S. Plesser. *Aktive Funktionsbeschreibungen zur Planung und Überwachung des Betriebs von Gebäuden und Anlagen*. PhD thesis, TU Braunschweig, Braunschweig, Germany, 2013.

[Pol09] M. Polanyi. *The tacit dimension*. University of Chicago press, Chicago, USA, 2009.

[PRPO06] K. Poschlad, M. A. Remelhe Pereira, and M. Otter. Modeling of an experimental Batch Plant with Modelica. In *Proceedings of the 5th International Modelica Conference*, pages 651–660, Vienna, Austria, September 4-5 2006.

[PRR12] G. Probst, S. Raub, and K. Romhardt. *Wissen managen: Wie Unternehmen ihre wertvollste Ressource optimal nutzen*. Springer Gabler, Wiesbaden, Germany, 2012.

[PS17] E. Prud'hommeaux and A. Seaborne. The SPARQL query language for RDF. Permanent URL: https://www.w3.org/TR/rdf-sparql-query/, 2017. Recommendation, World Wide Web Consortium (W3C), Cambridge, USA.

[PT16] P. Pauwels and W. Terkaj. EXPRESS to OWL for construction industry: Towards a recommendable and usable ifcOWL ontology. *Automation in Construction*, 63:100–133, 2016.

[PVDV+11] P. Pauwels, D. Van Deursen, R. Verstraeten, J. De Roo, R. De Meyer, R. Van de Walle, and J. Van Campenhout. A semantic rule checking environment for building performance checking. *Automation in Construction*, 20(5):506–518, 2011.

[PVGC18] M. Poveda Villalón and R. García Castro. SAREF extension for building devices. Permanent URL: `https://w3id.org/def/saref4bldg`, 2018.

[PZL17] P. Pauwels, S. Zhang, and Y.-C. Lee. Semantic web technologies in AEC industry: A literature overview. *Automation in Construction*, 73:145–165, 2017.

[RCW13] K. Ridgeway, C. W. Clegg, and D. J. Williams. *The factory of the future*. The National Metals Technology Centre, University of Sheffield AMRC, Sheffield, UK, October 2013.

[RFW09] S. Runde, A. Fay, and W. O. Wutzke. Knowledge-based requirement-engineering of building automation systems by means of semantic web technologies. In *Proceedings of the 7th International Conference on Industrial Informatics (INDIN)*, pages 267–272, Cardiff, UK, June 23-26 2009. IEEE.

[RGPK08] C. Reinisch, W. Granzer, F. Praus, and W. Kastner. Integration of heterogeneous building automation systems using ontologies. In *Proceedings of the 34th Annual Conference of IEEE Industrial Electronics Society (IECON)*, pages 2736–2741, Orlando, Florida, November 10-13 2008. IEEE.

[RK07] T. Ritala and S. Kuikka. Uml automation profile: Enhancing the efficiency of software development in the automation industry. In *Proceedings of the International Conference on Industrial Informatics (INDIN)*, volume 2, pages 885–890, Vienna, Austria, June 23-27 2007. IEEE.

[RKIK11] C. Reinisch, M. J. Kofler, F. Iglesias, and W. Kastner. Thinkhome energy efficiency in future smart homes. *EURASIP Journal on Embedded Systems*, 2011(104617):1–18, 2011.

[RKK10] C. Reinisch, M. J. Kofler, and W. Kastner. ThinkHome: A smart home as digital ecosystem. In *Proceedings of the 4th International Conference on Digital Ecosystems and Technologies (DEST)*, pages 256–261, Dubai, UAE, April 13-16 2010. IEEE.

[RQd12] C. Rupp, S. Queins, and die SOPHISTen. *UML 2 glasklar: Praxiswissen für die UML-Modellierung*. Hanser, Munich, Germany, 2012.

[Rud11] S. Rudolph. Foundations of description logics. In A. Polleres, C. d'Amato, M. Arenas, S. Handschuh, P. Kroner, S. Ossowski, and P. Patel-Schneider, editors, *Lecture Notes in Computer Sciences*, volume 6248, pages 76–136. Springer, Berlin, Germany, 2011.

[Run11] S. Runde. *Wissensbasierte Engineeringunterstützung in der Automatisierungstechnik am Beispiel der Gebäudeautomation*, volume 434 of *Fortschritt-Berichte VDI : Reihe 20, Rechnerunterstützte Verfahren*. VDI-Verlag, Düsseldorf, Germany, 2011.

[RWT11] H. Rijgersberg, M. Wigham, and J. Top. How semantics can improve engineering processes: A case of units of measure and quantities. *Advanced Engineering Informatics*, 25(2):276–287, 2011.

[Sab16] M. Sabou. An Introduction to Semantic Web Technologies. In S. Biffl and M. Sabou, editors, *Semantic Web Technologies for Intelligent Engineering Applications*, pages 53–84. Springer International Publishing, Cham, Switzerland, 2016.

[Sal05] T. I. Salsbury. A SURVEY OF CONTROL TECHNOLOGIES IN THE BUILDING AUTOMATION INDUSTRY. *IFAC Proceedings Volumes*, 38(1):90–100, 2005.

[SBF98] R. Studer, V. R. Benjamins, and D. Fensel. Knowledge engineering: principles and methods. *Data & knowledge engineering*, 25(1-2):161–197, 1998.

[Sch07] J. Schein. An Information Model for Building Automation Systems. *Automation in Construction*, 16(2):125–139, 2007.

[SE13] P. Shvaiko and J. Euzenat. Ontology matching: state of the art and future challenges. *IEEE Transactions on knowledge and data engineering*, 25(1):158–176, 2013.

[SH17] A. Strahilov and H. Hämmerle. Engineering Workflow and Software Tool Chains of Automated Production Systems. In S. Biffl, A. Lüder, and D. Gerhard, editors, *Multi-Disciplinary Engineering for Cyber-Physical Production Systems*, pages 207–234. Springer International Publishing, Cham, Switzerland, 2017.

[SKK+12] J. Sztipanovits, X. Koutsoukos, G. Karsai, N. Kottenstette, P. Antsaklis, V. Gupta, B. Goodwine, J. Baras, and S. Wang. Toward a science of cyber-physical system integration. *Proceedings of the IEEE*, 100(1):29–44, 2012.

[SKN16] M. Sabou, O. Kovalenko, and P. Novák. Semantic Modelling and Acquisition of Engineering Knowledge. In S. Biffl and M. Sabou, editors, *Semantic Web Technologies for Intelligent Engineering Applications*, pages 105–136. Springer International Publishing, Cham, Switzerland, 2016.

[Sno00] D. Snowden. The ASHEN Model: an enabler of action. *Knowledge Management*, 3(7):14–17, 2000.

[Sno05] D. Snowden. The ASHEN Model: an enabler of action. `http://old.cognitive-edge.com/wp-content/uploads/2000/04/7-Organic-KM-1-of-3-ASHEN.pdf`, 2005. Last accessed: 22 October 2018.

[Sow14] J. F. Sowa. *Principles of semantic networks: Explorations in the representation of knowledge*. Morgan Kaufmann, Burlington, USA, 2014.

[SPG+07] E. Sirin, B. Parsia, B. C. Grau, A. Kalyanpur, and Y. Katz. Pellet: A practical OWL-Dl reasoner. *Web Semantics: science, services and agents on the World Wide Web*, 5(2):51–53, 2007.

[SQM+07] S. Solomon, D. Qin, M. Manning, Z. Chen, M. Marquis, K. Averyt, M. Tignor, and H. Miller, editors. *Contribution of Working Group I to the Fourth Assessment Report of the Intergovernmental Panel on Climate Change*. Cambridge University Press, Cambridge, UK, 2007.

[SSKD11] T. Sauter, S. Soucek, W. Kastner, and D. Dietrich. The Evolution of Factory and Building Automation. *IEEE Industrial Electronics Magazine*, 5(3):35–48, 2011.

[TA16] T. Tudorache and L. Alani. Semantic Web Solutions in the Automotive Industry. In S. Biffl and M. Sabou, editors, *Semantic Web Technologies for Intelligent Engineering Applications*, pages 297–326. Springer International Publishing, Cham, Switzerland, 2016.

[TAO98] C. Tasso, E. Arantes, and E. Oliveira. *Development of knowledge-based systems for engineering*. Springer, Vienna, Austria, 1998.

[TBB+15] N. M. Tomašević, M. v. Batić, L. M. Blanes, M. M. Keane, and S. Vraneš. Ontology-Based Facility Data Model for Energy Management. *Advanced Engineering Informatics*, 29(4):971–984, 2015.

[TRPE17] H. Thaller, R. Ramler, J. Pichler, and A. Egyed. Exploring code clones in programmable logic controller software. In *Proceedings of the 22nd International Conference on Emerging Technologies and Factory Automation (ETFA)*, pages 1–8, Limassol, Cyprus, September 12-15 2017. IEEE.

[TS 18] TS 0012. oneM2M Base Ontology, 2018. oneM2M, `http://onem2m.org`, Last accessed: 22 October 2018.

[TSD11] E. Turban, R. Sharda, and D. Delen. *Decision Support and Business Intelligence Systems*. Pearson Education, London, UK, 2011.

[TVM+09] J. Trojanová, J. Vass, K. Macek, J. Rojíček, and P. Stluka. Fault diagnosis of air handling units. *IFAC Proceedings Volumes*, 42(8):366–371, 2009.

[USG16] USGBC. LEED v4 for building operations and maintenance. Technical Report 4, US Green Building Council (USGBC), Washington D.C., USA, 2016.

[VDI07] VDI 3825. Automation and control of air-conditioning systems examples, 2007. Verein Deutscher Ingenieure (VDI), Düsseldorf, Germany.

[VDI08] VDI 3814-6. Building automation and control systems (BACS) Graphical description of logic control tasks, 2008. Verein Deutscher Ingenieure (VDI), Düsseldorf, Germany.

[VDI09a] VDI 3814. Building automation, 2009. Verein Deutscher Ingenieure (VDI), Düsseldorf, Germany.

[VDI09b] VDI 5610-1. Wissensmanagement im Ingenieurwesen - Grundlagen, Konzepte, Vorgehen, 2009. Verein Deutscher Ingenieure (VDI), Düsseldorf, Germany.

[VDI11a] VDI 3813. Building automation and control systems - Room automation, 2011. Verein Deutscher Ingenieure (VDI), Düsseldorf, Germany.

[VDI11b] VDI 3813-2. Building automation and control systems (BACS) Room control function (RA functions), 2011. Verein Deutscher Ingenieure (VDI), Düsseldorf, Germany.

