FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

An Ontology as Shared Vocabulary for Distributed Intelligence in Smart Homes

DISSERTATION

submitted in partial fulfillment of the requirements for the degree of

Doktor der technischen Wissenschaften

by

Dipl.-Ing. Mario Jerome Kofler
Registration Number 0025206

to the Faculty of Informatics
at the Vienna University of Technology

Advisor: Ao. Univ.Prof. Dipl.-Ing. Dr. Wolfgang Kastner

The dissertation has been reviewed by:

(Ao. Univ.Prof. Dipl.-Ing. Dr. Wolfgang Kastner)	(Ao. Univ.Prof. Dipl.-Arch. Dr. Georg Suter)

Wien, 30.01.2014

(Dipl.-Ing. Mario Jerome Kofler)

Erklärung zur Verfassung der Arbeit

Dipl.-Ing. Mario Jerome Kofler
Hasnerstraße 142/13, A-1160 Wien

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit - einschließlich Tabellen, Karten und Abbildungen -, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

(Ort, Datum)	(Unterschrift Verfasser)

Abstract

Automation of buildings in the commercial but also the residential sector is on the rise, not least because of advanced opportunities developing from the integration of a multitude of devices into a single control system. While home automation thereby promises improved personal comfort and the optimized operation of the so-called smart home, these goals are currently only partly reached with regard to energy consumption. Mostly, configuration and monitoring of the building are in the responsibility of the home owners often leaving them overwhelmed with the amount and variety of influencing parameters and control options.
One solution is to hand over control of a residential building to an autonomous system. For such a system, an unambiguous view of the environment is indispensable. In this regard, one major challenge is to address yet unconsidered factors that can be taken into account for the optimized control of a dwelling. The integration of these additional parameters into building control strategies allows the realization of advanced use cases such as proactive control that consider the current and future state of the building. In this context, highly heterogeneous domains need to be combined to reach an accurate representation of the building and its surroundings which is mandatory when an intelligent and optimized operation is desired.
This dissertation presents an ontology that comprehensively models the smart home environment in order to provide the foundation for autonomous control of user comfort and energy efficiency. Using the Web Ontology Language (OWL) as knowledge representation formalism thereby facilitates the semantic description of terms and their interconnections. In this respect, one of the main problems addressed in this work is the abstraction from reality while still providing all major influencing factors to a building control system. It is shown how capabilities of OWL like subtle reasoning can be optimally utilized when creating a knowledge base for a smart home. The designed model is furthermore evaluated against competency questions retrieved from a set of advanced smart home use cases to demonstrate its suitability for this domain. One of the major benefits of the resulting ontology is the formal description of a large variety of parameters which is accessible through a well-defined interface yet at the same time independent from the actual implementation of the control mechanism. Although the focus is the residential sector, the created conceptualization is to a great extent also useful for commercial buildings. While the application of the ontology is demonstrated for an individual building, as a shared vocabulary it may also be the basis for the integration of multiple building control systems in the context of a smart city by providing a unified view of the world.

Kurzfassung

Gebäudeautomation im Zweck- sowie Wohnbau gewinnt zunehmend an Popularität, nicht zuletzt aufgrund einer Vielzahl von Möglichkeiten, die sich durch die Integration unterschiedlichster Komponenten ergeben. Heimautomation stellt in diesem Zusammenhang mit dem Schlagwort “Smart Home” generell einen höheren persönlichen Komfort bei gleichzeitiger Optimierung des Energiebedarfs in Aussicht. Meist bleibt jedoch trotz einer hoch technologisierten Umgebung, die Verantwortlichkeit für Konfiguration, Überwachung und optimierten Betrieb des Gebäudes beim Bewohner. Eine mögliche Lösung dieser Problematik ist, die Kontrolle des Wohngebäudes (teilweise) an ein autonomes System abzugeben, das Routineaufgaben im Auftrag des Benutzers durchführt. Eine Herausforderung hierbei ist unter Anderem die Berücksichtigung neuer, bisher außer Acht gelassener Einflussfaktoren. Eine Integration dieser zusätzlichen Parameter in Kontrollstrategien eines intelligenten Gebäudesystems ermöglicht die Umsetzung von zukunftsorientierten Anwendungsfällen, wie beispielsweise proaktiver Kontrolle. Um eine möglichst akkurate Abbildung des Gebäudes zu erreichen, müssen höchst unterschiedliche Bereiche berücksichtigt und entsprechend in einem grundlegenden Modell verknüpft werden. In diesem Zusammenhang wird im Rahmen dieser Arbeit ein umfassendes Wissensmodell präsentiert, welches die Umgebung eines Smart Homes möglichst vollständig abbildet, um eine wesentliche Grundlage für die autonome Schaffung von Benutzerkomfort sowie optimierter Energieeffizienz zu liefern. Als Formalismus der Wissensrepräsentation wurde die Web Ontology Language (OWL) gewählt, die eine semantische Beschreibung von Begriffen ermöglicht. Eines der Hauptprobleme in diesem Zusammenhang ist die Abstraktion von der Wirklichkeit, um eine effiziente Steuerung des Gebäudes zu gewährleisten, ohne dabei wichtige Einflussfaktoren zu verlieren. Zur Demonstration der Eignung des erstellten Modells für die genannte Domäne werden aus einer Reihe von Anwendungsfällen Kompetenzfragen generiert, die notwendige Eigenschaften einer derartigen Wissensbasis widerspiegeln und zur Evaluierung herangezogen werden. Die Ontologie stellt eine formale Beschreibung einer großen Anzahl unterschiedlicher Parameter dar, die unabhängig von der Umsetzung des Kontrollmechanismus entwickelt wird. Wenngleich die Wissensbasis im Zuge dieser Arbeit für ein Wohngebäude definiert wird, ist ein Großteil der Konzepte auch für den Einsatz im Zweckbau geeignet. Die Ontologie wird anhand eines einzelnen Gebäudes demonstriert, ist jedoch als verteiltes Vokabular insbesondere für die Anwendung in mehreren intelligenten Gebäudesystemen im Rahmen einer Smart City sinnvoll.

Contents

CHAPTER 1

Introduction

The deployment of automation technology in the home offers several attractive benefits, among them most prominently increased energy (or even resource) efficiency, improved residential comfort and peace of mind for the home owner. In the last decade, the term *smart home* has emerged as the keyword for automated dwellings of any form. The vision of a smart home is a house populated by a multitude of devices (actuators and sensors) that cooperate in an intelligent way to process the main control tasks relevant in building automation such as lighting/shading and heating/ventilation/air conditioning (HVAC). Recently, smart homes gather momentum as more and more residential buildings are equipped with building networks and become capable of offering advanced services to their users. From a high-level viewpoint, good performance of a smart home can be abstracted to two main goals: on the one hand, with ever rising energy costs, generally the *energy efficiency* of buildings should be improved. On the other hand, *user comfort* may be seen as one of the necessary objectives, as user satisfaction about a smart system's performance is a main acceptance factor for controlled buildings in general and especially those that involve the private area such as residential homes.

With the residential sector forming the second largest part of energy consumption in the European Union [29], there naturally exists a lot of optimization potential, particularly when it comes to energy-intensive building processes such as heating, cooling and ventilation as well as lighting. Efforts of the European Union in this respect involve the extensive deployment of smart meters facilitating real-time monitoring of a building's energy consumption through operators and further giving feedback to the customers about their energy expenditure [119]. Apart from measuring energy consumption and giving feedback to residents, a smart building further has to be aware of the current situation and assess the interior and exterior environment as whole in order to autonomously identify and exploit energy saving potentials. It is obvious that the more services are available in a residential home, the more parameters need to be calibrated either by experts or, more likely, by the inhabitants themselves. In the second case, residents often become overwhelmed with the multitude of different settings and parameters that influence building control. This may lead to the situation that default parameters of a smart control system are kept unchanged. However, each building as well as the behavior of people living in it are different,

and as such, pre-defined parameter settings are hardly ever optimal for a particular building or group of occupants. Therefore, what the residents and in this sense the users expect from a truly "smart" home is more that the building learns from their behaviors and detects habits that may subsequently be used to improve building performance.
Considering these challenges, it becomes clear that for a future smart home an intelligent software framework has to be created which operates the environment according to its current and near-future state and is capable of addressing the interrelations existing in the complexity of a residential building. In addition to a concept capable of addressing these challenges, a sophisticated representation of reality is vital due to the enormous amount of parameters from different domains influencing building operation: in this respect, the parameter model may take away some complexity from the intelligent smart home control system by explicitly and unambiguously representing the real world and dependencies existing in the domain of interest, i.e., the smart home, its surroundings and inhabitants. With regard to a real-world representation for a smart home system, the central question that needs to be addressed can be phrased as follows:

> *"How can a comprehensive and sophisticated representation of environmental parameters support intelligent behavior in future smart home systems?"*

This dissertation presents a parameter model for an autonomous smart home system and illustrates how to achieve a comprehensive view of the environment which is machine-interpretable and facilitates logical reasoning. The thesis focuses on knowledge representation and demonstrates how the inherent capabilities of the used representation language may be used to create a knowledge base supporting the advanced operation of an automated residential building through an autonomous control system.

1.1 State of the Art in Home and Building Automation

1.1.1 Building Automation

The automatic control, management and monitoring of various facilities to adjust indoor environmental parameters is already widespread in large functional buildings [141]. Different domains may be involved when considering building automation, with the field of heating, ventilation and air conditioning being the most common field to be automated for the longest time. One of the reasons for this traditional field of automation is that the tasks covered by this domain (i.e., heating, cooling, humidification) are among the main energy consuming tasks in the building for which depending on the size of the building often a huge amount of facilities are necessary. Building automation can in this case to a great extent support human operators to find optimization potentials and possible problem areas in the building. Building automation systems however often go much further than controlling HVAC and also integrate lighting and shading devices, safety, security and automatic energy management to name but a few. This variety of domains usually involves a highly heterogeneous device landscape often communicating through different protocols. In the past, this lack of interoperability often led to the situation that a single system operator had to cope with several systems each being responsible for a single task not capable of exchanging information between the systems and often remaining isolated. In the past

few years, therefore, the focus lay on the integration of systems from different domains into one single building automation system. This integration is mainly realized through a decentralized architecture where control of the environment is accomplished through communication between devices on different layers. Generally, three layers can be distinguished which are the field, the automation and the management layer [141] (cf. Figure 1.1): the *field level* includes actuators and sensors and the methods of how to access, alter or read their states. Devices at this level are used to sense the actual state of the physical world or perform actions that may alter the state of the environment. The *automation level* includes control tasks and is used to communicate with devices on the field level as well as to other controllers at the automation level. These two types of communication are used to on the one hand aggregrate information coming from the field level for access through the management level (vertical) while on the other hand directly exchange data of global interest between controllers (horizontal) [140]. The *management level* embraces information from the whole building automation system and may visualize it to an operator via a single interface or implement global tasks such as alarm collection or task scheduling. Also the interfacing with other dedicated systems is realized at this level.
As there exist various of different technologies and standards for building automation systems the comprehensive integration at the device level still remains a challenging task [141]. Different building automation networking technologies like KNX [136], BACnet [134] or LonWorks [77] traditionally use diverging field bus networks which mainly need to be integrated with each other through an IP-based network often used as system backbone. In this case, the integration over management-level gateways interfacing with all building automation subsystems becomes necessary. This integration however firstly tends to be problematic because of the high overhead of mapping multiple technologies to each other and further almost always results in the loss of information specific to single technologies. An abstracted view on field-level devices and their capabilities through a sophisticated model is therefore urgently needed to take away complexity from this task.
Recently, especially with the increasing affordability of high processing power the introduced hierarchy tends to flatten. Often the tasks of the automation level can to some parts be adopted by powerful microprocessor-enhanced sensors and actuators. Further, devices traditionally positioned at the automation level can through more advanced networking capabilities often be positioned at the management level. Also with respect to networks it is nowadays often only distinguished between backbone networks and field-level networks. The backbone that arranges the communication at the management level is mostly implemented through an IP network while on the field level the different technologies communicate via their open or proprietary field bus network. For communication between field bus networks and the backbone network special devices that facilitate tunneling of data packets may be used [75]. In current building automation therefore often only the field level and the management level are distinguished, while the automation level is split among these two layers.