[VDI13] VDI 3814-1. Building automation and control systems (BACS) system basics, 2013. Verein Deutscher Ingenieure (VDI), Düsseldorf, Germany.

[VDI15] VDI/VDE GMA. Automation 2020 – Bedeutung und Entwicklung der Automation bis zum Jahr 2020 – Thesen und Handlungsfelder, January 2015. VDI/VDE-Gesellschaft Mess- und Automatisierungstechnik (GMA), Düsseldorf, Germany.

[VH+14a] M. Vegetti, G. P. Henning, et al. ISA-88 Formalization. A Step Towards its Integration with the ISA-95 Standard. In *Proceedings of the 6th Workshop on Formal Ontologies meet Industry (FOMI) and 8th International Conference on Formal Ontology in Information Systems (FOIS)*, pages 1–9, Rio de Janeiro, Brazil, September 22 2014. CEUR-WS.org.

[VH14b] B. Vogel-Heuser. Herausforderungen und Anforderungen aus Sicht der IT und der Automatisierungstechnik. In T. Bauernhansl, M. Ten Hompel, and B. Vogel-Heuser, editors, *Industrie 4.0 in Produktion, Automatisierung und Logistik: Anwendung, Technologien und Migration*, pages 37–48. Springer Vieweg, Wiesbaden, Germany, 2014.

[VHDB13] B. Vogel-Heuser, C. Diedrich, and M. Broy. Anforderungen an CPS aus Sicht der Automatisierungstechnik. *Automatisierungstechnik (at)*, 61(10):669–676, 2013.

[VHDF+14] B. Vogel-Heuser, C. Diedrich, A. Fay, S. Jeschke, S. Kowalewski, M. Wollschlaeger, and P. Göhner. Challenges for software engineering in automation. *Journal of Software Engineering and Applications*, 7(5):440–451, 2014.

[VHFST15] B. Vogel-Heuser, A. Fay, I. Schaefer, and M. Tichy. Evolution of software in automated production systems: Challenges and research directions. *Journal of Systems and Software*, 110:54–84, 2015.

[VJS+98] S. Viswanathan, C. Johnsson, R. Srinivasan, V. Venkatasubramanian, and K. E. Årzén. Automating operating procedure synthesis for batch processes: Part i. knowledge representation and planning framework. *Computers & Chemical Engineering*, 22(11):1673–1685, 1998.

[VSH+16] R. Verborgh, M. V. Sande, O. Hartig, J. V. Herwegen, L. D. Vocht, B. D. Meester, G. Haesendonck, and P. Colpaert. Triple pattern fragments: A low-cost knowledge graph interface for the web. *Journal of Web Semantics*, 37-38:184–206, 2016.

[Vya13] V. Vyatkin. Software Engineering in Industrial Automation: State-of-the-Art Review. *IEEE Transactions on Industrial Informatics*, 9(3):1234–1249, 2013.

[W3C12] W3C OWL Working Group. OWL 2 Web Ontology Language. Permanent URL: `https://www.w3.org/TR/owl2-overview/`, 2012. Recommendation, World Wide Web Consortium (W3C), Cambridge, USA.

[Wet18] M. Wetter. OpenBuildingControl - Performance Evaluation, Specification and Verification of Building Control Sequences. `http://obc.lbl.gov/`, 2018. Last accessed: 22 October 2018.

[Wic16] H. Wicaksono. *An Integrated Method for Information and Communication Technology (ICT) Supported Energy Efficiency Evaluation and Optimization in Manufacturing: Knowledge-based Approach and Energy Performance Indicators (EnPI) to Support Evaluation and Optimization of Energy Efficiency*. PhD thesis, Karlsruhe Institute of Technology (KIT), Karlsruhe, Germany, 2016.

[WJRO14] H. Wicaksono, F. Jost, S. Rogalski, and J. Ovtcharova. Energy efficiency evaluation in manufacturing through an ontology-represented knowledge base. *Intelligent Systems in Accounting, Finance and Management*, 21(1):59–69, 2014.

[WMM11] A. Wiesner, J. Morbach, and W. Marquardt. Information integration in chemical process engineering based on semantic technologies. *Computers & Chemical Engineering*, 35(4):692–708, 2011.

[WR12] K. Wenzel and H. Reinhardt. Mathematical computations for linked data applications with OpenMath. In *Joint Proceedings of the 24th Workshop on OpenMath and the 7th Workshop on Mathematical User Interfaces (MathUI)*, pages 38–48, Bremen, Germany, 2012.

[WZNP14] M. Wetter, W. Zuo, T. S. Nouidui, and X. Pang. Modelica Buildings Library. *International Journal of Building Performance Simulation*, 7(4):253–270, 2014.

Bibliography of the Author

[SBNM16] G. F. Schneider, A. Bougain, P. S. Noisten, and M. Mitterhofer. Information Requirement Definition for BIM: A Life Cycle Perspective. In *Proceedings of the 11th European Conference on Product and Process Modelling (ECPPM)*, pages 225–233, Limassol, Cyprus, September 7-9 2016.

[Sch17] G. F. Schneider. Towards Aligning Domain Ontologies with the Building Topology Ontology. In *Proceedings of the 5th Linked Data in Architecture and Construction Workshop (LDAC)*, pages 1–8, Dijon, France, 2017.

[SPS17] G. F. Schneider, P. Pauwels, and S. Steiger. Ontology-based Modeling of Control Logic in Building Automation Systems. *IEEE Transactions on Industrial Informatics*, 13(6):3350–3360, 2017.

[SPT19] G. F. Schneider, G. A. Peßler, and W. Terkaj. Knowledge-based Conversion of Finite State Machines in Manufacturing. *Procedia Manufacturing*, 28:189–194, 2019.

[SWO19] G. F. Schneider, H. Wicaksono, and J. Ovtcharova. Virtual engineering of cyber-physical automation systems: The case of control logic. *Advanced Engineering Informatics*, 39:127–143, 2019.

[TSP17] W. Terkaj, G. F. Schneider, and P. Pauwels. Reusing Domain Ontologies in Linked Building Data: the Case of Building Automation and Control. In *Proceedings of the 8th International Workshop on Formal Ontologies Meet Industry (FOMI)*, pages 1–12, Bolzano, Italy, September 21 2017. CEUR-WS.org.

A Foundations of the Semantic Web

Ever since the internet was invented it was meant to be a medium to transport information from person-to-person and, potentially, from machine-to-machine. The internet in its beginnings constitutes a linked network of hypertext media, where information can be displayed for humans and other related information can be linked to it. However, since its beginnings finding respective information from the huge number of websites has always been a challenging task. The idea of the introduction of the semantic web [BLHL01] is to give data published on the web a well-defined meaning, which *machines* can understand. Hence, potentially, utilising the example from Sabou [Sab16], when using a search engine to search for the researcher *Sabou* the web profile of *Marta Sabou*[1] is displayed and information about her rather then information on the small town *Sabou*[2] in Burkina Faso is returned.

Within this chapter a brief overview and introduction to the building blocks of SWT [BLHL01] is presented. The material presented in this chapter is condensed knowledge from text books [HKR10, Rud11, DFH11, Sab16] where the interested reader is referred to for in depth treatment of the material.

SWT are used for the implementation of the semantic model as described in detail in Chapter 4 as well as the knowledge-based methods presented in Chapter 5 of this thesis. The foundations of knowledge representation are introduced in Section 2.2. If not indicated differently the code examples presented here are given using turtle syntax [BBLPC14].

Similar to the Open Systems Interconnection (OSI) [ISO94] reference model, which structures communication systems in distinct layers [HMS11], the *semantic web stack* is introduced in SWT as illustrated in Figure A.1. The stack aims at providing a conceptual model and structure of SWT. Its foundations are URIs to identify resources on the web and UNICODE [ISO17b] for the encoding. The XML technology [BPSM+06] provides the next layer and a standardised syntax. The subsequent layers, e.g. expressing taxonomies via RDFS [BG14], ontology modelling using OWL [W3C12] or querying triples from a triple store, a data base for RDF, via SPARQL [PS17], are introduced in the following sections. It might be noticed that in contrast to the layers of the OSI model, which can operate separately from each other, the SWT layers

[1] ORCID: 0000-0001-9301-8418

[2] `https://www.wikidata.org/wiki/Q3460692`, Last accessed: 22 October 2018

cannot always be clearly distinguished. For example, both in RDFS and in OWL the concept of a class exists [HMS11].

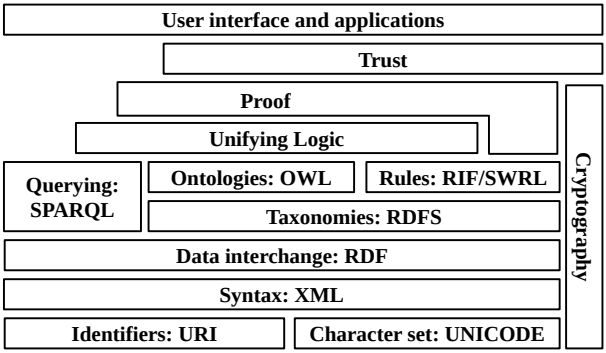


Figure A.1: The Semantic Web stack: A conceptual model and structure for technologies associated to the semantic web (adapted from Harth *et al.* [HMS11]). Acronyms are explained in the text.

A.1 Description Logics

DL are a kind of knowledge representation formalism, where DL are a fragment of FOL. FOL are a formalism from the mathematical logic domain [Rud11]. Within DL it is possible to express knowledge on a domain of interest and three entities exists in this regard: individuals, roles and concepts [KSH12]. Individuals are unique entities of a respective domain, roles are binary relationships among these individuals and concepts group individuals depended on their characteristics [KSH12]. These entities have a direct counterpart in FOL, where individuals can be identified as constants, roles as binary predicates and concepts as unary predicates [KSH12, Rud11].

For instance, it is possible to define formal statements, also referred to as axioms, encoding knowledge on individuals, concepts and more complex statements. The following DL statements are denoted following a notation inspired by set theory, which can be mapped to equivalent FOL statements [Rud11].

For instance, assertional knowledge on individuals can be expressed, e.g. the statement

```
TemperatureSensor( rt001 )
```

describes an individual `rt001` as a member of the set `TemperatureSensor`. Also, relationships between individuals can be expressed, e.g. the statement

`isLocatedIn( rt001 , room1 )`

describes that a binary relationship `isLocatedIn` exists between the individuals `rt001` and `room1`. Knowledge on concepts, e.g. subsumption can be expressed as exemplified in the following statement for a `TemperatureSensor` to be a subset of the set of all `Sensors`:

$$\texttt{TemperatureSensor} \sqsubseteq \texttt{Sensor}$$

Other statements can be expressed as part of a DL knowledge base, such as *role inclusion* [KSH12]. For example, the statement

$$\texttt{isLocatedIn} \sqsubseteq \texttt{feeds}$$

specifies a 'subrole' [KSH12] relationship between the role `isLocatedIn` and `feeds`. This implies logically that each pair of individuals related to each other via the `isLocatedIn` relationship, also is related to each other via the `feeds` relationship [KSH12, Rud11].
More complex axioms are possible but are omitted here. All statements together allow to model a respective domain. Through the formal nature of DL a machine is able to draw inferences from these statements (see Section 2.2 and Brachman & Levesque [BL04]). The definition of DL are always motivated by generating fragments, which are decidable, in contrast to FOL, which are in general undecideable [Rud11]. The reason for this are practical considerations as the undecideable characteristic may lead to infinite execution time of reasoning algorithms in applications of the technology.
In practical applications the logic $\mathcal{AL}$ can be gradually extended [PA16] to add the respective expressibility needed, of course, by adding additional complexity for the respective reasoning tasks. Table A.1 summarises the supported concepts of the DL $\mathcal{ALC}$, which extends $\mathcal{AL}$ by concrete concepts [PA16], where A is a concept name, C and D are concepts, R is a role and o is an individual name [HPS11]. $\mathcal{SROIQ}(\mathcal{D})$ constitutes the formal basis of OWL Full [MPSCG12] (see Section A.3). As OWL Full is in general undecidable a popular dialect is OWL-DL which is decidable and its formal basis is $\mathcal{SHOIN}(\mathcal{D})$. The meaning of the characters used for $\mathcal{SHOIN}(\mathcal{D})$ is as follows [HPS11, PA16]:

- $\mathcal{S}$ includes everything from $\mathcal{ALC}$ extended by role transitivity;
- $\mathcal{H}$ refers to role hierarchy;
- $\mathcal{O}$ refers to nominals, which allows using individuals in the terminological knowledge of the knowledge base;
- $\mathcal{I}$ refers to inverse roles;

Table A.1: Syntax and semantics of $\mathcal{ALC}$; adapted from Petnga & Austin [PA16] and Horrocks *et al.* [HPS11].

Name	Syntax	Semantics
atomic concept	A	$A^{\mathcal{I}} \subseteq \triangle^{\mathcal{I}}$
atomic role	R	$R^{\mathcal{I}} \subseteq \triangle^{\mathcal{I}} \times \triangle^{\mathcal{I}}$
top	$\top$	$\triangle^{\mathcal{I}}$
bottom	$\bot$	$\emptyset$
negation	$\neg C$	$\triangle^{\mathcal{I}} \setminus C^{\mathcal{I}}$
conjunction	$C \sqcap D$	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$
disjunction	$C \sqcup D$	$C^{\mathcal{I}} \cup D^{\mathcal{I}}$
existential quantification	$\exists R.C$	$\{o \mid \exists o'.\langle o,o'\rangle \in R^{\mathcal{I}} \, and \, o' \in C^{\mathcal{I}}\}$
value restriction	$\forall R.C$	$\{o \mid \forall o'.\langle o,o'\rangle \in R^{\mathcal{I}} \, implies \, o' \in C^{\mathcal{I}}\}$

- $\mathcal{N}$ refers to unqualified number restrictions;
- $(\mathcal{D})$ denotes data types.

A full treatment of the theory of DL is out of the scope of this thesis. Instead the interested reader is referred to the respective text books by Baader *et al.* [BCM+03] and Rudolph [Rud11].

A.2 Resource Description Framework (Schema) - RDF(S)

The RDF [CWL14] is the foundational data model adopted by all SWT. Within the RDF statements in the form of triples are defined consisting of a *subject*, *predicate* and *object*. In general a resource identified by a URI can be anything such as a web page, a relationship among resources, a number or a physical thing such as buildings (see [CWL14]). Often Hyper Text Transfer Protocol (HTTP) [FR14] URIs are used to reference a resource but URIs are not restricted to this.

The basic possibilities for triple statements in RDF are depicted in Figure A.2. In the figure resources are illustrated as ellipses, literals are depicted as squares and predicates are arrows. All triple statements consist of the subject, predicate, object pattern. In a triple subject and an object can be resources or the subject is a resource and the object is a literal. In both cases the predicate is termed a property. Resources and properties can be identified by a URI. The resulting structure is a *Graph* where properties are directed edges and resources and literals

are the nodes. For serialising RDF triples into a file a number of formats exist, e.g. Turtle [BBLPC14] or RDF/XML [GS14].

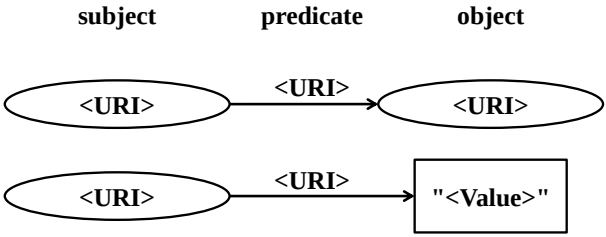


Figure A.2: Statements in RDF are triples in the form subject, predicate and object. Each is identified by its URI. Resources (ellipses) can be related to other resources or to literals (squares) by a property (arrows).

RDF is the foundational data model for the semantic web. In addition to link resources and provide semantics to data, knowledge can be specified in a machine interpretable form. This enables machines to consume formalised knowledge and reason upon it. To enable this, additional semantics are needed. The Resource Description Framework Schema (RDFS) [BG14] is a knowledge representation language [Sab16], which constitutes the next layer in the Semantic Web stack. In RDFS it is possible to declare a resource to be a `rdfs:Class` and to build taxonomies of classes by establishing a hierarchy among two classes through defining one class as a `rdfs:subClassOf` another. Similarly, this is possible for a `rdf:Property` to by defined as a `rdfs:subPropertyOf` another property. Typing of resources can be expressed via the `rdf:type` property between a resource and a class.
Literals (`rdfs:Literal`) in RDFS are unicode encoded strings. Optionally there can be language tags for plain literals and data types for typed literals. Examples are `"123.0"^^xsd:double` for a double-precision floating point number, `"Control Actor"@en` for a string with a language tag or `"2018-11-11T11:11:00+00:00"^^xsd:dateTimeStamp` for a literal encoding a date time stamp.
It is possible to define domain (`rdfs:domain`) and range (`rdfs:range`) restrictions to imply class membership of an individual, when using a certain property. For instance, the following sentence can be axiomised using range restrictions 'The range of property `ctrl:hasInput` is `ctrl:ControlActor`'. Having this axiom in a knowledge base allows to classify respective individuals related to via the described property.
RDF and RDFS offer some additional features for example to describe lists (`rdf:first`, `rdf:rest`). Further details on RDF and RDFS can be found in the respective literature ([CWL14, BG14, HKR10, Sab16, DFH11]).

A.3 Web Ontology Language - OWL

The OWL [W3C12] was introduced as the expressibility of existing languages such as RDFS is limited. For instance, it is not possible in RDFS to define that something is not true [HKR10]. OWL has a standardised syntax [MPSP12] and a formal semantics [MPSCG12]. The second version OWL 2 is the current recommended version and supersedes the former version OWL 1 [MvH04], which, however is still in use and remains a recommendation of the W3C. The history of the development of OWL is summarised in Domingue *et al.* [DFH11]. For convenience OWL is used to refer to OWL 2 in this thesis.

A.3.1 Components of OWL

OWL is a knowledge representation language and can be used to formally specify a domain of interest. The basic components of OWL ontologies are *individuals*, *properties* and *concepts* [HKR+04]. As an example, Figure A.3 shows how to describe terminological (Tbox) and assertional (Abox) knowledge using OWL. Note, the nomenclature used is defined in Figure 4.1, p. 63.

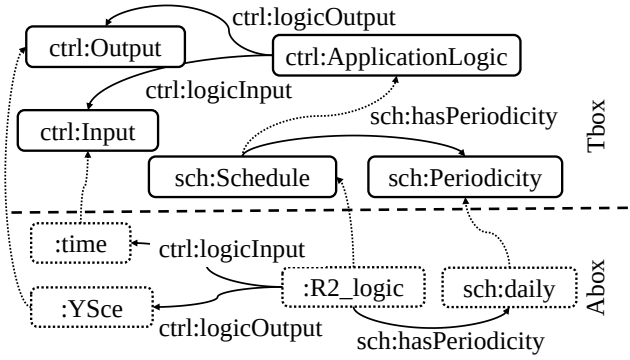


Figure A.3: Figure illustrating the representation of the assertional knowledge (ABox) through individuals, and terminological knowledge (TBox) in OWL. Graphical nomenclature as defined in Figure 4.1, p. 63.

Individuals (`owl:NamedIndividual`) represent assertional knowledge on items in the domain of interest, e.g. the statement `:Cologne rdf:type owl:NamedIndividual` could be an individual referring to the city of Cologne[3]. *Properties* are binary relationships

[3] `https://www.wikidata.org/wiki/Q365`, Last accessed: 22 October 2018

between two resources or a resource and a literal. In OWL properties can be further specified, i.e. `owl:ObjectProperty` refers to a property, which relates two resources and `owl:DataTypeProperty` refers to a property, which relates a resource and a literal. Further specification of properties is detailed in the subsequent sections. *Concepts* in OWL are interpreted as sets and individuals can be classified by concepts [HKR+04]. The terms concepts and class are used synonymically in this context.

A.3.2 A Closer Look on Properties in OWL

In OWL `owl:ObjectProperty` and `owl:DataTypeProperty` are distinguished as mentioned above. In OWL it is possible to restrict the cardinality of properties, e.g. to restrict the minimum, maximum and exact number of relationships a resource may have via a certain property [Sab16].
Additionally the characteristics of object properties can be further specified via object property restrictions [MPSP12]:

- **Functional**: A property defined as `owl:FunctionalProperty` implies that only one individual can relate via this property to another;
- **Inverse Functional**: A property defined as `owl:InverseFunctionalProperty` implies that 'there can be at most one individual related to that individual via the property' [HKR+04];
- **Transitive**: A property defined as `owl:TransitiveProperty` implies that, if an individual A relates via a property to an individual B and B via the same property to an individual C, also A can be related to C via this property;
- **Symmetric**: A property defined as `owl:SymmetricProperty` implies that if an individual A relates via a property p to individual B, then also B relates via p to A holds;
- **Asymmetric**: A property defined as `owl:AsymmetricProperty` prohibits the symmetry as defined for `owl:SymmetricProperty`;
- **Reflexive**: A property defined as `owl:ReflexiveProperty` implies that an individual using the property also must relate to itself via this property;
- **Irreflexive**: A property defined as `owl:IrreflexiveProperty` implies that an individual cannot relate to itself via this property. Moreover, individuals related by such a property or considered to be different from each other [HKR+04].