1.1.2 Home Automation & Smart Homes

Building automation is currently becoming more attractive for the residential domain and the home of the future is filled with devices helping the resident to control the environment according to personal comfort and needs [206]. The interoperability of appliances in residential homes

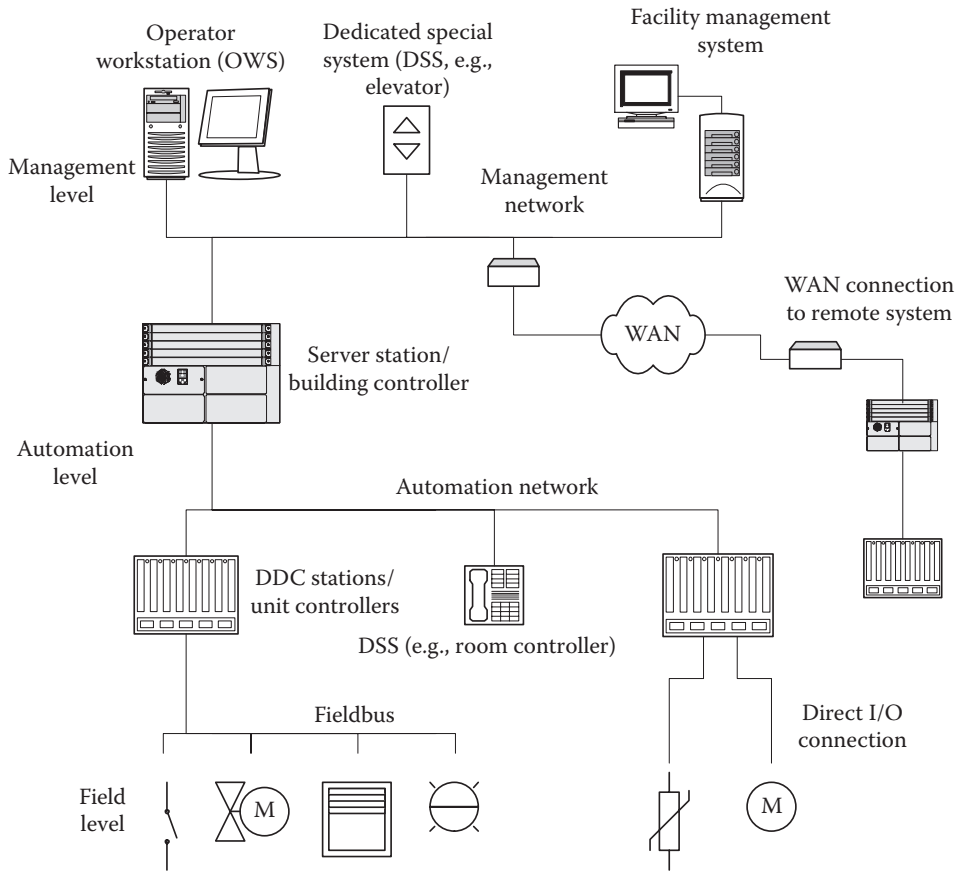

Figure 1.1: Three classical layers of the building automation architecture [141]. For each level, respective devices are shown.

that is recently discussed in numerous projects (e.g., [159, 33]) may in further consequence lead to ample energy-saving possibilities that have so far just been thoroughly investigated for commercial buildings. The inclusion of security and safety devices into the domotic system guarantees peace of mind of the residents what in turn can be seen as greater comfort. While in home automation the amount of appliances to be controlled as well as the size of the premises is usually an order of magnitude smaller than in classical (functional) building automation, the complexity of the domain remains high. This is not least due to the fact that a large amount of heterogeneous devices from different domains need to be integrated in an automated home. Further, some application domains which are prevalent in building automation do not take an important role in home automation while others not considered in functional buildings emerge in the residential sector due to the different field of application. Viewing the residential home as comprehensive building automation system, the following areas for appliances to be integrated can be identified [199] (cf. Figure 1.2): building automation (i.e., *domotics*) as first domain only covers parts of the field. In this area, the classic services also relevant for functional building au-

tomation like HVAC, lighting/shading as well as safety and security services are summarized. To obtain a comprehensive system that supports the occupants in day-to-day household activities, it is further necessary to integrate household appliances into the system. This domain comprises facilities commonly known as *white goods* like dishwasher, washing machine and refrigerator and allows the user to control and monitor them. To increase personal comfort the integration of consumer electronics (*brown goods*) is another area not to be neglected. In this case, for example facilities like TV, home theater, hi-fi equipment or generally entertainment devices need to be integrated into the system. *Ambient Assisted Living* (AAL) is a field that can be defined as the assistance of elderly people by a ubiquitous environment to live an autonomous life while the building system assists them in several tasks concerning day-to-day life and health issues. As this field is becoming more popular not least because of demographic developments in industrial countries [141], it is necessary for a home automation system to integrate devices and appliances that are not necessarily associated with the building or white and brown goods however specifically tailored to elderly people (e.g., blood pressure monitors, glucometers).
When these domains are integrated into one single home server, the monitoring and control of all necessary parameters becomes possible through a single user interface. Facilitating a user to start/stop devices, define schedules or change setpoints through one device that communicates with the home server (e.g., smart phone) assures comfortable control as well as supervision of setpoints and values. A uniform user interface further allows the fast propagation of possible alarm scenarios to the user assuring security and life safety in case of threatening situations. Of course, in this respect it needs to be considered that other than in functional buildings the people operating the environment are not necessarily familiar with the backgrounds of the different integrated domains. This naturally needs to result in interfaces that have to be tailored to the users' needs and state of knowledge, thus ensuring usability.
Considering the last presented domain AAL, it can be seen that simple integration of devices is often not sufficient. In this case, more complex tasks need to be solved by the home system to detect possibly hazardous situations or health concerns. Therefore, a more intelligent management system has to be introduced that plays the role of a virtual house keeper and supports the user in activities of daily living. Also, when looking at tasks like optimizing energy consumption or automatically establishing user comfort, an adaptive approach is necessary so that the system can cope with interchanging situations in the building. With the maturing of technical capabilities thus the vision of a so-called *smart home* can be considered more realistic. A smart home is a system that usually resides on the management layer of the building automation hierarchy and controls the devices at the field layer to combine different building services to an integrated responsive environment. The vision of the future smart home is that a system intelligently manages the environment on behalf of the user and takes care of routine tasks as well as optimizations optimally without user intervention. At the same time, the system has to remain intuitive and easy to control also for non-experts in case manual adjustments are necessary. While this vision already exists for some time, current smart home systems are not yet ready for all of these tasks as again a great complexity arises through pervasive control of the environment on behalf of the user.
Looking at current demonstration buildings as reviewed in [47] and research projects discussed in [199], it becomes clear that the development of intelligent buildings is still in its infancy. Four

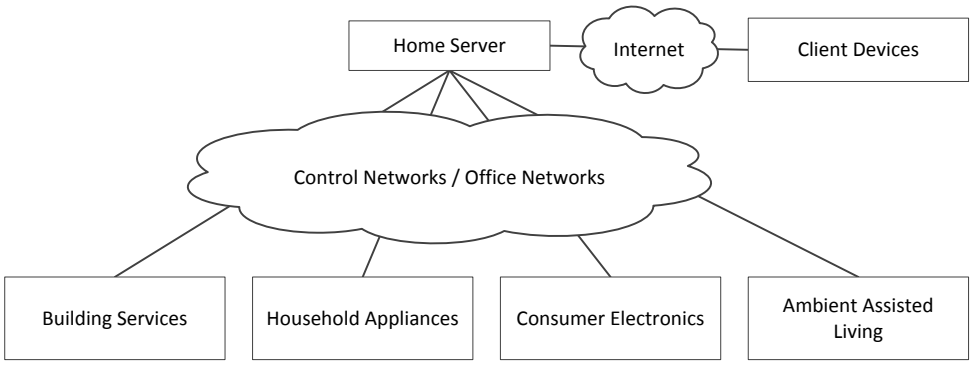

Figure 1.2: Domains of home automation appliances [199]

major trends can be identified in this context which are integration, assistance, efficiency and comfort. The majority of todays demonstration projects and buildings are concerned with the *integration* of the devices from different domains. A building is considered a smart home if it integrates a variety of building services and appliances into one system and make them available to the resident through a unified interface (cf. Figure 1.2). When looking beyond integration of the heterogeneous building landscape, a lot of research projects are devoted to create *assistive* buildings. The building system is in this case designed to allow the self-determined life of elders by informing them about situations out of the norm and actively act on behalf of the resident if dangerous or health-critical situations are detected. An active and proactive behavior is also needed when the focus lies on *efficient* homes. Often persuasion of occupants through appropriate reporting of the current state of the environment is a suitable mean to teach the home owner to behave energy efficiently. The author of [133] even proposes that persuasion of the occupant is the superior strategy to proactive control. While it is undoubtedly true that a system seldom adjusts the environment as perfectly as the user, the ordinary home owner often does not want to be responsible for all optimization tasks but desires the building to carry out tasks for him or her. In this case, it is of importance to detect the usual behavior of residents and act accordingly in order to save energy. With the advent of smart metering and Demand Side Management (DSM) for residential buildings [247], some recent projects also cover the integration of the smart home into the upcoming *smart grid*. A smart energy grid manages distributed energy production and consumption and smart grid-enabled smart homes may actively contribute to the optimization of these important tasks. Again focusing on the individual smart home and its users, the relief of responsibility to carry out tasks manually in the intelligent building in turn leads to a *comfortable* environment. However, comfort for a resident is not only to have the environment to be controlled on his or her behalf, but also to stay in control of the system and its actions in the case the system does not perform as desired [59]. In this case, the focus on providing appropriate interfaces for the predominantly non-expert users is a major field of research.

1.1.3 Challenges for Smart Home Automation

While there exist a huge amount of smart home projects in the scientific literature and also some demonstration buildings show the technical feasibility of smart homes, a widespread adaptation to smart homes is still to come. Reasons for this are manifold, however, there exist certain challenges that can be seen universally valid and have to be solved for smart homes to reach a breakthrough on the market.

Cost of Adaptation: Most of the demonstration objects nowadays are specifically built for the purpose to exhibit possibilities of automation in the future home. However, virtually all of the existing residential buildings are not designed for the integration of home automation technology and thus have to be retrofitted to accommodate home automation. In this respect, the authors of [70] speak of the "accidentally" smart home, a residential building that contains home automation but has initially not been intended for its implementation. In this case, the acquisition of home automation systems usually takes time as technology is costly and often limited to a single purpose. Apart from the relatively high initial investment, retrofitting existing houses is usually more expensive than equipping a home that is currently being built with automation technology [113]. This leads to a relatively slow adoption of smart technologies in the residential sector. For that reason, it is further likely that in existing residential buildings automation technology is introduced piece by piece, eventually leading to a highly heterogeneous system landscape. As already mentioned in Section 1.1, this integration challenge has to be met by a system operating the smart home regardless if the building is specifically designed for or retrofitted with home automation technology.

Domestic Routines: Residential life is often organized around relatively stable routines, like always getting up at the same time during workdays. One challenge for future smart homes is to detect such routines and optimize the building behavior accordingly. If, for example, in the morning the resident usually takes a shower before dressing, the building can prepare hot water for this activity in advance, knowing the usual time an inhabitant gets up. After the resident left the home for work it may then be safe to turn off the boiler as the system knows from the usual routine that the occupant only returns in the evening after work. In [59], it is observed that the times of waking up and returning home are main areas for optimization in the smart home. One particular challenge regarding this topic is further the common case that more than one occupant is living in the building and as such multiple routines are often interlaced.

Occupancy Recognition: As occupancy is one of the main influence factors when it comes to smart home control, special focus has to be put on this subject. While instantaneous occupancy can be sensed quite easily, it is a more complex challenge to exactly locate a person in a particular part of the building. In this respect, it would be desirable for the system to automatically infer a particular activity from a person's position. This however is a non-trivial task which is currently addressed in research literature (e.g., [217]) and no solution outside of testbed buildings is available so far. The differentiation between various residents is another challenge that needs to be solved in the context of occupancy recognition. While approaches using radio-frequency

identification (RFID) or near field communication (NFC) are possible means to address this challenge, they cannot be seen as a perfect solution. One major drawback of this approach is that humans tend to forget to put on these targets obfuscating their movements in the house. A solution on how to follow a person through a building which does not involve specific items or targets that need to be put on by the inhabitant still remains to be found.

User/System Interaction: In current residential homes due to missing visualization mechanisms it often remains unclear to the occupant where most energy is consumed. For example, the fact that electric devices consume a considerable amount of energy in stand-by mode is generally not realized. As studies show (e.g., [28]), a considerable amount of hidden energy is wasted by devices that stay in stand-by mode unutilized during long intervals although they could be taken off the power supply safely. Feedback is therefore important to make energy more visible to the user and allow him or her to adjust behaviors or the environment accordingly [58]. Many times, it is even necessary to provide the inhabitant with feedback on energy consumption to initiate beneficial changes in daily routines. Through an appropriate, timely feedback mechanism thus the inhabitants' behavior can be optimized [133]. When it comes to automated control, the system engineer further has the tendency to abstract from the occupant only focusing on system-driven optimizations. However, also with an automated environment the inhabitant wants to be capable of expressing his or her wishes to the system in form of preferences or manually defined schedules. These control capabilities have to be easily comprehensible as other than in functional buildings where the automation system is usually controlled and maintained by an expert (i.e., system administrator), the operators of an automation system in the residential area are mainly non-experts, i.e., the occupants themselves. The design of the user interface is a topic that is certainly of importance in this case. The occupant usually wants an interface that allows the easy access to simple system functions leaving the more complex tasks to expert users [18]. In future buildings, the role of the expert user may be taken over by an intelligent smart home system.

Predictive Control: In order to achieve a significant additional value in comparison with non-automated homes, smart homes have to provide an environment that responds to internal and external influences and anticipates future activities. While occupancy recognition and domestic routines have already been discussed, further information about the building, its embedded devices as well as inside and outside environmental conditions must be made available. Offering broad information about influencing parameters to an intelligent system, it becomes capable of computing better or even optimal solutions with respect to energy efficiency and user comfort. Obviously, the vast amount of information available about devices, sensors and actuators at hand in a fully equipped smart home enormously increases the effort for computing the best control choices for any given problem which poses a significant challenge to intelligent buildings. Another aspect that needs to be taken into account is that a control system has to adjust the environment unobtrusively according to inhabitants' needs while still giving them enough freedom to manually adapt the control of the building if necessary.