Properties can be specified from another property via the `rdfs:subPropertyOf` relationship. Also, an inverse of a property can be specified via `owl:inverseOf`.

A.3.3 Specific Features of OWL

Within OWL all axioms supported RDF and RDFS are possible. Additionally, OWL defines additional expressive capabilities, which are outlined in the following paragraphs.
Some of these language axioms focus on defining abstract class descriptions via the definition of restrictions on properties. A *value restriction* allows to define abstract classes, where individuals, which have a relationship to a certain individual, are classified as members of this class. For example, in the following code listing a value restriction is defined where every individual which is related via the `hasRelevance` property to the individual `:Important` is a member of this abstract class. The class `FireSafety` is a subclass of this abstract class.

Code A.1: Example for the definition of the abstract class `FireSafety` via a value restriction. In turtle syntax [BBLPC14].

```
:FireSafety rdf:type owl:Class ;
  rdfs:subClassOf [
    rdf:type owl:Restriction ;
    owl:onProperty :hasRelevance ;
    owl:hasValue :Important
  ] .
```

Universal restrictions can be used to define abstract classes. A universal restriction allows to restrict the relationship via a given property *only* to individuals of a specific class. An example for a universal restriction is given in the following. If an individual A is in a relationship with an other individual B via the `hasApplicationLogic` property, then individual B has to be member of the class `ApplicationLogic` or not in a relationship to A using this property at all. The latter is a common pitfall and has to be taken into account when designing universal restrictions [HKR+04]. It can be noted that a restriction on the range of a property is semantically equivalent to a universal restriction [HKR10].

Code A.2: Example for a universal restriction. In turtle syntax [BBLPC14].

```
:ControlActor rdf:type owl:Class ;
  rdfs:subClassOf [
    rdf:type owl:Restriction ;
    owl:onProperty :hasApplicationLogic ;
    owl:allValuesFrom :ApplicationLogic
  ] .
```

Existential restrictions allow to specify that an individual of a class should be related via a certain property to *at least* one individual of a specific class. An example describing this is given in the following code listing, where individuals of the class `:ControlActor` should be related to at least one individual of the class `ApplicationLogic` via the `hasApplicationLogic` property.

Code A.3: Example for an existential restriction. In turtle syntax [BBLPC14].

```
:ControlActor rdf:type owl:Class ;
  rdfs:subClassOf [
    rdf:type owl:Restriction ;
    owl:onProperty :hasApplicationLogic ;
    owl:someValuesFrom :ApplicationLogic
  ] .
```

When in a class an existential restriction is defined and a universal restriction is added then this duo of restrictions is termed *Closure Axiom*. An example is given in the following code listing.

Code A.4: Example for a closure axiom. In turtle syntax [BBLPC14].

```
:ControlActor rdf:type owl:Class ;
  rdfs:subClassOf [
    rdf:type owl:Restriction ;
    owl:onProperty :hasApplicationLogic ;
    owl:someValuesFrom :ApplicationLogic
  ] ,
  [ rdf:type owl:Restriction ;
    owl:onProperty :hasApplicationLogic ;
    owl:allValuesFrom :ApplicationLogic
  ] .
```

Cardinality restrictions can be used when the actual number of relationships of an individual via a certain property to other individuals should be restricted. If additionally the class of the restricted individuals is limited these restrictions are termed *Qualified Cardinality Restrictions*. In the following example the three possibilities to restrict the number of tyres of a car to be at least four, exactly four and at most four are illustrated with qualified cardinality restrictions.

Code A.5: Example for defining qualified cardinality restrictions. In turtle syntax [BBLPC14].

```
:Car rdf:type owl:Class ;
 rdfs:subClassOf [
      rdf:type owl:Restriction ;
      owl:onProperty :hasTyres ;
      owl:minQualifiedCardinality "4" ;
      owl:onClass :Tyre
    ] ,
    [ rdf:type owl:Restriction ;
      owl:onProperty :hasTyres ;
      owl:qualifiedCardinality "4" ;
      owl:onClass :Tyre
    ] ,
    [ rdf:type owl:Restriction ;
      owl:onProperty :hasTyres ;
      owl:maxQualifiedCardinality "4" ;
      owl:onClass :Tyre
```

```
    ] .
```

OWL provides a number of additional axioms, which are briefly summarised here [HPS11, HKR10, W3C12, MPSP12]. Note, not all language axioms are supported in all OWL profiles (see Section A.3.4).
For individuals beyond the assignments known from RDF it is possible to state that two individuals are different from each other (`owl:differentFrom`) or logically the same (`owl:sameAs`). It is possible to negate an assertion on an individual (`owl:NegativePropertyAssertion`).
Additional to the expressions from RDF, RDFS and the already described restrictions it is possible to describe equivalence of classes (`owl:equivalentClass`) and equivalence of properties (`owl:equivalentProperty`), intersection (`owl:intersectionOf`), union (`owl:unionOf`) and complement of classes (`owl:complementOf`). Properties can be defined explicitly disjoint (`owl:propertyDisjointWith`). OWL provides means to express property chain axioms (`owl:propertyChainAxiom`), that is if an individual A is related to B via some property p and B is related via an arbitrary number of properties s to C then A related to C via an property m. Note, property chain axioms easily can lead to an undecidable ontology. On data properties (`owl:onDataType`) restrictions can be defined, e.g. to restrict the value range (`xsd:minInclusive`) or enumerations (`owl:oneOf`).
It might be the case that the language axioms of OWL are not sufficient to represent the required knowledge of a use case. Technologies exist which support more complex statements, in particular for rules the SWRL [HPSB$^+$04] is to be mentioned.

A.3.4 OWL Profiles

A number of OWL dialects and alternatives such as Datalog exist [KD11, PA16]. The main aim of the different dialects is to manage the trade-off between expressibility and computational complexity (see comparison in Figure A.4). In OWL three dialects [MCGH$^+$12] have been proposed and standardised to manage this trade-off, while still providing a standardised set of supported semantics. It may be be noted, also visible from Figure A.4, many more dialects and alternatives exist but not all are standardised.

- *OWL 2 EL* has been designed for representing knowledge with a large quantity of assertional knowledge such as medical ontologies [HPS11]. The profile offers reasoning time, which increases polynomial with respect to the number of individuals [MCGH$^+$12] and specific reasoners are available with associated superior performance in comparison to off-the-shelf OWL-DL reasoners;
- *OWL 2 QL* constitutes a profile with increased expressibility and complexity supporting all most all features of the OWL 2 DL profile. It is designed for the efficient answering

of queries [HPS11]. The performance of dedicated reasoning algorithms for conjunctive query answering is within LOGSPACE with respect to the number of assertions [MCGH+12];

- *OWL 2 RL* is designed to accommodate most of the features of OWL 2 and offers support for rule like expressions and axioms. 'The ontology consistency, class expression satisfiability, class expression subsumption, instance checking, and conjunctive query answering problems can be solved in time that is polynomial with respect to the size of the ontology.' [MCGH+12].

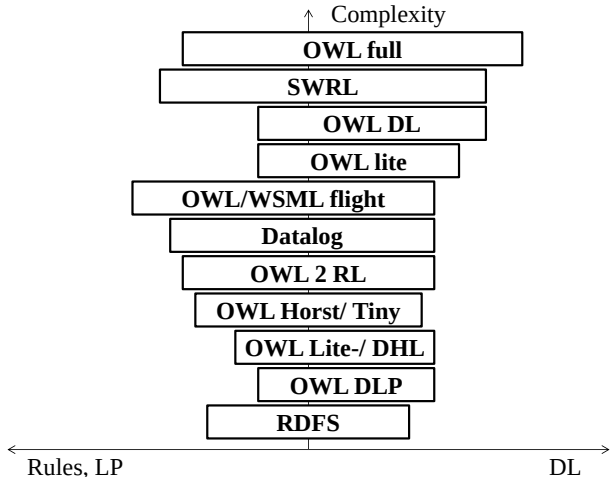


Figure A.4: Comparison of the expressibility of OWL dialects and alternative languages; LP - Linear Programming (adapted from Kiryakov and Damova [KD11]).

A.4 SPARQL Protocol and RDF Query Language - SPARQL

SPARQL allows to query and manipulate RDF data [PS17]. SPARQL queries are send by some client to a SPARQL endpoint storing the respective data set and the client receives in return the result of that query. The query language is much inspired by existing query languages such as SQL [ISO16], but is specifically adapted for the needs of SWT.
In Code A.6

Code A.6: Anatomy of a SPARQL query in SPARQL syntax [PS17].

```
PREFIX pre: <http://www.example.org/test#>
SELECT ?s
CONSTRUCT
INSERT
WHERE {
 ?s rdf:type owl:Class .
}
```

a skeleton SPARQL query is listed, which highlights some features of SPARQL. It is custom, but not mandatory, to capitalise SPARQL key words (e.g. `INSERT`, `CONSTRUCT`, `WHERE`). However, it is suggested to be consistent with only upper case or lower case letters and one of the conventions should be applied.

A query starts with the `PREFIX` section where for convenience namespaces can be prefixed such that queries are better readable by humans. When the retrieval of instances is needed the `SELECT` key word follows the prefixes and variables can be specified (e.g. `?s`). All individuals found, which comply to the within the `WHERE` clause specified graph pattern, are returned by the hosting triple store. The `CONSTRUCT` and `INSERT` key words instead of `SELECT` allow to specify a graph pattern, which is instantiated with the results of the executed query. The newly generated graph can be inserted into the existing knowledge base through the `INSERT` statement.

A comprehensive and thorough introduction into SPARQL, also including notions of query efficiency, is given in Du Charme [DuC15]. The topic of federated queries from different triple stores is treated in detail by Della Valle *et al.* [DVC11] and a discussion of triple stores as the infrastructure to host the semantic web is given in Kiryakov and Damova [KD11]. It should be noted that other possibilities to host RDF triples on the web are investigated [VSH+16].

It might be noted through its flexible and expressive features the SPARQL language is identified and used to specify knowledge additionally to, e.g. OWL. In particular, its ability to add newly generated triples in a sense of inferencing is important here. There exists attempts to integrate knowledge expressed as SPARQL queries in knowledge-bases [KK08, Knu11].