Energy Efficiency: To reach comfortable conditions in a smart home always goes hand in hand with the general building energy consumption. While in some smart buildings already a more or less intelligent system helps the user to reach personal comfort or increase energy efficiency seldom all possibilities are exploited. Many times, energy-intensive tasks such as temperature control are realized by simple two-level controllers, solely attempting to reach a certain pre-defined setpoint, leaving the operating environment and building conditions unconsidered. In case the control system optimizes energy efficiency, the general focus often lies on improving the consumption of the associated devices (e.g., [137]) more than the global building situation. While this is unquestionably an important step when coping with energy optimization, there are also other factors that need to be considered when trying to obtain a global view of the environment. The orchestration of devices in the home in order to reach an optimal behavior is an active field of research and can be seen as one major challenge not least because of the huge amount of energy consuming facilities. A further topic in this context is the integration of energy supply into smart home control with regard to different providers and tariffs as well as the future integration of the automated home into a smart grid. These influencing factors not only have to be considered for the energy demand optimization of the building itself but also to achieve an energy-optimal behavior in the broader context of a smart city.
Summarizing, two main general challenges can be identified: firstly, there exists the need for an integrated and predictive system environment that operates unobtrusively on behalf of the resident to automatically achieve comfortable conditions in the home. Still, such a system has to provide feedback and possibilities to manually adjust the environment so that the user does not feel patronized by the control system. All involved challenges with these tasks can be summarized under the common goal *user comfort*. Further, the usually high density of appliances in a smart home and the associated energy consumption need to be addressed by a future smart home. Also energy production and scheduling of specific operations with regard to the current energy supply situation can be seen as important in this context. Challenges involved with these objectives can be summarized under the goal *energy efficiency*. There exists the need for an integrated smart home system in which all facilities can equally participate and control tasks are executed autonomously operating on a comprehensive information representation comprising all kinds of data originating from appliances, the building its tenants and many more.

1.2 An ontology-based multi-agent-controlled Smart Home System

Regarding the challenges defined in the previous section, a future smart home is considered to be an intelligent building that acts on behalf of the users and is capable of optimizing energy efficiency as well as providing user comfort. However, the state-of-the-art smart home is much inferior to that vision. At the moment what is considered to be smart is often not more than a central access of different appliances. Frankly speaking, smart home systems of today allow some facilities of home automation and leisure to be controlled, for example, via a central infoscreen or PDA. Randall in his study introduced a categorization of smart houses into five categories with respect to the complexity of the system [194]: Houses that contain intelligent objects, houses that contain intelligent communicating objects, connected homes, learning homes and alert homes. Considering this classification, the most common type of smart homes nowadays can be seen

as *connected home*. While there exist certain projects to extend capabilities of current smart homes to learning homes (e.g., [170]) and alert homes (e.g., [55]), a comprehensive coverage of the environment is not yet reached. There are some projects which emphasize the importance of context awareness (e.g., [272]) however, the involved environmental influences are seldom observed comprehensively. According to [194], an *alert home* has to anticipate users' needs and therefore has to be aware of the internal and external conditions of the building at any point in time. An efficient operation of the building is dependent on a variety of parameters in addition to already proposed influences and classical building automation. The future smart home has to consider the whole environment completely and can be seen as adaptive system, supporting the residents in their aim of saving energy and at the same time not losing their personal comfort. This is the approach of the *ThinkHome* system [202], a comprehensive system for smart home control. In this system, a knowledge base in form of an ontology forms the base for an intelligent multi-agent system striving for the energy efficient operation of a smart residential home.

Realizing improved energy performance in the smart home first of all requires to intelligently control the smart home services and devices with respect to the current situation and context. The task of controlling and improving various devices and building services can be taken over by a dedicated smart home system. The smart home system shall be able to automatically control its environment, working under the constraints of guaranteeing user comfort at any time, yet operating as energy efficient as possible towards this task. Therefore, a lot of information must firstly be made available to the smart home system and, secondly, be analyzed and interpreted by it in order to derive appropriate control decisions. The first statement implies the availability of a multitude of information about the smart home environment in a machine-processable form. This can be achieved by designing an extensive knowledge base for smart homes which is also the focus of this thesis. The second characteristic demands the control system to exhibit at least some kind of artificially intelligent behavior. In Figure 1.3, an overview of such a combined system architecture is shown. The "ThinkHome" system framework has been conceptualized and designed in the course of an Austrian research project at Vienna University of Technology, Automation System Group from 2009 until 2012. It was funded by the Austrian research promotion agency FFG under the contract P822 170.

To cope with the complexity of the system at hand, the different parts have been assigned to three PhD students, each responsible for a particular system component. The design and implementation of a comprehensive software framework and multi-agent system is presented in [199]. Further, as context recognition is of major concern in a smart home system, a predictive control mechanism was created, that enables the system to perceive human behavior. In [129], research results regarding profile-based context recognition are comprehensively covered. Finally, it was necessary to define and implement a comprehensive knowledge base which allows to fully represent the environment of the system in a way that it can act as a basis for the other developed parts of the framework. This dissertation represents an exhaustive coverage of the findings in this respect.

The system concept was designed as team and therefore, naturally there exist overlaps in the three doctoral theses with respect to the description of the ThinkHome system. Certainly, the same set of use cases had to be applied in order to assure that all components base their functionality on the same set of requirements. Therefore, some contents regarding system design and

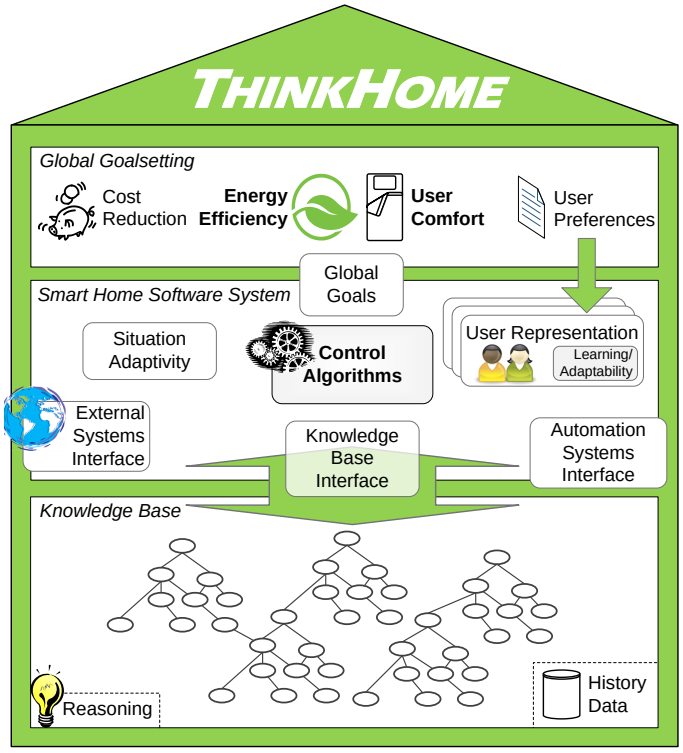

Figure 1.3: ThinkHome System Architecture Overview

requirements analysis in this thesis may be similar to explanations in the other dissertations that emerged from this project. It is however explicitly stated and emphasized in case content from the other theses is directly used. Nevertheless, the main part of this thesis, the definition of an ontology for smart homes, was done independently, and parts that may be similar to the other two mentioned theses only serve the purpose to put the presented developments into context with the designed system.
The following section discusses related works regarding multi-agent systems in the area of smart homes. The conceptual design of the ThinkHome system is presented afterwards.

1.2.1 Related Work

As research in the area of intelligent home automation is growing, there already exist some preliminary works on the use of multi-agent systems in automation systems. In the MavHome project, Cook et al. [55] propose using a multi-agent system in the home that is capable of learning inhabitant behavior. Methods such as data mining and event sequencing (episodes) are employed. Data is gathered using agents that communicate with the help of the Common Object Request Broker Architecture (CORBA). Actions are triggered based on hierarchic Markov models. Similar tasks are distributed to agents in the MASBO project [192]. Also Davidsson and Boman [61] proposed a multi-agent system in which different agents pursue the goal of energy

efficiency. They specified Personal Comfort Agents representing the inhabitants, Badge System Agents for user tracking, and, Environmental Parameter Agents to control the indoor conditions. A rudimentary evaluation is given based on a simplified room model. In [157], Liang et al. describe a basic agent based framework for smart homes. Five agents are differentiated: Space Agents interface with devices, Function Agents shall guarantee the realization of global goals, Personal Preference Agents make use of user preferences when executing control strategies, Resource Access Right Control Agents provide a lookup service, and, an Environment Variable Server Agent supports data acquisition over the Internet. While the approach seems feasible, it misses details on agents and their interactions and probably requires additional components to work. Also no prototype implementation can be verified. Zhang et al. [273] rely on the OSGi (formerly Open Services Gateway initiative) platform to implement an agent based framework. The approach targets the integration of different heterogeneous domotic devices and supports remote device control and fault diagnosis. UPnP (Universal Plug and Play) in combination with an agent framework is used for device discovery, registry, and management. Possibilities for the use of Artificial Intelligence (AI) in the home are described by Augusto and Nugent [10]. Similar to MavHome, the detection of discrete events is of importance. Event sequences are then derived using machine learning mechanisms such as decision trees and case based reasoning. All projects listed above have in common that they describe similar approaches on how to use agents for different tasks in buildings. Still, none of them proposes a comprehensive strategy how knowledge is managed in the system. Also, most approaches lack details on control strategy parameters and or focus on particular sub-challenges only, e.g., MavHome specializes on context awareness.

However, also preliminary work that proposes agent systems in combination with ontologies can be found. In [49], Chen et al. propose an ontology-based system for a smart meeting room. They introduce several use cases for a meeting room and employ context reasoning. Another approach for ontology-based context reasoning is taken in [74]. The suggested system uses the OWL for context modeling. Benta et al. [22] describe a multi-agent system working with an ontology mapping of the environment. The work also focuses on context awareness as well as user tracking and especially user behavior. Although these articles show some promising approaches in the field of context modeling and context awareness, architectural or energy efficiency considerations are not sufficiently considered.

Retkowitz in his work [204] describes a possibility to integrate heterogeneous services in smart homes with the help of an ontology. It is based on an ontology mapping for semantically equivalent service interfaces. In [205], the respective system architecture is specified in more detail. Different service layers are exemplified with the help of use cases. Main achievement are the unified service interfaces that enable a continuous specification, configuration and deployment process. The SOPRANO project [266] deals with the application of intelligent systems in the domain of Ambient Assisted Living. In the article, the authors stress that ontologies are not a perfect choice for reasoning on complex activities and conflicted data. Therefore, they propose their own approach where with the help of an context manager the services of the sensor layer are mapped to a user-service level. The authors of [120] propose a system which is based on J2EE and also uses multiple agents in combination with an ontology. The focus of their system is put on the industrial sector, in particular targeting logistics and scheduling applications.

Nevertheless, their study is a rare example of the practical application of an ontology-based multi-agent approach in a large real-world system. As shown in the previous paragraphs, none of the previously proposed work fulfills all the demands we stated. Nevertheless, the related work has been a valuable starting point for the ThinkHome project.

1.2.2 ThinkHome System Concept

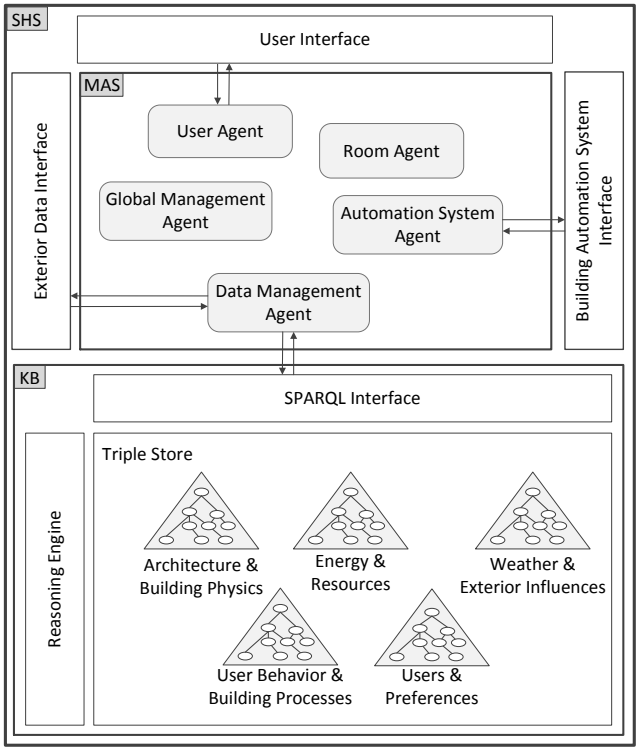

Figure 1.4: Smart Home System Components

Thinking of a future smart home, one key requirement is to provide a proactive system that is capable of acting on behalf of the user. While this is mandatory in the domain of Ambient Assisted Living to actively support impaired individuals, this characteristic is seen as desirable in any future smart home. In particular, it may support the occupant in different day-to-day activities concerning energy-efficient building operation while ensuring a maximum of user comfort. In this respect, intelligent control promises considerable benefits, especially if tailored scenarios and associated use cases are equally developed.
As pointed out in [201], the distributed nature of a typical smart home system in combination with a dynamic and inherently complex environment calls for a control software that is sophisticated and capable of handling these requirements sufficiently. The author furthermore states that a traditional, object-oriented approach may not be appropriate in this case and other, more

flexible software engineering paradigms have to be considered. These can readily be extended and are then capable of adapting to unexpected situations like spontaneous occupancy, inexact sensor readings or even temporal and permanent subsystem failures.