A.5 Assumptions in the Semantic Web: Unique Naming, Closed World, Open World

Some assumptions hold when dealing with SWT, which differ from classical data base technologies.

Unique Naming Assumption (UNA): In SWT such as OWL the unique naming assumption does not hold. On a webscale an individual could have multiple URIs referencing to it. Two re-

sources with different identifiers could be considered as the same individual unless the opposite is explicitly defined, .e.g through `owl:differentFrom`.
Open World Assumption (OWA): The open world assumption is related to the web scale of SWT. If a fact is not known by the knowledge base it does not follow that the fact is false. The reason for this is that the knowledge base might just does not have the respective information. However, it is possible to close down the OWA by adding statements, which explicitly state that a fact is false to the knowledge base. In engineering contexts the OWA of SWT might cause problems and might not be applicable [BS16b].

A.6 Ontology Engineering Methods

The following definition for ontology is adopted from Studer *et al.* [SBF98], which builds on an initial definition by Gruber [Gru93]:

Definition A.1 *'An ontology is a formal, explicit specification of a shared conceptualization.' [SBF98].*

Following this definition important characteristics of ontologies are, first, formality meaning that ontologies ought to have a formal syntax and semantics to be machine-interpretable; Second, explicit in the sense of that conclusion on the world can be drawn only if explicit concepts of the world are available; Finally, a shared conceptualisation, which requires a number of domain experts to reach a consensus, share and iteratively improve the found conceptualisation until some certain level of maturity is reached. By Definition A.1 ontologies qualify as a knowledge base when being combined with individuals of concepts [NM01]. Several methods are proposed for ontology development [SKN16] here the two most relevant are briefly summarised in the following two subsections.

A.6.1 Ontology Development 101

The ontology development 101 [NM01] provides guidance for domain experts to develop an ontology. The method is explained by developing an OWL ontology but is not restricted to this language. The authors emphasise, and it can be generalised to all modelling, that there does not exist a single correct way to model a domain but there are always alternatives. Additionally, iterations are a key component needed in ontology development as well as the developed concepts and relationships should be close to the real world artefacts to be modelled.
The method embodies a step by step approach towards developing an ontology. The terminology used in here is adapted to the current terms used for OWL ontologies: Concepts instead of classes, properties instead of slots, role restrictions instead of facets, individuals instead of instances [NM01]. The method proposes the following steps:

Step 1 is related to define the domain and scope of the intended ontology. It is quite helpful to the knowledge engineer to wisely choose the boundaries of the respective domain of interest. The definition of use cases is helpful to determine competency questions [GF95], which are supposed to be answered by the knowledge base implementing the intended ontology.
In *Step 2* the reuse of existing ontologies is stipulated. Besides reviewing literature related to the respective domain(s), tools exist[4] to help finding ontologies closely related to, or define similar concepts of the intended domain. Ontology reuse is particularly interesting as high level domain knowledge can be reused in different domains and is expressed using one ontology. Knowledge-based services built on top hence can be applied across domains.
Step 3 basically is about generating a vocabulary of terms, which should be supported by the intended ontology. It makes sense to describe propositional statements such as sentences about the respective domain to extract concepts, relationships and individuals.
From the concepts compiled in Step 3 a concept hierarchy is defined in *Step 4*. The definition of concepts and establishing a hierarchy among them should be defined carefully. Helpful suggestions can be found in Noy & McGuinness [NM01].
In *Step 5* using the remaining terms compiled in Step 3 relationships are defined among the derived concepts. Often the terms for relationships constitute predicates in sentences.
Step 6 involves the definition of role restrictions on the defined properties encoding additional domain knowledge. The possible definitions are dependent on the utilised modelling technology and for OWL they can include amongst others domain, range, value, existential, and universal restrictions [HKR+04].
Finally, in *Step 7* individuals of concepts in the concept hierarchy are defined. This may also be the case when the developed ontology is used in an application. Individuals are considered as assertional knowledge (see Section 2.2.4).
Again ontology development is an iterative process and it might be necessary to repeat singular steps in between as well as all steps until the final result is achieved. Checking the defined competency questions against the defined ontology provides a means for evaluation of the resulting ontology.

A.6.2 METHONTOLOGY

METHONTOLOGY is an ontology engineering method proposed by Fernandéz *et al.* [FLGPJ97]. It constitutes a step by step approach guiding through the respective steps of designing an ontology from scratch and a set of tools for the specification and documentation.
The initial activity in *(1) Specification* is the preparation of the Ontology Requirement Specification Document (ORSD). The ORSD includes in natural language the domain and scope of the intended ontology. The purpose should be clearly defined including potential use cases,

[4] `https://lov.okfn.org/`, Last accessed: 22 October 2018

and users and the level of formality of the document should be defined. The ORSD includes a collection of terms deemed relevant for the modelling of the respective domains. Also, sources of knowledge can be defined.
(2) Knowledge-Acquisition involves all activities of elicitating relevant knowledge from expert interviews, text books, standards, brainstorming, etc.
In the *(3) Conceptualisation* step the actual definition of concepts, concept hierarchies, properties etc. takes place. The method provides here a helpful set of tools, which support the formal definitions starting with a glossary of terms, concept classification trees and verb diagrams. A set of tables allow the strict definition and documentation of the design.
(4) Integration involves all activities and their documentation in determining suitable ontologies to be reused. The integration document lists the terms, which will be reused from existing ontologies.
In *(5) Implementation* the ontology is implemented using the respective tools for the chosen technology.
Within the *(6) Evaluation* the designed and implemented ontology is evaluated against the original ORSD document holding the information requirements of the intended ontology as well as use cases for the evaluation.
The documents and tables generated throughout executing the method provide proper means for the *(7) Documentation* of the design decisions. They need to be gathered and appropriately stored for future iterations and to enable and stipulate the reuse of the designed ontology.

A.7 Tools for Developing SWT applications

A range of free and commercial tools exist for the rapid implementation of ontologies and KBSs relying on SWT technologies. The most famous ontology engineering tool is the Protégé ontology editor[5] [Mus15] and another free editor is the NeOn Toolkit[6], which is not developed further any more as of writing. A commercial tool and suite for the engineering of ontologies is the TopBraid Composer and associated tools by TopQuadrant[7].
A number off-the-shelf performant OWL-DL reasoners exist. A main advantage of these is that most of them directly integrate or are shipped with the Protégé ontology editor, e.g. HermiT [GHM+14] and Pellet [SPG+07]. The performance for specific problems is compared in the work of Dentler *et al.* [DCTTDK11]. For hosting large quantities of triple data specific data bases exist which offer a variety of capabilities (see review in Luyen *et al.* [LTV+16]).

[5] `http://protege.stanford.edu`, Last accessed: 22 October 2018
[6] `http://neon-toolkit.org/index.html`, Last accessed: 22 October 2018
[7] `https://www.topquadrant.com/`, Last accessed: 22 October 2018

B Supplementary Ontology Material

B.1 Specification of the Basic Datatype Ontology

The following tables describe concepts and individuals defined in the `BasicDatatypeOntology`. The ontology is bootstrapped from the XSD specification [BM04]. For each basic data type specified in the XSD specification a concept (Table B.1) and an individual is created (Table B.2). The concept name is derived from capitalising the original XSD data type and the individual name from converting all characters to lower case.

Table B.1: Concepts defined in the `BasicDatatypeOntology`.

Concept	**Super-Concept (`rdfs:subClassOf`)**	**Corresponding XSD datatype (`rdfs:seeAlso`)**
Anytype	-	anyType
Complextype	Anytype	complexType
Anysimpletype	Anytype	anySimpleType
Anyuri	Anysimpletype	anyURI
Base64binary	Anysimpletype	base64Binary
Boolean	Anysimpletype	boolean
Byte	Short	byte
Date	Anysimpletype	date
Datetime	Anysimpletype	dateTime
Decimal	Anysimpletype	decimal
Double	Anysimpletype	double
Duration	Anysimpletype	duration
Entities	Entity	ENTITIES
Entity	Ncname	ENTITY
Float	Anysimpletype	float

Gday	Anysimpletype	gDay
Gmonth	Anysimpletype	gMonth
Gmonthday	Anysimpletype	gMonthDay
Gyear	Anysimpletype	gYear
Gyearmonth	Anysimpletype	gYearMonth
Hexbinary	Anysimpletype	hexBinary
Id	Ncname	ID
Idref	Ncname	IDREF
Idrefs	Idref	IDREFS
Int	Long	int
Integer	Decimal	integer
Language	Token	language
Long	Integer	long
Name	Token	Name
Ncname	Name	NCName
Negativeinteger	Nonpositiveinteger	negativeInteger
Nmtoken	Token	NMTOKEN
Nmtokens	Nmtoken	NMTOKENS
Nonnegativeinteger	Integer	nonNegativeInteger
Nonpositiveinteger	Integer	nonPositiveInteger
Normalizedstring	String	normalizedString
Notation	Anysimpletype	NOTATION
Positiveinteger	Nonnegativeinteger	positiveInteger
Qname	Anysimpletype	QName
Short	Int	short
String	Anysimpletype	string
Time	Anysimpletype	time
Token	Normalizedstring	token
Unsignedbyte	Unsignedshort	unsignedByte
Unsignedint	Unsignedlong	unsignedInt
Unsignedlong	Nonnegativeinteger	unsignedLong
Unsignedshort	Unsignedint	unsignedShort

Table B.2: Individuals defined in the `BasicDatatypeOntology`.