1.2.2.1 Multi-agent-based smart home control

With respect to the just mentioned challenges, a number of projects (e.g., [55], [22], [165]) identify an agent-oriented design methodology as an appropriate paradigm and framework for a smart home system architecture. The agent paradigm promotes the use of loosely coupled software entities designed to take over well-defined parts of the system's functionality while collectively implementing a sound system behavior [201]. In the building control domain, the implementation of a multi-agent system may especially aid in overcoming conflicting situations arising between different inhabitants, but also when concurrently striving for competing goals like optimized energy consumption versus individual comfort [165]. By definition, a *multi-agent system* can be seen as a group of software components where each entity has one or more specific tasks within the overall system. These components in further consequence interact with each other to overcome the complexity of a specific environment in order to reach previously defined goals. An autonomous agent in this respect generally perceives the environment and acts on it so that the environment's state is altered in a way that the agent's proclaimed goals can be reached [86].
Coming back to the previously described three-tier architecture of building automation, a multi-agent system may be seen as an additional, abstracted layer above the management level that learns from the situation in the building by means of sensors and accordingly performs interrelated tasks to reach designated global goals. In this context, *intelligent* agents are considered components that show features reflecting human behavior such as reactivity, proactiveness and social ability [267]. In [268], further the distinction between reactive, deliberative and hybrid agents is made. The choice between one of these types determines how powerful the capabilities of the multi-agent system are with respect to internal reasoning as well as how flexible the agents can react to changes in the environment.
For the envisioned field of application in a smart home, *deliberative* agents have been argued to be most appropriate as they can be seen the most flexible incarnation of intelligent agents [199]. Agents of this type on the one hand keep an internal model of the world while on the other hand are capable of autonomously following their own goals and may eventually also abandon or switch goals in case of changes in the environment. In other words, deliberative agents are capable of making decisions based on the knowledge available to them. For this reason, the smart home multi-agent system of the ThinkHome system is effectively composed of intelligent deliberative agents. The design and implementation of the multi-agent framework is extensively described in [199] and the interested reader is referred to this work for further details.

1.2.2.2 A knowledge base as foundation of advanced control

The construction of a world model in each agent of the envisioned multi-agent system is strongly supported by knowledge represented in a dedicated knowledge repository. The design and discussion of such a *knowledge base* (KB) is the main focus of this dissertation. The KB, in the

present case realized as ontology, most importantly represents a source of common knowledge shared between different agents, providing an unambiguous view of the operable world while describing the entities of interest for a particular application [103]. To define a shared ontology in the building automation sector, several different domains need to be covered (cf. Figure 1.4). The identification of these domains and the appropriate representation of associated concepts relevant for the autonomous control of a building constitutes one of the main contributions of this work. If, like in the ThinkHome system, the optimization of energy efficiency and the concurrent maximization of comfort are main goals, the inclusion of models about the building architecture and building physics properties, facilities and their current energy consumption or production, exterior influences like the current and future weather, building processes as well as user habits and preferences are valuable to reach the proclaimed goals. As the broad definition of these domains leads to a massive amount of associated concepts and relationships, it is the aim of this work to determine which parts need to be covered and are relevant for the operation of a building while also investigating whether information of a particular domain can be automatically retrieved. With a concise representation of knowledge, the multi-agent system can take several influencing factors into account to, for example, optimize the start/stop time of energy intensive tasks like heating and cooling. Another advantage of a knowledge base for a smart home system is that it facilitates the definition of relationships of sufficiently high complexity to generate a sophisticated view of the world the multi-agent system operates on. The inherent representation capabilities of an ontology allow to model the environment in a sophisticated way, taking away complexity from the software system and leading to a disconnection of information representation from the control logic. With other words, an ontology permits the separation of domain and operational knowledge [179] which in turn results in a clearer system design as well as portability and reusability. Reasoning as integral feature of the operation on ontologies can in this case be used to complete the perception of the environment considerably supporting the multi-agent system in the achievement of its objectives. For the successful autonomous operation of a smart home, an extensive knowledge base therefore provides the mandatory foundation for a multitude of use cases and application scenarios.

Obviously, an isolated design of the knowledge base may not allow a smart home system to yield the expected benefits to full extent as decision processes in the control system (e.g., which cooling strategy should be applied) are based on extensive information. Therefore, the modeling of the knowledge base is done with the future application in mind. During the operation of the smart home system, further a close interaction and cooperation between the multi-agent system and knowledge base needs to be considered. As the focus of this dissertation lies on the representation of relevant domains, it only gives an outline of the interaction with the knowledge base leaving out advanced topics regarding the communication (e.g., time constraints). By representing knowledge about the building in a machine-interpretable way it is ensured that necessary information can be accessed from and exploited by a smart home system that automatically executes control strategies in the residential home.

CHAPTER 2

Knowledge and its Representations

Storing data for processing by computational programs can be done at various levels of complexity. To one extreme, an application may extract the desired data from a simple list of terms. While this sort of data-keeping does not imply any structure, it however tends to become problematic as soon as the amount of information to be processed increases. On the other side, data may be highly structured and enhanced with semantic annotations grounded in formal logic describing its characteristics and applications and making reasoning and automatic inferences possible. In this case, one may safely talk about knowledge. In order to discuss different data structuring possibilities, first of all it is important to clearly define the different terms data, information and knowledge.

2.1 Data, Information, Knowledge, Wisdom

When designing a knowledge-based system, it is necessary to have a clear vision of the dependencies between data, information and knowledge. With such a definition, it becomes comprehensible which advantages arise through the definition of a knowledge base, and what its benefits are compared to classical relational database systems. In the article [1], the author was the first to present a hierarchical model relating the terms of data, information, knowledge and wisdom to each other. Figure 2.1 shows a further developed version of the original model, omitting the additional level "understanding" that exists in the original model. According to [20] the level of "understanding" is merely a process that facilitates a transition between individual levels of the pyramid, meaning that a transition can only happen if understanding is present. *Data* as such are symbols that can be read, stored and processed however do not have any associated meaning. When data is described in a certain context and thus a meaning is added, it becomes *information*. Different interpretations may exist for specific data and for each of these interpretations a model can be defined that represents information. *Knowledge* builds on data and information, and facilitates reasoning to infer additional information. *Wisdom* at the top of the pyramid can solely be reached with understanding the rest of the world, setting the known knowledge in context with the world and involves some kind of assessment of what is right or wrong or good or

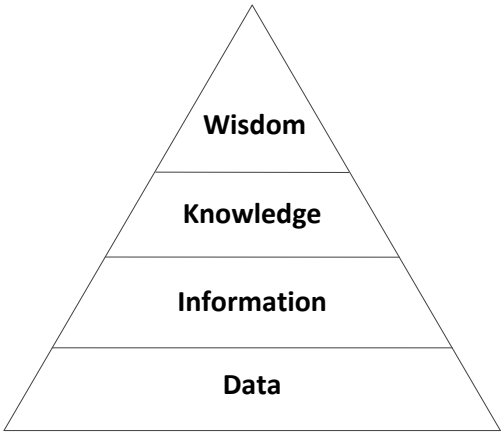

Figure 2.1: Data, Information, Knowledge and Wisdom pyramid, based on [1] and [20]

bad in this world. Wisdom can generally not be reached by a simple transformation process, and thus perhaps has more to do with human intuition, understanding, interpretation and actions than with systems [208]. For this reason, the following elaborations abstract from wisdom, and lay the main focus on the three levels data, information and knowledge, that are significant for an autonomous computer system.

Data

In a way, every simple statement in form of characters, digits or terms can be named data. Terms on this level come without attached semantics or any particular interpretation, they do not have a meaning attached. For manipulation in a computer system, simple data is not sufficient, as to process terms without a meaning is infeasible. As an example consider Table 2.1: Without further explanation, these data are meaningless as it could for example denote an identifier, a temperature value or a paper size. This means, data cannot be interpreted by a computer system without further structure, however, it can be seen as the basic building block for information and knowledge and thus represents the lowest level of the pyramid.

Table 2.1: Some data

22	C

Information

Information is organized data with a meaning in a particular interpretation. Data is therefore connected via relations and structured with definitions. This means, through relational modeling it becomes visible to a system what a given datum represents. In current software systems, the most common way to structure data is as relational database, which organizes information

in tables and a set of different relations. For the aforementioned data a simple explanation like “room temperature”, with the fields “value” and “unit” (Table 2.2) makes it clear for the processing program how to interpret the given data, and further to relate the value to other known values.

Table 2.2: Data extended to information

Room Temperature	
Value	Unit (C/F)
22	C

Knowledge

“Knowledge is information that is put into action” is a definition that can be found in [63], and describes the connection between information and knowledge. This means, information may be interpreted by a human or computational inference mechanism that reasons on available information, infers new facts from it, and thus generates knowledge. A logical reasoner may interpret the information that it receives through a formal knowledge definition. This means, data that is received is interpreted in the context in which it is used and through this context-based use becomes knowledge. Considering the definition of a temperature sensor shown in Listing 2.1, this device can be described as any sensor that has a relationship to a temperature value via the `senses` relation. Such definitions may be part of a system’s knowledge base while similar state-

2.1: Axiom describing a temperature sensor in Description Logic

$$TemperatureSensor \equiv Sensor \sqcap \exists senses.TemperatureValue \tag{2.1}$$

ments will be necessary for `Sensor` and `TemperatureValue` respectively. If a new element showing these characteristics is added to the knowledge base, with the right kind of inference mechanism the system may infer, that the added element is of type `TemperatureSensor`. This means, the inference on basis of the logical definition may produce additional information about a specific item. One main quality of knowledge is therefore the inferability of new information out of existing facts. Different processes that convert information into knowledge include among others [208]:

- *Synthesis of multiple sources of information over time*
- *[A sort of] belief structuring*
- *Study and experience*

Further, in [208] some important points that differentiate knowledge from information are identified, while the two most relevant in the present context are:

- *[Knowledge is] a mix of contextual information, values, experience, and rules*
- *[Knowledge is] information, expert opinion, skills and experience*

Most of the mentioned points depend directly on the expertise of the knowledge engineer or domain expert in a particular field. The designed knowledge models in Chapter 4 do consider these points when representing knowledge and they acted as a kind of guideline of how knowledge can be obtained.
Considering intelligent agents as protagonists in an intelligent building, we can further differentiate between knowledge and beliefs: Knowledge represents information about a domain, and can be seen as true and justified *belief* [189]. While this distinction is necessary for definitional purposes, in Artificial Intelligence they are not necessarily distinguished in a strict manner. In this work, knowledge can be seen as generally accepted information and is represented in a knowledge base, while belief is the subjective view on the world of each agent, that can change, the more information it gathers.

2.2 Forms of Knowledge

Considering knowledge representation in computerized information systems, a distinction between different forms of knowledge needs to be given. Types of knowledge may be identified in various ways in philosophy, religion or engineering. This section introduces different classifications of knowledge that are important for the representation of a specific domain in knowledge-based computer systems.

Qualitative and Quantitative Knowledge

Qualitative knowledge is a type of knowledge that describes features of an entity in the environment that can be observed. The article [87] defines qualitative knowledge as *"any kind of knowledge that does not always allow a correct and consistent match between the represented objects and the real world, but can nevertheless be used to get approximate characterizations of the behavior of the modeled domain"*. As such, qualitative knowledge is fuzzy and allows a general description of a situation or entity. An example would be a weather service that reports a "rainy" weather condition: while in this case, precipitation can be expected it is however not clear how intense the expected precipitation is going to be. The counterpart, *quantitative knowledge* describes knowledge about an object that can be measured, like exact dimensions of a window or the amount of access points of a specific building automation device.
While quantitative knowledge is explicitly recorded in a knowledge base, qualitative knowledge may be derived from the context in which a specific item is used. For example, the exact dimensions of a window could be used by the knowledge base to derive if the window is large, small or medium in relation to other windows in the building. Both types of knowledge have to be represented, as they are both needed in order to reach a thorough understanding of a domain.

Procedural and Declarative Knowledge

The distinction between procedural and declarative knowledge is essential for any computer-based information and knowledge procession. *Procedural knowledge* can therefore be seen as *"knowledge of how things are done"* [53]. Classic computer programs mainly hold procedural knowledge, which is stored in their statements and procedures. However, procedural knowledge may hardly be used for making statements about a domain or entity. As such it is the propositional or *declarative knowledge* that describes properties of an object with the help of assertions [239]. Declarative assertions may be either true or false and as such represent facts that are known about a specific entity in the domain of interest. Clearly, in a knowledge base depicting a model of reality, the type that can be found is propositional knowledge, as the main purpose is to describe the world of interest by facts and logical rules. It is however also possible that procedural knowledge is described with the help of declarative knowledge in a knowledge base (e.g., building processes, cf. Section 4.4). In a knowledge-based system, procedural knowledge is however for the greater part being found in the software or agents that operate on knowledge, for example, in the execution plans of BDI-Agents [199].