Individual	**Concept (`rdf:type`)**	**Corresponding XSD datatype (`rdfs:seeAlso`)**
anytype	Anytype	anyType
complextype	Complextype	complexType
anysimpletype	Anysimpletype	anySimpleType
anyuri	Anyuri	anyURI
base64binary	Base64binary	base64Binary
boolean	Boolean	boolean
byte	Byte	byte
date	Date	date
datetime	Datetime	dateTime
decimal	Decimal	decimal
double	Double	double
duration	Duration	duration
entities	Entities	ENTITIES
entity	Entity	ENTITY
float	Float	float
gday	Gday	gDay
gmonth	Gmonth	gMonth
gmonthday	Gmonthday	gMonthDay
gyear	Gyear	gYear
gyearmonth	Gyearmonth	gYearMonth
hexbinary	Hexbinary	hexBinary
id	Id	ID
idref	Idref	IDREF
idrefs	Idrefs	IDREFS
int	Int	int
integer	Integer	integer
language	Language	language
long	Long	long
name	Name	Name
ncname	Ncname	NCName
negativeinteger	Negativeinteger	negativeInteger
nmtoken	Nmtoken	NMTOKEN
nmtokens	Nmtokens	NMTOKENS
nonnegativeinteger	Nonnegativeinteger	nonNegativeInteger
nonpositiveinteger	Nonpositiveinteger	nonPositiveInteger
normalizedstring	Normalizedstring	normalizedString

notation	Notation	NOTATION
positiveinteger	Positiveinteger	positiveInteger
qname	Qname	QName
short	Short	short
string	String	string
time	Time	time
token	Token	token
unsignedbyte	Unsignedbyte	unsignedByte
unsignedint	Unsignedint	unsignedInt
unsignedlong	Unsignedlong	unsignedLong
unsignedshort	Unsignedshort	unsignedShort

B.2 Additional Visualisation

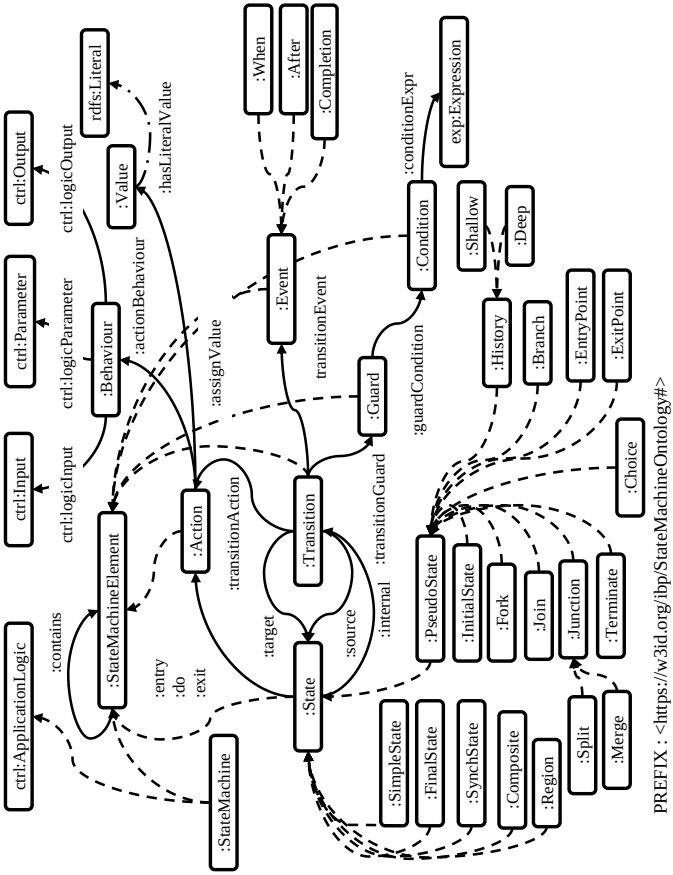


Figure B.1: Full visualisation of the concepts and relationships of the `StateMachineOntology` [SPS17, SWO19].

C Implementation

Within this chapter implementation specific code listings of the use cases presented in Chapter 5 are denoted.

C.1 Implementation of SPARQL Queries for the Automated Rule-Based Verification of State Graphs and Schedules

Here, SPARQL queries needed for the implementation of the knowledge-based method presented in Section 5.1 are denoted for reference. The archetype query listed in Code C.1 formalises the rule-like knowledge to verify the correct behaviour of state graphs according to VDI 3814-6 as defined in Definition 5.1.

Code C.1: Parameterised SPARQL query to filter points in time with faulty state graph behaviour [SPS17]. Placeholders enclosed by $$ are to be replaced for configuration, for instance by the results presented in Table C.1 .

```
PREFIX [...]
SELECT ?pk ?timeValue
WHERE {
 ?pk a tb:PrimaryKey .
 ?pk tb:time ?timeValue .
 ?pk tb:$vDPMeas$  ?yOutpValue .
 ?pk tb:$vDPState$ ?yStateValue .
 FILTER( ?yStateValue = "1.0" &&
       ?yOutpValue != "$vValue$" ) . }
ORDER BY ?timeValue
```

Here, a straightforward mapping mechanism is applied for the time series data. The time series is organised in columns and each column name is used as a data type property to map to the value of every entry, e.g. `tb:time`. An arbitrary prefix `tb:` is used.
The query listed in Code C.2 retrieves the required knowledge to configure the archetype query listed in Code C.1. Example results are given in Table C.1 for state `Off` of the state graph illustrated in Figure 5.2.

Code C.2: Query to retrieve necessary state graph description for the configuration of the parametrised query listed in Code C.1 [SPS17].

```
PREFIX [...]
SELECT ?vState ?vDPState ?vDPMeas ?vValue
WHERE {
  ?vSG  rdf:type SG:StateGraph ;
        SG:contains vState .
  ?vState   rdf:type SG:State .
  ?vState   SG:stateAction ?vAction .
  ?vAction  rdf:type SG:Action .
  ?vAction  SG:assignValue ?vValue .
  ?vAction  ctrl:logicOutput ?vOutput .
  ?vOutput  rdf:type ctrl:Output .
  ?vOutput  seas:connectsSystemThrough ?vConn .
  ?vDPMeas  seas:connectsSystemThrough ?vConn .
  ?vDPMeas  rdf:type seas:Communication ConnectionPoint .
  ?vState   ctrl:logicOutput ?vOutputS .
  ?vOutputS rdf:type ctrl:Output .
  ?vOutputS seas:connectsSystemThrough ?vConnS .
  ?vDPState seas:connectsSystemThrough ?vConnS .
  ?vDPState rdf:type seas:CommunicationConnectionPoint .
}
```

Table C.1: Example results retrieved from knowledge base for state `Off` by query listed in Code C.2. See Section 5.1 and Figure 5.2.

vState	vDPState	vDPMeas	vValue
:StateOff	:r1.SG_StateOff.active	:r1.YMix	'0'
:StateOff	:r1.SG_StateOff.active	:r1.YFan	'0'
:StateOff	:r1.SG_StateOff.active	:r1.YPump	'0'

The archetype query listed in Code C.3 formalises the rule-like knowledge to verify the correct behaviour of schedules as defined in Definition 5.2.

Code C.3: Parametrised SPARQL query to identify points in time with faulty schedule behaviour.

```
PREFIX [...]
SELECT ?pk ?timeValue
WHERE {
  ?pk a tb:PrimaryKey .
  ?pk tb:$DPTime$   ?timeValue .
  ?pk tb:$vDPMeas$  ?yOutpValue .
  FILTER( "$xValStart$" < ?timeValue && ?timeValue < "
      $xValEnd$" && ?yOutpValue != "$valPoly$" ) .
```

```
}
ORDER BY ?timeValue
```

To retrieve the necessary knowledge for configuring the archetype query listed in Code C.3 the query listed in Code C.4 can be used.

Code C.4: Query to retrieve necessary description of a schedule for the configuration of the parametrised query listed in Code C.3.

```
PREFIX [...]
SELECT ?vSch ?vPer ?vDPTime ?vInt ?vDPMeas ?xValStart ?xValEnd
     ?vPolyDeg ?vPolyVal
WHERE{
  ?vSch rdf:type sch:Schedule .
  ?vSch ctrl:logicInput ?vIpt .
  ?vIpt rdf:type ctrl:Input .
  ?vIpt seas:connectsSystemThrough      ?vConnIpt .
  ?vDPTime  seas:connectsSystemThrough  ?vConnIpt .
  ?vDPTime rdf:type seas:CommunicationConnectionPoint .
  ?vIpt rdf:type ctrl:Input .
  ?vSch ctrl:logicOutput ?vOtp .
  ?vOtp rdf:type ctrl:Output .
  ?vOtp seas:connectsSystemThrough      ?vConnOtp .
  ?vDPMeas  seas:connectsSystemThrough  ?vConnOtp .
  ?vDPMeas rdf:type seas:CommunicationConnectionPoint .
  ?vSch sch:hasPeriodicity ?vPer .
  ?vSch sch:hasInterval ?vInt .
  # Fetch start and end value of each interval
  ?vInt sch:hasStartPoint ?vSP .
  ?vSP sch:hasXCoordinate ?vXCooStart .
  ?vXCooStart sch:hasLiteralValue ?xValStart .
  ?vInt sch:hasEndPoint ?vEP .
  ?vEP sch:hasXCoordinate ?vXCooEnd .
  ?vXCooEnd sch:hasLiteralValue ?xValEnd .
  # Fetch polynomial constants and degree
  ?vInt sch:hasMathematicalFunction ?vMF .
  ?vMF sch:hasPolynomialConstant ?vPolyCon .
  ?vPolyCon sch:polyConstantLiteralValue ?vPolyVal .
  ?vPolyCon sch:polyConstantDegree ?vPolyDeg .
}
```

For convience the configured queries can be aggregated into one query, which is executed against the knowledge stored in a triple store. For instance, Code C.5 lists a query where configured instances of the queries in Code C.1 and Code C.3 are aggregated to be executed on a triple store.