Explicit and Tacit Knowledge

In literature about knowledge management, it is further sometimes distinguished between knowledge that is assertively stated, and such knowledge that is "hidden" in the mind of an expert. These two types of knowledge are on the one side known as conscious or explicit and on the other side as tacit knowledge. *Explicit knowledge* can be seen as knowledge that is formally described in a document, database or similar storage system and thus directly available to a processing program. *Tacit knowledge*, on the other hand, is harder to comprehend: it represents specific know-how of an expert or belief of an agent that is not being held in explicit form. Therefore, it is relatively difficult to express tacit knowledge for a computer system.
In order to represent tacit knowledge, it may be modeled as facts into a knowledge base by the particular person that holds the tacit knowledge, again transferring it to explicit knowledge. According to [208], however, there is no difference between explicit knowledge and information, which is to some extent true, as everything that is explicitly stated may be stored and processed in an information system just as it is. In [93], tacit knowledge is further described as *"what an agent obtains when it observes its environment and makes internal representations of what it perceives"*. When the environment of the agent is described in a logically sound form in a knowledge base, the explicitly stated tacit knowledge codified by a human expert can be utilized to supply the agent with a complete view of the environment including knowledge that may not be explicitly stated, i.e., which is only available implicitly. In a knowledge-based system therefore, a kind of reasoning mechanism is necessary which is capable of evaluating explicitly stated facts to retrieve knowledge that is only implicitly present [12].

Theoretical and Practical Knowledge

Another distinction that can be drawn is one between theoretical and practical knowledge. Referring to the philosopher Kant, the article [78] describes that *"theoretical knowledge is knowledge*

of what is, or of what exists whereas practical knowledge is knowledge of what ought to be, or of what ought to exist". Theoretical knowledge therefore gives an exact definition of the reason why a specific concept exists in the described domain and as such is responsible for creating the context for a particular object. At the same time *practical knowledge* leads to a deeper understanding of an entity and its connections through experiencing and learning how the object reacts to specific relations. Practical knowledge can therefore be seen as an important requirement for deliberate actions of autonomous agents.
Theoretical knowledge allows a generalized view and therefore is applicable to various types of interaction with an object. Then again, practical knowledge describes the ability to apply knowledge in specific circumstances [109]. In this context, theoretical knowledge is therefore the foundation to automatically apply knowledge, hence to exercise practical knowledge. According to [109], practical knowledge further has an explicit and tacit dimension and can be seen as a conglomeration of these two forms of knowledge. What can be seen through this example is that the discussed knowledge categories do not have strictly defined borders and may be explained by one another or be merged into each other. The two dimensions of theoretical and practical knowledge are also reflected in a knowledge base: theoretical knowledge about an entity in the domain of discourse is represented in its axiomatic description which is specified using axioms satisfying specific logical rules. Further, practical knowledge is how these rules and axioms are applied by agents and what is the outcome of the knowledge application in different contexts and situations.

Syntactic and Semantic Knowledge

For computer systems, the phrase *syntactic knowledge* describes the knowledge about the structure of the used terms and language. Syntactic knowledge about a programming language for example is which terms are allowed in this particular language and in which sequence these terms may be used to generate a valid program structure. The test for syntactic correctness of a computer program is generally carried out by the parsing mechanism of a compiler. Also within databases and knowledge bases, there exist syntactic rules according to which information is represented. As such, syntactic knowledge is important to recognize the structure of the knowledge store and in further consequence to determine the meaning of a term or sentence, hence, its semantics. *Semantic knowledge* therefore is the knowledge about the meaning of terms. Semantics of terms always involve an objective and systematic discourse of the treated term, its referenced object and the environment (i.e., reality) in which it exists. While syntactic knowledge does describe the term itself, semantics create a connection between the term and the concrete object in the world that is portrayed by this term. When it comes to the representation of knowledge in AI and general in computer programs, the concept of semantics is deemed to be one of the most important.
The semiotic triangle (Figure 2.2) as defined in [182] is a foundational description for the connection between symbols or terms, concepts and reality. A symbol (or term) in this case stands for a specific object in the real world. However, without interpretation of the term and the creation of a relation to an entity in the real world through an observer, the term and object do not have a direct connection, which is indicated by the dashed line in Figure 2.2. The interpretation of the term as a conceptual notion therefore needs to be used to connect it with the actual object.

There may exist more than one interpretation of a term and as such, the meaning of a term may be ambiguous when viewed from different observers in different angles and in different contexts. As the relation between terms, concepts and objects tend to fuse in the mind of a human observer [231], the semiotic triangle can be seen as fundamental for a knowledge base to relate terms and objects in reality and describe their meaning in form of logical concepts.

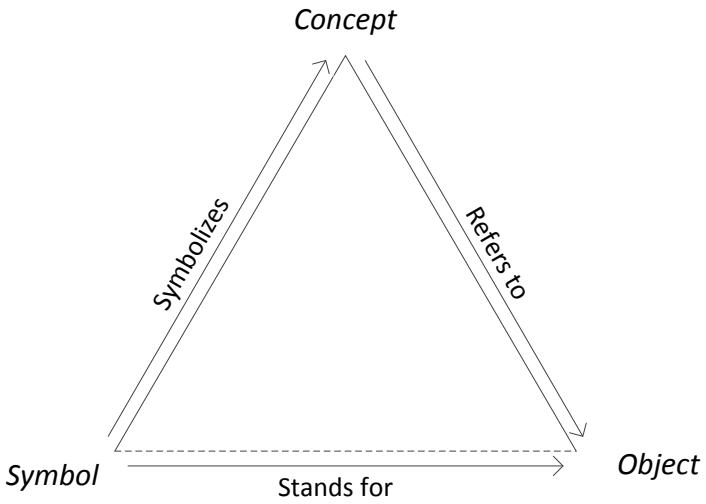

Figure 2.2: The semiotic triangle [182].

2.3 Knowledge Representation

Generally speaking, knowledge representation is a crucial field in Artificial Intelligence [87] and nowadays also has grown in appeal to web-based solutions in the context of the Semantic Web [27]. A knowledge representation formalism roughly needs to be able to create a declarative, semantic description of a particular domain and provide a mechanism that is able to deduce knowledge implicitly available in the representation [12]. While implicit and explicit knowledge has to be represented in a formal way by logical dependencies and deduction rules, a reasoning mechanism has to be capable of making the implicit relations visible to the human user or software that operates on and manipulates data in the knowledge store. Another attempt to generally describe the concept behind knowledge representation and the underlying formalisms is taken in [62] where five different roles are introduced:

- **Surrogate:** Above all, a knowledge representation is a substitute for reality and as such describes a more or less accurate picture of the world. An agent (e.g., in a multi-agent system), basing its beliefs on a knowledge base becomes capable of reasoning about the world and making decisions already before taking actions. This can be accomplished with the help of a concise description of objects in the world through logically defined terms. The conciseness or "fidelity" in this respect means that it is necessary to represent

dependencies in the world in a way that its representation is sufficiently rich in detail for the agents' needs while abstracting from other parts not being of further interest for the system or agent operating on it.

- **Set of ontological commitments:** To limit the complexity of the knowledge base when representing reality it is necessary to abstract from certain parts of the world. In this case, the definition of a set of ontological commitments means to create a model of the world and select the parts that need to be represented. As described in [103], *ontological commitment* in the case of a knowledge base describes an agreement about objects and relationships between them. The ontological commitment for an object in the domain of discourse therefore covers all objects and relationships that are needed to completely define the entity with respect to the application domain.

- **Fragmentary theory of intelligent reasoning:** As a further requirement, a knowledge representation needs to entail intelligent reasoning which involves the fundamental definition of intelligent inference through a set of permitted and recommended inferences. Intelligent reasoning through a knowledge representation is furthermore usually fragmentary as with the help of this reasoning only a part of human intelligent reasoning is covered.

- **Medium for pragmatically efficient computation:** To facilitate efficient reasoning with the help of a computational inference mechanism, a knowledge representation has to define a structure as framework for the organization of information. This frame may not be too expressive in order to ensure computational efficiency, however at the same time needs to provide enough logical power to make intelligent reasoning possible and thus represent knowledge in a sufficiently rich form.

- **Medium of human expression:** A knowledge description is also a way to model the world as it is seen by a human observer to communicate its dependencies to either a machine or another observer. Among other things, with respect to this role it is necessary to evaluate how well a specific type of knowledge representation is capable of describing the world as well as to assess the expressive capabilities of the representation with respect to simplicity.

It is argued in [62] that all five roles have to be considered to provide a practicable and powerful knowledge representation formalism.
Summarizing, knowledge representation for computer systems thus is about focusing on important aspects of the world to be expressed, omitting unnecessary details. In this way, a specific domain representation does not have to model the world to every detail, as every *model* to be used for describing a certain part of reality may be seen an abstraction from the world it represents to the parts that are interesting for a particular domain or application. An adequate level of detail has to be chosen in order to represent knowledge for a software system: while a high-level description may be easier to comprehend for humans, a low-level description brings more information, at the same time complicating the reasoning mechanism [189]. In this respect, the *universe of discourse* is defined to be the accumulation of terms which are relevant in the covered domain. This *domain representation*, which is created from domain knowledge retrieved

through an expert, must further be encoded in a specific knowledge representation formalism or language in order to make the knowledge machine-interpretable [265].

2.3.1 Knowledge Representation Formalisms

Knowledge representation formalisms in Artificial Intelligence have a long history, starting in the 1970s with approaches on the one hand evolving from trying to represent human memory in a machine-readable form (i.e., non-logical approaches) while on the other hand embracing pure logic (cf. [172]). One of the main requirements for a knowledge formalism is that it is capable of representing the knowledge that is relevant for a particular application [12]. As such, it necessarily has to satisfy the five roles of a knowledge representation introduced before. While there were attempts to find one single knowledge representation valid for all application domains, there also arise certain benefits from a diversity regarding formalisms [220]: Firstly, different formalisms can support varying inference processes. Further, there may exist the need for different representations with respect to the field of application that they are applied to.
Considering formalisms relevant for ontologies, the non-logical formalisms semantic networks, taxonomies, and frame-based systems need to be mentioned. The first concept of ***semantic networks*** was already published in the 1960s [193], and it is broadly used in various fields of application in the AI. Semantic networks model knowledge with the help of directed graphs, this means, as basic elements it differentiates between nodes and edges. With respect to knowledge representation, *nodes* may be seen as terms that are described, and *edges* are the connection between the terms in the domain. With the help of different relation types, miscellaneous associations between the terms may be defined, facilitating the modeling of knowledge about a domain. The weighing of edges and the distance between two objects in the graph can further be used to express a measure for their relationship [232]. Generally, the definition of a semantic network is logically unrestricted, however, different restrictions may identify sub-categories. In this respect, a ***taxonomy*** may be seen as a special form of semantic network, describing a subtype where associations between nodes are limited to "is-a" relations. In this case, an inheritance structure between concepts can be realized. As a semantic network is usually purpose-built, it can be customized regarding the application that it is applied. This leads to a highly specialized knowledge representation that may support tailored reasoning. However, the application of semantic networks in a general context is not easily achieved, mainly due to the lack of defined semantics [172]. Further, in semantic networks, many nodes and directions are needed for relatively simple knowledge representations. Therefore, relatively easy dependencies tend to become complex and use many nodes. Another drawback of this type of knowledge representation is, that querying a semantic network is relatively inefficient, as many nodes and relationships need to be searched to gather full knowledge about a represented concept [63].
Frame-based models (cf. [164]) can be seen as another example for a non-logical knowledge representation formalism. Introduced in the early 1980s, they were initially intended to represent a non-logical alternative for knowledge representation to more rigid logical approaches. In frame systems, *frames* are used to describe objects while *slots* represent properties of frames. Additionally, there may exist *facets* that are identified for frame slots and may include statements that, for example, are executed in case a value is written to a slot. This way it becomes possible

to integrate simple type checking in a frame-based model. From the general structure, a frame-based knowledge representation is comparable to object-oriented programming [94]: similar to the distinction between classes and objects in object-oriented programming, in frame systems a differentiation between a class frame and an instance frame is possible. Usually frames are connected to each other through a network-like structure and therefore, frame models and semantic networks can be associated with the same class of network-based knowledge representation formalisms. With respect to reasoning it becomes possible to conduct a mapping of more specific frames into more general ones and inferring slot fillers [213].

Continuing with logic-based knowledge representation formalisms, for years, classical logic has been considered as one possible knowledge representation formalism for computer science mainly because logic can be seen as a concise and clear way to infer new information from a set of formal statements through evaluating them to true or false values. Two important types of classic logic are first-order predicate logic and propositional logic. As a functional subset of first-order logic (FOL), ***propositional logic*** is a basic form of logic and as such may be used to represent facts. While its decidability and simplicity are certainly advantages of this logic, it however lacks of certain qualities classifying it as a competitive choice for realizing a knowledge base in a computer system. One of the main disadvantages is its limited expressivity: propositional logic allows a basic representation of facts that may be related to the states "true" or "false", as well as combinations of facts, however it is not possible to express more advanced concepts like relations between objects or to define variables.