Code C.5: Aggregated query holding instances of configured queries to evaluate monitoring data [SPS17].

```
PREFIX [...]
```

```
SELECT ?pk ?timeValue
WHERE {
        {# Configured query 1}
UNION
        {# Configured query 2}
UNION
        {# ...}
}
```

C.2 Implementation of SPARQL queries for the Knowledge-Enhanced Engineering of Control Logic in Automation Systems

C.2.1 Implementation of SPARQL queries for the Verification of Control Logic

Code C.6: SPARQL query checking if more then one transition leaves an initial state.

```
PREFIX [...]
INSERT {
    [] br:initialStateCardinalityMismatch ?initialState .

}
WHERE {
  { SELECT ?initialState
    WHERE {
        ?IniState rdf:type sm:InitialState .
        ?tran rdf:type sm:Transition .
        ?tran sm:source ?initialState .
    }
    GROUP BY ?initialState
    HAVING( COUNT( DISTINCT ?tran ) != 1 )
  }
}
```

Code C.7: SPARQL query checking for faulty transitions outgoing from forks with guards.

```
PREFIX [...]
INSERT {
  [] br:guardForkOutgoing ?tran .
}
WHERE {
  ?fork a sm:Fork .
  ?tran a sm:Transition .
  ?tran sm:source ?fork .
```

```
  ?tran sm:transitionGuard ?gua .
}
```

C.2.2 Implementation of SPARQL queries for the Simultaneous Verification of Different Control Logic Types

Code C.8: SPARQL query which can be used to verify if the basic data type of a value assigned to an output complies to the basic data type of a parameter compared to this value in another control actor.

```
INSERT {
  [] br:dataTypeMismatch ?par , ?ActVal .
}
WHERE {
  ?ActVal sm:hasLiteralValue ?LitVal .
  ?ActBehav sm:assignValue ?ActVal .
  ?ActBehav ctrl:logicOutput ?outp .
  ?outp ctrl:isConnectedTo ?inp .
  ?var ctrl:logicInput ?Inp .
  ?var ll:nextElement/ll:nextElement ?ele .
  ?ele ctrl:logicParameter ?par .
  ?par ctrl:hasDataType ?ParDataType .
  ?ParDataType rdfs:seeAlso ?derivedDataType .
  BIND( DATATYPE( ?LitVal ) AS ?LitValDT )
  FILTER( ?LitValDT != ?derivedDataType )
}
```

Code C.9: SPARQL query checking if compared parameters and inputs used in expressions match with regard to their quantity.

```
PREFIX [...]
INSERT {
  [] br:quantityMismatch ?ele1mis ;
     br:quantityMismatch ?ele2mis .
}
WHERE {
  ?a1 ctrl:hasQuantity ?q1 .
  ?a2 ctrl:hasQuantity ?q2 .

  ?ele1 ctrl:logicInput | ctrl:logicParameter ?a1 .
  ?ele2 ctrl:logicInput | ctrl:logicParameter ?a2 .

  ?ele1 ll:nextElement ?ope .
  ?ope ll:nextElement ?ele2 .
  ?ope math:operator ?sym .
```

```
  FILTER( strStarts( STR( ?sym ) , STR( REL: ) ) )

  BIND(( IF( !sameTerm( ?q1 , ?q2 ) , ?ele1 , "" )) AS ?
      ele1mis )
  BIND(( IF( !sameTerm( ?q2 , ?q1 ) , ?ele2 , "" )) AS ?
      ele2mis )
}
```

C.2.3 Code examples for the Verification of Control Logic Designs and Plant Data

Code C.10: Triples specifying the measurement range of temperature sensor ABB model SensyTemp TSP121 [ABB18] using turtle syntax [BBLPC14].

```
@prefix [...] .
@prefix : <https://w3id.org/ibp/test#>

:B6-T-Sensor a owl:NamedIndividual , ssn:System , sosa:Sensor
    , OC-mea:T-Sensor ;
  rdfs:comment "Sensor in tank B6."@en ;
  rdfs:seeAlso <http://bit.ly/2Eln615> ;
  ssns:hasOperatingRange :B6-T-OpRa ;
  ssns:hasSystemCapability :B6-T-Capa ;
  sosa:observes :B6-T ;
  seas:connectsAt :TIS602-CP .

:B6-T-OpRa rdf:type owl:NamedIndividual , ssns:OperatingRange
    ;
  rdfs:comment "The conditions in which the B6 temperature
      sensor is allowed to operate. Table 3.1 in technical
      data sheet, option plastic."@en ;
  ssns:inCondition [ a ssns:Condition ,
    schema:PropertyValue ;
    schema:minValue -40.0 ;
    schema:maxValue 120.0 ;
    schema:unitCode om:degreeCelsius ] .

:B6-T-Capa a owl:NamedIndividual , ssns:SystemCapability ;
  rdfs:comment "The measurement range of ABB SensyTemp TSP121.
       Tab 12.2 in technical data sheet option A6"@en ;
  ssns:hasSystemProperty [ a ssns:MeasurementRange ,
    schema:PropertyValue ;
    schema:minValue 0.0 ;
    schema:maxValue 120.0 ;
    schema:unitCode om:degreeCelsius ] .

# Connection
:B6-T rdf:type owl:NamedIndividual , sosa:ObservableProperty .
```

```
:TIS602-CP rdf:type owl:NamedIndividual , seas:
    WireCommunicationConnectionPoint ;
  dcterms:identifier "TIS602" ;
  seas:connectsSystemsThrough :TIS602-Conn .

:TIS602-Conn rdf:type owl:NamedIndividual , seas:
    WireCommunicationConnection .

# Control actor description
:fB_StateMachine.TIS602 rdf:type owl:NamedIndividual , ctrl:
    Input , seas:CommunicationConnectionPoint ;
  ctrl:hasDataType BDO:float ;
  seas:connectionPointOf :fB_StateMachine ;
  dcterms:identifier "fB_StateMachine.TIS602" ;
  seas:connectsSystemsThrough :TIS602-Conn .

# Control logic with parameter
:test-Expr rdf:type exp:Expression , owl:NamedIndividual ;
  mer:isComposedOf :ele0 , :ele1 , :ele2 .

:ele0 rdf:type ll:FirstElement .
:ele0 ll:nextElement :ele1 .
:ele1 ll:nextElement :ele2 .

:ele0 rdf:type owl:NamedIndividual , exp:ExpressionElement ,
    exp:Operand , math:Variable ;
  math:name "TIS602"^^xsd:string ;
  ctrl:logicInput :fB_StateMachine.TIS602 .

:ele1 rdf:type owl:NamedIndividual , exp:ExpressionElement ,
    exp:Operator ;
  math:operator REL:leq .

:ele2 rdf:type owl:NamedIndividual , exp:ExpressionElement ,
    exp:Operand , math:Literal ;
  math:value "298.0"^^xsd:decimal ;
  ctrl:logicParameter :ele2-Par .
```

Code C.11: SPARQL query identifying a mismatch between the upper bound of the measurement range of a sensor and a parameter value which is compared to the sensor output.

```
PREFIX [...]
INSERT {
  ?ele br:parameterValueLargerThanMeasurementRange ?sens .
}
WHERE {
  ?sens a ssn:System .
  ?sens ssns:hasSystemCapability/ssns:hasSystemProperty ?MeaRa
      .
  ?MeaRa a ssns:MeasurementRange .
```

```
  ?MeaRa schema:maxValue ?max .

  ?sens seas:connectsAt/seas:connectsSystemsThrough ?conn .
  ?var ctrl:logicInput/seas:connectsSystemsThrough ?conn .
  ?var ll:nextElement ?Ope .
  ?Ope ll:nextElement ?ele .
  ?Ope math:operator ?sym .
  FILTER( strStarts( STR( ?sym ) , STR( REL: ) ) )
  ?ele math:value ?value .
  FILTER( ?max < ?value )
}
```

C.2.4 Code examples Demonstrating the Bidirectional Exchange and Incremental Verification

Code C.12: Source code of pump safety feature in ST syntax [IEC14a].

```
FUNCTION_BLOCK FB_PUM901_Safety
VAR_INPUT
  PIS901 : LREAL := 0.0 ;
  bPUM901 : BOOL := FALSE ;
END_VAR
VAR_OUTPUT
  bPUM901_SAFE : BOOL := FALSE ;
END_VAR
  bPUM901_SAFE := bPUM901 = TRUE AND NOT ( PIS901 > 200000 ) ;
END_FUNCTION_BLOCK
```

Code C.13: Triples representing pump safety feature in turtle syntax [BBLPC14].

```

:FB_PUM901_Safety rdf:type owl:NamedIndividual , ctrl:
    ControlActor ;
  ctrl:hasApplicationLogic :FB_PUM901_Safety-ApplicationLogic
      ;
  ctrl:hasInput :FB_PUM901_Safety-PIS901 ,
    :FB_PUM901_Safety-bPUM901 ;
  ctrl:hasOutput :FB_PUM901_Safety-bPUM901_SAFE ;
  ctrl:hasParameter :FB_PUM901_Safety-ExpEle-2-Par ,
    :FB_PUM901_Safety-ExpEle-7-Par .

:FB_PUM901_Safety-PIS901 rdf:type owl:NamedIndividual , ctrl:
    Input ;
  ctrl:hasUnit om:pascal ;
  ctrl:hasQuantity om:Pressure ;
  ctrl:hasDataType BDO:double .
```

```
:FB_PUM901_Safety-bPUM901 rdf:type owl:NamedIndividual , ctrl:
    Input ;
  ctrl:hasBasicType BDO:boolean .

:FB_PUM901_Safety-bPUM901_SAFE rdf:type owl:NamedIndividual ,
    ctrl:Output ;
  ctrl:hasBasicType BDO:boolean .

:FB_PUM901_Safety-ApplicationLogic rdf:type owl:
    NamedIndividual , tfe:MISO ;
  tfe:hasExpression :FB_PUM901_Safety-Expr .

:FB_PUM901_Safety-ExpEle-0 rdf:type owl:NamedIndividual , ll:
    FirstElement , math:Variable , exp:ExpressionElement , exp
    :Operand ;
  ll:nextElement :FB_PUM901_Safety-ExpEle-1 ;
  math:name "bPUM901"^^xsd:string ;
  ctrl:logicInput :FB_PUM901_Safety-bPUM901 .

:FB_PUM901_Safety-ExpEle-1 rdf:type owl:NamedIndividual , exp:
    ExpressionElement , exp:Operator ;
  ll:nextElement :FB_PUM901_Safety-ExpEle-2 ;
  math:operator REL:eq .

:FB_PUM901_Safety-ExpEle-2 rdf:type owl:NamedIndividual , math
    :Literal , exp:ExpressionElement , exp:Operand ;
  ll:nextElement :FB_PUM901_Safety-ExpEle-3 ;
  math:value "True" ;
  ctrl:logicParameter :FB_PUM901_Safety-ExpEle-2-Par .

:FB_PUM901_Safety-ExpEle-2-Par rdf:type owl:NamedIndividual ,
    ctrl:Parameter .

:FB_PUM901_Safety-ExpEle-3 rdf:type owl:NamedIndividual , exp:
    ExpressionElement , exp:Operator ;
  ll:nextElement :FB_PUM901_Safety-ExpEle-4 ;
  math:operator LOG:and .

:FB_PUM901_Safety-ExpEle-4 rdf:type owl:NamedIndividual , exp:
    ExpressionElement , exp:Operator ;
  ll:nextElement :FB_PUM901_Safety-ExpEle-5 ;
  math:operator LOG:not .

:FB_PUM901_Safety-ExpEle-5 rdf:type owl:NamedIndividual , math
    :Variable , exp:ExpressionElement , exp:Operand ;
  ll:nextElement :FB_PUM901_Safety-ExpEle-6 ;
  math:name "PIS901"^^xsd:string ;
  ctrl:logicInput :FB_PUM901_Safety-PIS901 .

:FB_PUM901_Safety-ExpEle-6 rdf:type owl:NamedIndividual , exp:
    ExpressionElement , exp:Operator ;
```

```
  ll:nextElement :FB_PUM901_Safety-ExpEle-7 ;
  math:operator REL:gt .

:FB_PUM901_Safety-ExpEle-7 rdf:type owl:NamedIndividual , math
    :Literal , exp:ExpressionElement , exp:Operand ;
  math:value "200000" ;
  ctrl:logicParameter :FB_PUM901_Safety-ExpEle-7-Par .

:FB_PUM901_Safety-ExpEle-7-Par rdf:type owl:NamedIndividual ,
    ctrl:Parameter ;
  ctrl:hasUnit om:pascal ;
  ctrl:hasQuantity om:Pressure ;
  ctrl:hasBasicType BDO:double .


:FB_PUM901_Safety-Expr rdf:type owl:NamedIndividual , exp:
    Expression ;
   mereology:isComposedOf :FB_PUM901_Safety-ExpEle-0 ,
    :FB_PUM901_Safety-ExpEle-1 ,
    :FB_PUM901_Safety-ExpEle-2 ,
    :FB_PUM901_Safety-ExpEle-3 ,
    :FB_PUM901_Safety-ExpEle-4 ,
    :FB_PUM901_Safety-ExpEle-5 ,
    :FB_PUM901_Safety-ExpEle-6 ,
    :FB_PUM901_Safety-ExpEle-7 .
```

Code C.14: Source code of pump safety feature in Modelica [Mod17b] syntax.

```
block FB_PUM901_Safety
  Modelica.Blocks.Interfaces.RealInput PIS901 ;
        Modelica.Blocks.Interfaces.BooleanInput bPUM901 ;
  Modelica.Blocks.Interfaces.BooleanOutput bPUM901_SAFE ;
algorithm
  bPUM901_SAFE := bPUM901 = TRUE AND NOT ( PIS901 > 200000 ) ;
end FB_PUM901_Safety ;
```

D Namespaces

The namespaces utilised in texts and illustrations of this work are denoted in this Chapter. Table D.1 presents prefixes used, which are recommendations of the W3C. Table D.2 presents prefixes from reused ontologies. Table D.3 presents prefixes used, which are defined by the author for ontologies representing terminological knowledge and Table D.4 for assertional knowledge.

Table D.1: Prefixes and namespaces of W3C recommendations used in this work.

Prefix	URI
rdf	http://www.w3.org/1999/02/22-rdf-syntax-ns#
rdfs	http://www.w3.org/2000/01/rdf-schema#
xsd	http://www.w3.org/2001/XMLSchema#
owl	http://www.w3.org/2002/07/owl#
sosa	http://www.w3.org/ns/sosa/
ssn	http://www.w3.org/ns/ssn/
ssns	http://www.w3.org/ns/ssn/systems/

Table D.2: Prefixes and namespaces of reused ontologies.

Prefix	URI
om	http://www.ontology-of-units-of-measure.org/resource/om-2/
schema	http://schema.org/
hto	http://vcharpenay.github.io/hto/hto.xml#
seas	https://w3id.org/seas/
ifc	http://www.buildingsmart-tech.org/ifcOWL/IFC4_ADD2#

math	http://numerateweb.org/vocab/math#
REL	http://www.openmath.org/cd/relation1#
ARI	http://www.openmath.org/cd/arith1#
LOG	http://www.openmath.org/cd/logic1#
mer	file:/C:/OntoCAPE/meta_model/mereology/mereology.owl#
ll	file:/C:/OntoCAPE/meta_model/data_structures/linked_list.owl#
OC-mea	file:/C:/OntoCAPE/OntoCAPE/chemical_process_system/CPS_realization/process_control_equipment/measuring_instrument.owl#
OC-equ	file:/C:/OntoCAPE/chemical_process_system/CPS_realization/plant_equipment/apparatus.owl#

Table D.3: Prefixes and namespaces of ontologies describing terminological knowledge of a domain (TBox) defined and used in this thesis.

Prefix	**URI**
ctrl	https://w3id.org/ibp/CTRLont#
sm	https://w3id.org/ibp/StateMachineOntology#
SG	https://w3id.org/ibp/StateGraphOntology#
sch	https://w3id.org/ibp/ScheduleOntology#
seq	https://w3id.org/ibp/SequenceControlOntology#
tpd	https://w3id.org/ibp/TwoPointDiscreteOntology#
exp	https://w3id.org/ibp/ExpressionOntology#
BDO	https://w3id.org/ibp/BasicDatatypeOntology#
tfe	https://w3id.org/ibp/TransferFunctionElementOntology#
tb	https://w3id.org/ibp/example#
br	https://w3id.org/ibp/BadRelationsOntology#

Table D.4: Namespaces of ontologies describing assertional knowledge of examples (ABox) defined and used in this thesis.

Figure	URI
Figure 4.5	https://w3id.org/ibp/CTRLontEval#
Figure 4.8	https://w3id.org/ibp/ExpressionOntologyEval#
Figure 4.11	https://w3id.org/ibp/ScheduleOntologyEval#
Figure 4.14	https://w3id.org/ibp/SequenceControlOntologyEval#
Figure 4.17	https://w3id.org/ibp/TwoPointDiscreteOntologyEval#
Figure 4.20	https://w3id.org/ibp/TransferFunctionElementOntEval#
Figure 4.24b	https://w3id.org/ibp/StateMachineOntologyEval#
Figure 4.27	https://w3id.org/ibp/StateGraphOntologyEval#
Figure 5.3	https://w3id.org/ibp/InstancesAutoVerification#
Figure 5.13	https://w3id.org/ibp/InstancesSensorExerpt#
Figure 5.15	https://w3id.org/ibp/InstancesTwoTranFromIni#
Figure 5.16	https://w3id.org/ibp/InstancesGuardFromFork#
Figure 5.17	https://w3id.org/ibp/InstancesQuantityMismatch#
Figure 5.18	https://w3id.org/ibp/InstancesAcrossControl#
Figure 5.19	https://w3id.org/ibp/InstancesPlantData#

List of Figures

List of Tables

List of Code Listings

List of Acronyms

Notation	Description
δD	Deuterium
`CTRLont`	Control Ontology [SPS17]
AEC/FM	Architecture, Engineering, Construction and Facility Management
AHU	Air Handling Unit
AI	Artificial Intelligence
AutomOnto	Automation Ontology [NSMŠ15]
BACnet	Building Automation and Control Networks, network protocol [ISO17a]
BAS	Building Automation Systems
BCK	Building Control Knowledge [CTM16]
BDO	Basic Datatype Ontology, see Section B.1
BIM	Building Information Modeling [ETSL11]
CAEX	Computer Aided Engineering Exchange [FD04]
CASCADE	CASCADE ICT for Energy Efficient Airports a research project funded by the European Commission under grant agreement No. 284920
CCO	Common Concepts Ontology (`http://data.ifs.tuwien.ac.at/engineering/cco`, Last accessed: 15 June 2018)
CDL	Controls Description Language [Wet18]
CH_4	Methane
CNC	Computer Numerical Control
CO_2	Carbon dioxide
CPS	Cyber-Physical Systems
CSV	Comma Separated Value
DEG	Datenmodell- und Austauschformat für das Engineering in der Gebäudeautomation, (eng.: Data model and exchange format for the engineering of BAS) [Run11]
DL	Description Logics

Notation	Description
DogOnt	Domotic Ontology [BC08]
EKB	Engineering Knowledge Bus [MB12]
EnCN	Energie Campus Nürnberg
ERP	Enterprise Resource Planning
ESIM	Energy System Information Model [KS15]
ETSI	European Telecommunications Standards Institute
FDD	Fault Detection and Diagnosis [KB05]
FOL	First Order Predicate Logic, e.g. [DFH11]
GRAFCET	GRAphe Fonctionnel de Commande Etapes/Transitions [IEC13c]
HTO	Haystack Tagging Ontology [CKAK15]
HTTP	Hyper Text Transfer Protocol
HVAC	Heating Ventilation and Air Conditioning
I/O	Input/ Output
IAS	Industrial Automation Systems
IBP	Fraunhofer Institute for Building Physics
ICT	Information and Communication Technologies
IDE	Integrated Development Environment
IEC	International Electrotechnical Commission
IEEE	Institute of Electrical and Electronics Engineers
IFC	Industry Foundation Classes [ISO13]
IMI	Institute for Information Management in Engineering
IML	Intermediate Modeling Layer [LEHM11]
IoT	Internet of Things
ISO	International Standards Organization
KBS	Knowledge-Based System
KE	Knowledge Engineering
KIT	Karlsruhe Institute of Technology
KNX	KNX; Network protocol in building automation [ISO07]
KR	Knowledge Representation
LON	Local Operating Network; Network protocol in building automation [ISO12]

Notation	Description
MES	Manufacturing Execution System
MoML	Modeling Markup Language, [LN00]
N_2O	Nitrous oxide
OntoCAPE	Ontology for Computer-Aided Process Engineering, [MMWY09]
OOM	Object-Oriented Modelling, [DR15, RQd12]
OSI	Open Systems Interconnection
OSS	Accumulated Weighted Operational Quality, [Ple13]
OWL	Web Ontology Language Version 2, [W3C12]
PEP	Procedure Execution Ontology, [LKGZ17]
PI&D	Pipes and Instrumentation Diagram
PID	Proportional-Integral-Derivative, [Abe10]
PLC	Programmable Logic Controller
POU	Program Organizational Unit [IEC14a]
RDF	Resource Description Framework [CWL14]
RDFS	Resource Description Framework Schema [BG14]
SAREF	Smart Appliances REFerence Ontology [ETS15]
SCADA	Supervisory Control and Data Acquisition
SEAS	Smart Energy Aware Systems [LKGZ17]
SFC	Sequential Function Chart [IEC14a]
SOA	State Of the Art
SOSA	Semantic Sensor Network Ontology [HKC+17]
SPARQL	SPARQL Protocol and RDF Query Language [PS17]
SPIN	SPARQL Inferencing Notation [Knu11]
ST	Structured Text [IEC14a]
SWRL	Semantic Web Rule Language [HPSB+04]
SWT	Semantic Web Technologies [BLHL01, DFH11]
UML	Unified Modeling Language [Obj15a]
UML AP	UML Automation Profile [HK13]
UNA	Unique Naming Assumption
URI	Unique Resource Identifier

Notation	Description
VDI	Verein Deutscher Ingenieure (Association of German Engineers)
W3C	World Wide Web Consortium
XMI	XML Metadata Interchange [Obj15b]
XML	eXtensible Markup Language [BPSM+06]
XSD	XML Schema Definition [BM04]