Considering first-order ***predicate logic***, or shortly first-order logic as second example of classic logic, it can be seen, that this form of logic is much more suitable for knowledge representation in computer systems, basically because of two facts: on the one hand it can express *predicates*, that may be translated to one or multi-dimensional relations. These predicates can be defined on variables, which significantly raises the expressiveness of the language when compared to propositional logic. On the other hand, a further extension of the language compared to propositional logic are *quantifiers* that facilitate statements about all or some objects in the universe described through a set of logical statements. While these extensions allow the creation of extremely powerful logical expressions that can successfully be used to describe relations in the real world, they also make unrestricted FOL undecidable with respect to reasoning. This means, an automated reasoning mechanism deducing inferences from a FOL-based knowledge representation would not always come to a conclusion whether a particular statement can be inferred or not. Another difficulty appears with respect to the representation structure: because of its lack of a concise structuring mechanism, classic logic is just in particular situations useful for a practical knowledge representation. A problem that develops in this respect is that if knowledge is to be represented with the help of propositional or predicate logic, an exponential growth known as "combinatorial explosion" can be observed [60], leading to overly complex descriptions of domain concepts. Another fact is that this growth significantly slows down reasoning on represented knowledge. Because of these deficiencies, especially in Artificial Intelligence often representations that are different from classical logic are used.

Conceptual graphs (cf. [226]) are a logical derivative from semantic networks and frame-based systems. This formalism can be used as a graphical knowledge representation, however, the graphs can also be represented as FOL formulas facilitating logical reasoning on the graphs. As

such, this representation mechanism is the first of the mentioned formalisms that defines *logical* reasoning on the basis of first-order predicate logic: on the one hand it can be inferred if a graph is *valid*, hence, if the corresponding logical formula is sound. On the other hand, logical reasoning on conceptual graphs can perform *subsumption* reasoning, which means that it can be deduced if a formula representing a given graph can be implied from another formula describing a different graph [213]. As previously mentioned, reasoning on unrestricted first-order predicate logic is undecidable, and therefore, the restriction of the conceptual graph to conjunctive queries is necessary to ensure termination of a reasoning algorithm.
In the past decades, research was focused on finding other, more suitable forms of logic for knowledge representation in information systems, not only because of deficiencies in classical logic regarding structure and decidability. Hereby, Description Logic gained importance in recent times and eventually became the prevalent logic when representing knowledge as ontology.

2.3.2 Description Logic

In the mid 1980s, it was recognized that semantic networks and frame-based systems essentially describe fragments of predicate logic [63]. This discovery led to the development of a family of logics named Description Logics (DL). Basically, DL developed from the system KL-ONE (cf. [39]) which was realized to confront ambiguities of semantic networks and frame-based systems [14]. Description Logic adds logic-based semantics and a formal description to this system. As a formal language, most DL dialects can on the one hand be seen as decidable fragments of first-order logic (cf. [13]) while on the other hand it was also shown that a basic version of Description Logic may be considered a variant of modal logic (cf. [213]). One of the great advantages of DL is therefore that through decidability also efficient reasoning mechanisms can be defined which is one of the prerequisites for an intelligent system.
In Description Logic, a *terminology* is a possibly hierarchical structure used to represent the knowledge of an application domain [172]. The logic in this case strictly separates between the description of general concepts and concrete instantiations. All logical statements that concern concepts and their definition are commonly known as terminological box or *TBox*. Otherwise, the assertions that are made for individuals of the particular domain to be represented and relationships between them are known as assertional box, hence, *ABox*. The TBox differentiates between concepts and roles to describe the domain. A *concept* can in this respect group sets of individuals and this way achieve a classificiation of terms in the modeled domain. With respect to concepts, simple names denote atomic concepts, while complex concepts can be built using atomic (or complex) concepts combined with certain logical constructs. *Roles* further allow the definition of relationships between individuals. Also for roles, names are used to describe the elements that can be used to connect individuals with each other. Further, the ABox uses the definitions in the TBox to associate a specific individual with a concept, which is similar to the previously described relationship between classes and individuals of frame-based systems. In an ABox again two different types of elements can be expressed: unary predicate symbols are used to describe the relationship between an individual and a concept (i.e., concept assertions) while binary predicate symbols can further be defined to associate two individuals with each other through a particular role (i.e., role assertions).
What can be noted is that the TBox in a Description Logic model is generally variable-free,

however concepts and roles may still be described using other elements of the TBox as well as a set of logical operators available in the respective Description Logic making the language family capable of describing the semantic structure of a particular domain.
As a general extension, logical rules may be defined for some languages of the Description Logic family to enhance the knowledge representation capabilities of the vocabulary. Rules in this case may be used on the individuals that are defined in the ABox to additionally infer dependencies between them which in turn can be added to the vocabulary description in an intelligent system operating on a knowledge base [94].
The reasoning tasks related with the TBox in Description Logic are satisfiability, subsumption, equivalence and disjointness. The article [14] shows that for basic Description Logics all of the reasoning tasks can be reduced to satisfiability. *Satisfiability* means that with a reasoning procedure it becomes possible to determine if the TBox is satisfiable, hence, if the model is defined in a way that it is noncontradictory [94]. This form of reasoning is mainly used in the development stage of a vocabulary, giving feedback about problems with definitions of certain concepts in the TBox. *Concept satisfiability* in this respect can be seen as a form of satisfiability test which can be used to determine if it is possible for a particular concept in the TBox to have members of a general ABox. If it is not the case, this concept is unsatisfiable, this means that regardless of the ABox it will not possibly contain any members because of its concept definition. The second form of reasoning on the TBox, *subsumption* reasoning, further may be used to determine if a particular concept can be inferred to be more general than another concept. This way, a hierarchical conceptualization of terms can be automatically created. With respect to the ABox, two more reasoning tasks can be identified which are consistency checking and instance reasoning. *Consistency* reasoning validates if the assertions in an ABox are consistent with respect to a particular TBox [14]. Finally, *instance* reasoning may be used to infer if an instance is member of a particular concept.
The complexity of a language from the Description Logic family depends on the use of certain logical constructs and therefore can serve as measurement of its expressiveness. Complexity in this respect is important considering reasoning on a Description Logic-based vocabulary: a logical reasoning algorithm has to support the expressiveness of the model to be able to deduce all possible inferences. Further, with a more complex Description Logic, naturally the reasoning algorithm that operates on the logic also becomes more complex and in turn takes longer to finish its deduction tasks. For a complete coverage of Description Logic complexities and associated constructs the interested reader is referred to [14].

2.4 Ontologies

Ontology in philosophy is known as the science that describes objects, their definition and relationships between them, hence, the science of being. In the field of philosophy, Ontology[1] may therefore be seen as a research field that strives for a description of reality which is universally valid and correct (cf. [63]). In information science, a slightly different approach is taken. As not everything has to be represented in one single Ontology, multiple ontologies may describe

[1] In this case Ontology is capitalized to denote that in philosophy there can only be one single ontology that describes the world “as it is”.

particular parts or views of reality. In this case, an ontology can merely be viewed as conceptual model of a particular domain rather than a complete description of reality. The most commonly cited definition for an ontology in information science is found in [103]:

> *"An ontology is an explicit specification of a conceptualization."*

This quotation characterizes an ontology as a description and classification of terms. A conceptualization in this case can be seen as abstraction from reality to terms that are of interest and useful in a particular application domain. Hereby, an ontology usually only partially describes terms and dependencies in reality through its ontological commitment. It is not necessary and in general with respect to computing capacities also not feasible to capture reality as whole.
The explicit specification is commonly represented using a declarative formalism that in turn allows the sharing of knowledge between different agents. In this respect, the *universe of discourse* is the set of objects that are described in a particular ontology [103]. Considering multiple agents operating on a mutually agreed ontology, this conceptualization can further be used as a kind of *shared vocabulary*, leading to a common understanding of the domain. A definition that particularly puts ontologies into context with the field of Artificial Intelligence while also taking into account the meaning of terms can be found in [108]:

> *"[...]in AI, an ontology refers to an engineering artifact, constituted by a specific vocabulary used to describe a certain reality, plus a set of explicit assumptions regarding the intended meaning of the vocabulary words."*

According to this definition, it is not sufficient to solely provide a classification of terms, however, it is usually also necessary to describe the relationships between terms using explicit statements and thus specify their meaning. Especially when agents with different views of the world communicate with each other this explicit definition of the meaning of terms is useful: it needs to be possible to relate terms in the vocabulary to objects in reality (cf. Figure 2.2) and the common ontology in this case needs to include the meaning of the term so that it is clear to all communicating agents, which real-world item a particular term refers to. In other words, with respect to Figure 2.2 the representation of meaning in an ontology facilitates the transition from symbol (i.e., term) to object (i.e., item).
In this context, also the term *ontological commitment* again needs to be mentioned [103]: ontological commitment describes how terms in the ontology are related to each other and this way can be used to express the meaning of terms in the ontology. The ontological commitment may be viewed as constraints that are imposed on a concept in the ontology necessary to identify it as a representative for a particular thing in the world. It is desirable to use the least ontological commitment possible to support knowledge sharing between the associated actors to assure the broad applicability of the definitions in the ontology [103].
The article [118], finally gives a definition that puts the term ontology in relation to the Semantic Web (cf. Section 2.4.1), one of the application areas that significantly contributed to the recent popularity of ontological knowledge representation:

> *"Ontology is a set of knowledge terms, including the vocabulary, the semantic interconnections, and some simple rules of inference and logic for some particular topic."*

When it comes to the realization of ontologies, there exist several, mainly logical, languages that can be used. The reason for the prevalence of logical languages lies in their unambiguity, what leads to a simpler exchange of models and the possibility for logical inference [232]. One example of a logical ontology language is the Knowledge Interchange Format (KIF) which has its basis in predicate logic and was created to exchange knowledge between computer systems in a machine-readable format. With respect to ontologies it is mainly used as SUO-KIF in the context of IEEE Standard Upper Ontology (SUO) [227]. Another language for representation of knowledge in ontologies is Frame Logic (F-Logic). F-Logic also bases on predicate logic but extends it with constructs that facilitate the representation of object-oriented concepts (e.g., classes, inheritance) [232]. Finally, also the family of Description Logics can be used to define ontology languages. As already mentioned in Section 2.3, Description Logic can be seen as subset of predicate logic and as such achieves computational decidability with respect to logical reasoning. DL further provides a strict separation of classes and objects (i.e., terminological and assertional knowledge) and thus can be regarded as fitting for representing knowledge in ontologies, where also a separation of general and instance knowledge is needed. Nowadays, one of the most popular ontology languages that is capable of representing knowledge with the help of Description Logic is the Web Ontology Language (OWL) which is one of the languages defined for the so-called Semantic Web (cf. Section 2.4.1).
The author of [102] further describes the difference between an ontology and a knowledge base: the purpose of a shared ontology is usually to describe the vocabulary of a domain while a knowledge base may include more information about solving problems or answering queries. This means, usually a knowledge base is built for a particular application and often exclusively useful in the context of this very application. An ontology otherwise should be defined so that it describes things in reality in a more general way, making its definitions as general as possible. With the broad adaptation of ontologies in the Semantic Web, they are increasingly specified for particular applications and as such are too detailed with respect to this definition [216]. In this context, a classification of ontologies into different types has been identified in [107] which is further discussed in Section 4.1. As the ontology designed in this work is mainly focused on a particular application domain, the terms knowledge representation, knowledge base and ontology are subsequently used interchangeably.

2.4.1 Semantic Web

To describe knowledge as ontology in an information system is gathering importance in information science and Artificial Intelligence not least because of its application in the upcoming Semantic Web. The vision of a *Semantic Web* was first defined by Tim Berners Lee as a web in which computers become capable of analyzing content, relationships and actions between people and between computers [26]. As such, it may be seen as an extension of the World Wide Web (WWW) in a way that the information stored on the web can be annotated with metadata describing the meaning of the represented data in a machine-readable form. Available information should be equally interpretable for humans and machines and allow software agents to carry out sophisticated tasks on behalf of users on the web [27]. To realize the vision of the Semantic Web it is therefore necessary to build models that describe relationships between terms. Several standards have been developed by the World Wide Web Consortium (W3C) to facilitate the for-

mal description of information, and thus the generation of such models. Each standard defines a set of properties which can be used to structure and give a meaning to relations between data. With a grounding in computational logic and defined standard relationships, further, automated reasoning may be performed on the metadescription of data and thus an understanding of context as well as relations between information sources can be reached. Ontologies are used in the Semantic Web to describe relationships between data and as such represent knowledge.
Concerning information, one can generally distinguish between complete and incomplete knowledge about the described domain. A relational database usually adopts the *closed-world assumption*: Any data, that is not explicitly mentioned in the database is supposed to be not existing and thus a query for information that is not existing in the database evaluates to false. In contrast, there exists the concept of the *open-world assumption* (OWA): using this principle, if data is not explicitly mentioned, it may not be derived that it does not exist, however, its existence is merely unknown. This assumption is also used in the Semantic Web and the underlying Description Logic. Especially considering the World Wide Web as open and distributed system this assumption is reasonable, as new, yet unknown items may emerge anytime. A knowledge formalism used to describe an ontology in this context cannot expect all entities that exist to also be actually available and needs to facilitate inferences based on incomplete knowledge. On that account, if some element or relationship is not available in a particular vocabulary, it cannot be safely inferred that this statement does not exist at all: it could be possible that it is only not known to the current model at the present point in time, however is still present in another part of the web.
The Semantic Web stack as shown in Figure 2.3, pictures different languages for semantic knowledge representation on the World Wide Web. As a basis for the Semantic Web it shows a layer consisting of Uniform Resource Identifiers (URI) and Unicode. A URI is a global references to a resource on the web and provides unique naming and identifiers for resources on the web while Unicode assigns a unique number to every character making it possible to express the character regardless of the used system or language. Similar to the concept of primary keys in conventional databases, the URI as name of an entity on the web facilitates the unique identification of a resource in the Semantic Web. The XML language bases on URIs and as a very popular data-exchange language of the web is considered to represent the Semantic Web on the syntax level. XML can therefore be seen as one of the possible serializations of higher-level Semantic Web languages while XML namespaces are adopted to abbreviate names of resources and facilitate easier management of resource names. The first layer that adds meaning to the description of data on the Semantic Web is the RDF Core layer basically including the Resource Description Framework (RDF) language. A schema as well as an ontologies layer further represent standards that are used to describe the semantics of data on the web. The rules layer includes logical rules and the definition of a uniform rule format that needs to be defined in order to reach more semantic expressivity as well as higher levels such as trust and proof. As addition to the original Semantic Web stack attention also needs to be given to the associated query layer which builds on top of the RDF Core layer and defines how to extract information from the Semantic Web. The higher layers of the Semantic Web stack, trust and proof are still an active field of research and no recommendations by the W3C exist to date.
It is vital for the understanding of this work to provide a short overview of existing W3C stan-

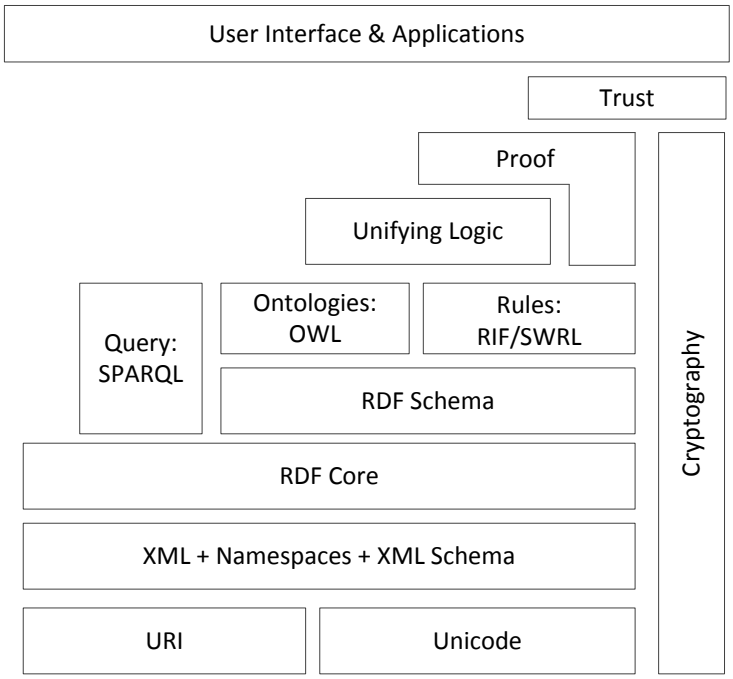

Figure 2.3: Semantic Web stack by the W3C [255], originally proposed in [25]

dards, and the type of information and knowledge these standards can express. The following sections describe technologies for different semantic layers based on information found in the standards' documents as well as the books [116], [4], [94], [63]. The focus of the language feature elaborations only lies on constructs relevant for the created ontologies presented in Chapter 4. A thorough description of all language capabilities can be found in the corresponding W3C standards.

RDF - Resouce Description Framework

When looking at the Semantic Web stack in Figure 2.3, the first and basic layer with respect to semantic interoperability is the Resource Description Framework. Originally RDF was designed to model metadata for XML documents (i.e., web resources), however, it was discovered very suitable for modeling real world relationships soon thereafter [236]. The RDF standard [249] was then developed to structure information and represent knowledge on the World Wide Web. As such, basically, RDF may be seen as data modeling language.
When breaking down a graph to its atomic elements, the basic building blocks are nodes and edges. Two nodes connected via an edge can in further consequence be interpreted as triples. RDF is designed to represent information in triples that follow an *subject-predicate-object* scheme (cf. Figure 2.4): A *subject* (`rdf:subject`) in this case is the entity that is being described. This resource is connected to an *object* (`rdf:object`) via a relation known as *predicate* (`rdf:predicate`). If several triples are connecting elements, an RDF instantiation may be

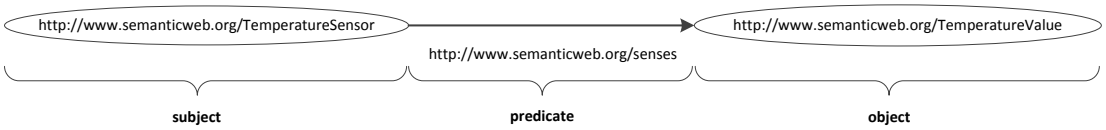

Figure 2.4: Example of an RDF triple

viewed as directed and linked graph. RDF differentiates between two types of elements, that can be connected via predicates which are resources and literals. RDF resources defined by the element `rdf:resource` are items in the world of interest, which may be web resources, but may also be things in the real world. Literals are lexical representations of values and may be plain strings with an optional language tag or typed, i.e., combined with a datatype URI [252]. Nodes in RDF are identified using URIs (cf. Figure 2.4) which facilitates the unique identification of a resource on the WWW. For that reason, other than in XML, the nodes referred in RDF triples do not necessarily have to be within the same document but may be located elsewhere on the WWW. A connection between different RDF graphs can this way be accomplished by referring to common elements identified through a single URI [4]. This way, an RDF triple graph may be seen as a special case of a semantic network. Further, agents operating on RDF may refer to the same resource by simply agreeing on a common URI for that resource. Anonymous nodes, hence, resources without associated URI are called *blank nodes*. For illustration purposes and brevity, XML namespaces are usually utilized to create a concise representation of resource URIs.
RDF defines a property for describing relationships named `rdf:type` which can be used to group entities that belong to the same type. For example, with respect to the example shown in Figure 2.4, it would be possible to describe concrete sensor values with a triple connecting them to the `TemperatureValue` resource element through the `rdf:type` property (cf. Listing 2.1).

Listing 2.1: Example of an RDF triple modeling a specific temperature value to be of type `swo:TemperatureValue`.

```
swo:TVal_20131127_0803h rdf:type swo:TemperatureValue
```

There exist different formats to represent the RDF language, so-called *serializations*. A commonly used form to describe RDF in the context of web-based applications is the RDF/XML syntax [251] which makes RDF readable for XML parsers. The popularity of this RDF serialisation stems from the intrinsic feature of most web-based systems to support the read or write of XML syntax. A disadvantage of this syntax is however, that RDF/XML is verbose and therefore the structure of the underlying RDF graph is often obfuscated and difficult to comprehend for the human observer. Otherwise, human-readable RDF serialization is the RDF/N3 notation: this format is a more compact way to represent RDF than the RDF/XML while still relying on the same abstract RDF model and therefore remaining machine processable. While the possibilities of N3 surpass the capabilities of RDF, there exists a subset of this language, the Terse RDF

Triple Language (Turtle) which is a popular format to express RDF models in a concise and readable form. Turtle is currently a Candidate Recommendation of the W3C (cf. [264]) and is expected to become a fully accepted standard in the near future. In this work, the RDF code examples are usually illustrated using Turtle while falling back on RDF/XML in some cases.
Physically, an RDF graph may be represented as a single file, or kept in an RDF triple store which is used to manage a great amount of triples. With this respect, a *triple store* can most likely be compared with a Relational Data Base Management System (RDBMS) while however information representation in a triple store is usually optimized for RDF data. Additionally to the capability of storing RDF data, triple stores often also provide the possibility to query for data through a query interface or support reasoning on information in case a more expressive form of RDF is stored. As a backend, a triple store may use a conventional RDBMS, however, by this time also storage backends tailored to the representation of RDF triples are available[2].
Summarizing, RDF may be seen as the syntax for the meaningful description of resources on the web. To interpret this syntax, it is necessary to further define a set of rules of how these resources may be combined with the help of a common vocabulary for logical dependencies. With such an extension of the RDF vocabulary, an ontology of resources throughout the World Wide Web may be created. To describe ontologies based on RDF, therefore, more expressive languages are needed, defining additional relationships and providing advanced possibilities to characterize resources. The W3C standardized ontology languages for this task are entitled Resource Description Framework - Schema (RDFS) and Web Ontology Language and the main language features necessary to create formal and expressive conceptualizations are elaborated in the following sections.

RDFS - Resource Description Framework Schema

The RDF Schema standard [250], can be seen as framework that adds capabilities to express the meaning of resources in an RDF graph. For that reason, this language defines concepts to describe the semantics of the subject-predicate-object triples of the RDF language. The definition of the schema itself is again realized as RDF graph, hence, no additional representation format is needed: metainformation for single RDF elements is simply added through triples using standardized relations and definitions from the RDFS vocabulary postulating a specific meaning. The language elements are described thoroughly in the RDFS standard which, as a common, standardized vocabulary allows a first set of inferences about relationships between RDF resources contributing to the understanding of the intent of a particular RDF element.
Most importantly, RDFS defines constructs to structure RDF triples in a hierarchical way and form a taxonomy. Therefore, it firstly provides the class `rdfs:Resource` which may be seen as the class of all things that are represented in the schema. Each RDFS class is connected to this class either directly or indirectly via a `rdfs:subClassOf` relation. To define a class, the RDF Schema provides the `rdfs:Class` element. A class in this respect can be seen as a set of instances which are `rdf:resources` of the same type. This class-to-instance relation is represented with an RDF triple connecting the class name with the resource through an `rdf:type` relation. Further, RDFS allows to define properties of classes which can be seen as instances of

[2]Jena TDB: http://jena.apache.org/documentation/tdb/

Table 2.3: Summary of important RDFS Language Constructs (excerpt)

RDFS construct	Description
`rdfs:Resource`	Class of all resources described in an RDFS vocabulary.
`rdfs:Class`	Class of all RDF classes.
`rdf:Property`	Class of all properties.
`rdfs:Literal`	Class of all literal values.
`rdfs:Datatype`	Class of datatypes.
`rdfs:subClassOf`	Subclass relationship between two classes.
`rdfs:subPropertyOf`	Subproperty relationship between two properties.
`rdfs:range`	Describes the type of the object in the triple connecting two resources through a property.
`rdfs:domain`	Describes the type of the subject in the triple connecting two resources through a property.
`rdfs:label`	A human-readable short description of a resource
`rdfs:comment`	A human-readable long description of a resource

a class `rdf:Property` [249]. Like classes, properties again can be related with each other by the `rdfs:subPropertyOf` relation: subproperties can be seen as specializations of superproperties and allow to represent more complex, hierarchically ordered relationships between classes. Properties can describe instances of an RDFS class in more detail and for this reason may have a domain and a range restriction, another important mechanism introduced by the RDFS vocabulary. The `rdfs:domain` restriction can be used to define a set of classes for a property. Each resource being at the subject position and using this property can then be inferred to be a member of these classes. The `rdfs:range` further describes the type of the values of a property: as with the domain statement, the range of a property allows to infer the type of resources connected through this property while being at the object position of the triple. RDFS further defines a class to describe simple literal values, `rdfs:Literal`, whose elements may be plain values or typed by being members of the `rdfs:Datatype` class. A summary of the most important language constructs of RDFS can be seen in table 2.3. Summarizing, RDFS allows the definition of a simple ontology providing different types of relationships among terms. With the application of the RDFS language, also first logical inference (reasoning) becomes possible.

OWL - Web Ontology Language

If more complex relations like existential and universal restrictions between classes or cardinalities of relations need to be expressed, the schema of RDFS does not suffice as this language only provides a limited set of constructs to semantically describe the concepts of a domain. To further detail classes and relations defined in RDFS, therefore its extension the Web Ontology Language [258] can be used. OWL is defined as a description language to represent meanings of resources on the World Wide Web and facilitates information integration and sophisticated reasoning on represented knowledge. Based on RDF and RDFS, OWL again uses triples for

data representation and supports the whole RDF/RDFS vocabulary, while extending it to model expressive semantic descriptions of concepts. The following paragraphs explain the different additional constructs and their relationships according to elaborations in the W3C OWL language reference (cf. [258]).
Also in OWL, there exists the concept of classes for which the `owl:Class` construct is provided. Members of an OWL class are entitled *individuals* and the group of individuals belonging to a specific class can be described by class descriptions that are used as basic building blocks to form a *class axiom*. OWL differentiates between necessary and sufficient restrictions that can be used to describe a class axiom. A *necessary class restriction* is a restriction that may be used to further describe the properties of individuals stated to belong to this class. This means, that members of the class have to fulfill the necessary restrictions so that the modeled ontology remains consistent. Otherwise, *sufficient class restrictions* may be used to determine membership of a class and in this respect to assign individuals to classes according to their properties and relationships to other classes. Necessary restrictions are modeled by an `rdfs:subClassOf` relation: this way, the class axiom allows to define the described class as subclass of the anonymous class of individuals identified by the necessary class axiom. Sufficient constraints are defined by the `owl:equivalentClass` construct, and as such all members of the anonymous superclass represented through the equivalence constraint can also be seen members of the defined class. These relations can be used to apply automated class membership reasoning for individuals which becomes possible for classes described with the help of a class axiom including at least one sufficient class restriction. This distinction mainly leads to two different types of classes in an OWL ontology, primitive and defined classes [125]: a *primitive class* is described by subclass axioms that may be used to manually create hierarchies or specify property relations of the particular class. The specification of a *defined class* otherwise includes an equivalence axiom and allows the reasoner to classify other concepts as subclasses as well as to reason about individual membership.
In contrast to RDFS, in OWL a class may be represented in several ways. According to the W3C OWL standard, there exist six types of class descriptions [248]: firstly, a class identifier can explicitly name a class by URI, the same way as a class is identified in RDFS. Further, a class can be described using an exhaustive enumeration (`owl:oneOf`) of instances of a class, thereby explicitly stating the individuals that belong to it. Another possibility is to describe a class by property restrictions, i.e., the membership of an individual to a specific class is defined by limiting relations to other classes and datatypes. Possible restrictions are firstly split into value and cardinality restrictions. *Value restrictions* can be realized by the constraints `owl:someValuesFrom`, `owl:allValuesFrom` and `owl:hasValue` and generally allow to restrict the relationship to individuals of other classes. In this case, `owl:someValuesFrom` represents the relation where each individual belonging to the described class needs to have a relation to at least one member of the connected class. This statement therefore represents the existential quantifier in predicate logic. The second value constraint is the `owl:allValuesFrom` construct: it denotes that all relations modeled with such a restricted property have to link to a particular class. For example, if an individual of the described class (subject) has a relation through the restricted property (predicate), the associated individual (object) has to be of the class type that is described in the `owl:allValuesFrom` relation. On

that account, this OWL statement is the equivalent of the universal quantifier in predicate logic. *Cardinality constraints* in OWL allow to restrict the relations of a described class with respect to how many relations are allowed through a particular OWL property. Cardinality in OWL can be described using three special statements: firstly, there is the `owl:maxCardinality` constraint, describing that there exist no more than the stated relations to individuals of the referred class. Further, the `owl:minCardinality` construct may represent the minimum of relations needed to a certain class and its individuals. Lastly, the `owl:cardinality` constraint is the combination of the two aforementioned cardinality restrictions and may be used to describe an exact amount of individuals connected to a class member via one specified property. Finally, logical intersection (`owl:intersectionOf`), union (`owl:unionOf`) and complement (`owl:complementOf`) of classes may also be used to define new classes.
Although the unique name assumption is often postulated in logical programs, OWL like Description Logics assumes a non-unique name assumption, what means that two different names do not necessarily imply two different individuals. For this peculiarity, OWL includes special constructs to express the relationship between items with same names which are `owl:sameAs` and `owl:differentFrom`. Also class descriptions may be explicitly stated distinct with the help of the `owl:disjointWith` statement. This way it becomes possible to represent that for the described domain no individual in the ontology may be member of the two disjoint classes at the same time.
OWL also provides a finer description of properties than is possible in RDF Schema by introducing a distinction between object and datatype properties. *Object properties* which are defined using the `owl:ObjectProperty` statement in this respect are used for relating different individuals while *datatype properties* identified through `owl:DatatypeProperty` describe relationships between individuals and data values. Also for properties connecting individuals there are specific constructs defined in OWL allowing to logically restrict the way individuals can be connected via the particular property. A list of these constructs together with a short description is presented in Table 2.4. Another special relation between properties that is supported in the OWL 2 standard [258] is the chaining of properties. These so-called *property chains* may be used to connect individuals via more than one property in a row. A simple property chain can be seen in Listing 2.2: the elements of this chain are the three object properties `producesEnergy`, `ofEnergyType` and `producesEnergyType`. Assuming three individuals X, Y and Z in the ontology related through the properties in a way that X is connected to Y through the `producesEnergy` property and Y is connected to Z via the `ofEnergyType` property, a reasoner can infer the third relation `producesEnergyType`, connecting the individuals X and Z directly.
The presented general information on OWL 2 reflects its utilization in this dissertation and highlights the constructs and statements that are needed to model the created ontologies. As OWL is a very powerful language it certainly has more capabilities than the ones mentioned at this point and an exhaustive discussion of the language can be found in the OWL 2 specification document of the W3C [258].

Listing 2.2: Example property chain in Manchester Syntax for OWL 2 [259]

```
ObjectProperty: producesEnergyType
  SubPropertyChain: producesEnergy o ofEnergyType
```

Table 2.4: Summary of property characteristics in OWL (excerpt)

OWL construct	Description
`owl:inverseOf`	The subject of this statement describes the same relation as the object only with interchanged subject and object individuals.
`owl:FunctionalProperty`	A property defined with this statement can be used to connect an individual in the subject slot to only one single individual in the object slot.
`owl:InverseFunctionalProperty`	A property defined with this statement can be used to connect an individual in the object slot to only one single individual in the subject slot.
`owl:TransitiveProperty`	This statement describes a transitive relationship.
`owl:SymmetricProperty`	This statement describes a symmetric relationship.

Manifestations of OWL

Observing the functions provided by OWL it can be seen that the language is designed to be mapped to a type of predicate logic. As already mentioned, OWL is an extension of the RDF and RDFS vocabularies and as such is capable of representing any model defined in one of these two languages. However, using OWL with no restrictions makes reasoning on this language undecidable. Therefore, it is necessary to define subsets of the complete language so that decidability is assured and automated reasoning is enabled. For this reason, the simple extension of the RDF(S) vocabularies termed *OWL Full* is usually not used. For OWL 1, the initial Web Ontology Language recommendation of the W3C [248], two sublanguages of OWL Full are defined, OWL DL and OWL Lite (cf. Figure 2.5). *OWL DL* can be seen as the most popular incarnation of OWL which essentially constrains the language to a form of Description Logic. This restriction brings about some essential qualities: while the limitation to FOL implies the exclusive usage of unary and binary functions, the limitation of OWL to DL restricts it to a decidable fragment of first-order logic and as such facilitates logical reasoning over explicitly modeled facts with the help of automated DL reasoning engines. A stricter sublanguage than OWL DL is *OWL Lite*: as a subset of OWL DL it is optimized for data processing. While OWL DL supports all constructs defined in OWL Full however restricts their usage, OWL Lite requires to only use specific functions in order to remain lightweight with low computational complexity and thus provide efficient means to operate on large bulk of data. However, mainly because of its limitations, OWL Lite is not widely used [114].
The current version of the Web Ontology Language is OWL 2 [258]. This language extends the original OWL 1 standard with additional constructs which among other things allow the construction of more complex logical statements resulting in the capability of describing more

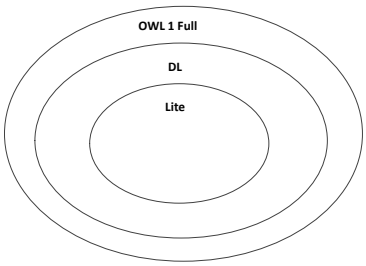

Figure 2.5: OWL 1 Full and its sublanguages DL and Lite

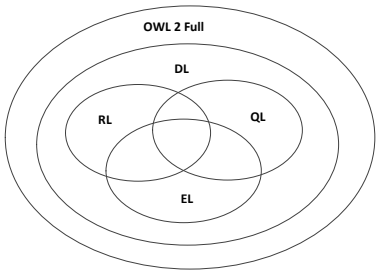

Figure 2.6: OWL 2 Full and DL as well as syntactic subsets of DL (profiles) [257]

expressive forms of Description Logic than OWL 1. This latest version of the OWL standard further defines different profiles on OWL DL to create logical subgroups of the language and allow efficient reason on the modeled ontologies (cf. Figure 2.6). While the new version of OWL omits the definition of OWL Lite, the sublanguages of the initial standard OWL 1 DL and OWL 1 Lite may also be seen as profiles of the OWL 2 DL vocabulary [260]. All of these profiles put syntactic restrictions on OWL 2 DL to lower the computation complexity and this way make the language more suitable for certain applications. The *OWL EL* profile, for example, is appropriate whenever an ontology includes very large class and role hierarchies however the amount of used OWL features is limited [122]. This profile is especially useful in the biomedical domain where ontologies often span several thousands of concepts and properties. Further, *OWL QL* is designed for applications involving large amounts of data. Therefore this sublanguage is especially suitable for tasks involving the representation of data that has originally been stored in databases or the integration with conventional relational database systems. As a third variant, OWL 2 provides the profile *OWL 2 RL* which is especially designed for further knowledge description and reasoning with a rule language. This way, it should become possible to use OWL also in systems with rule-based inference and furthermore facilitate the interoperability between knowledge representation languages [122]. As such, this profile puts additional syntactical restrictions on OWL DL to be as expressive as possible while allowing inference to be realized with a rule processing system (i.e., a rule-based reasoning mechanism) [116]. The ontologies constructed in this work are OWL 2 DL ontologies that are not restricted to a particular profile in order to use the full expressivity of the language. Therefore, whenever OWL is mentioned, it is actually referred to the OWL 2 DL vocabulary.

Semantic Web Rules Support

While using OWL for description of content on the Semantic Web leads to a significant improvement of machine-interpretability and machine-independent formulation through an expressive logic-based language, there still exist limitations. For that reason, rules may be seen as a complementary addition to the constructs of the OWL standard to overcome limitations in expressivity and make the language more versatile, especially with regard to describing properties and their characteristics [126]. A rule extension of OWL can further be seen as bridge towards F-Logic-based ontology languages [121] supporting the integration of ontologies described in

either of these approaches.
Although there exists a variety of different rule language incarnations for OWL, one of the most important and widely used is the Semantic Web Rule Language (SWRL) [253]. An example of an SWRL rule is illustrated in Listing 2.3[3]. The example rule is a simple chaining rule which similarly to a property chain allows the reasoner to infer the `isParameterOf` relationship found in the rule head in case restrictions in the rule body are met. Statements like `?a` describe variables which can be seen as placeholders for concrete individuals. According to [126], *SWRL rules* are an addition to OWL based on OWL DL and Horn clauses extending OWL in a syntactically and semantically coherent manner by describing rules as additional OWL axioms. Horn clauses represent implication rules consisting of a rule head and body [116]: the rule body in this case is the condition that needs to be met so that the implication modeled by the rule can be drawn. The rule head further describes the conclusion that can be inferred. An SWRL rule may thus be seen as a simple "if" statement in a programming language. In order to preserve the quality of automated reasoning, a logical rule system for OWL DL naturally also needs to take decidability into account. So-called *DL-safe rules* allow the description of complex relationships between entities in the ontology without losing the property of decidability [169].

Listing 2.3: Example SWRL rule which has a number of statements in the rule body and a single property as rule head.

```
isParameterOf:

        Absorptance(?a1), Building(?b1),
        Construction(?c1), Space(?sp1),
        Surface(?su1),
        containsSpace(?b1, ?sp1),
        hasDefinedAdjacentSurface(?sp1, ?su1),
        hasDefinedConstructionType(?su1, ?c1),
        containsAbsorptance(?c1, ?a1) -> isParameterOf(?a1, ?b1)
```

Because of the variety of different logical rule languages for the Semantic Web and no fully agreed-upon standard, recently the *Rule Interchange Format* (RIF) has been developed: RIF is proposed by the W3C as standard to exchange logical rules between different rule systems and languages [262]. As such, RIF may be seen as a part of the rule layer in the Semantic Web stack (cf. Figure 2.3) describing interoperability for Semantic Web implementations using different rule languages. RIF covers production rules and logical rules at the same time and as such facilitates the modeling of business processes in combination with OWL ontologies. Although this layer of the Semantic Web stack is not yet as thoroughly researched, developed and accepted as the other described W3C standards, it is expected to gain importance in the future.

Reasoning and Inference in the Semantic Web

Reasoning with respect to knowledge representation allows extending explicitly stated information with knowledge that is implicitly found in the semantic description of modeled facts. While in conventional database systems, the implicit relations between elements in the database are

[3] In this work, the syntax presented in [96] is used for illustration of rules. This syntactic model is an extension of the Manchester Syntax for OWL 2 [259] and allows the concise presentation of SWRL rules.

extracted with complex queries representing a lot of the elements' semantic interrelation, in the Semantic Web there exists the possibility of representing this information already in the knowledge store and use reasoning to make it accessible. What can be accomplished as a result is the more expressive representation of data, information and ultimately knowledge using reasoning to infer new dependencies between data out of a pile of explicitly stated facts. The restrictions that are imposed on basic RDF through the ontology language levels RDFS and OWL allow reasoners to infer additional knowledge of entities through their definition in the ontology language. Therefore, in order to draw logical conclusions, a richly axiomatized knowledge base is indispensable. As OWL DL builds on Description Logic, reasoning can be used to make inferences possible that are stipulated through this logic like subsumption reasoning, consistency checking, satisfiability and class membership but also class disjointness and equality (cf. Section 2.3). A logical reasoner in this case is a piece of software that implements algorithms which can perform the mentioned reasoning task on the logic automatically and can in further consequence be used to infer additional dependencies between triples in the knowledge store. Today, several different implementations of semantic reasoners for OWL DL exist, with FaCT++ [237], Pellet [219] or RacerPro [111] being some very well known examples. A comprehensive list of currently available OWL reasoners can be found in [261]. Most of the current Description Logic reasoners are based on tableaux algorithms which are very successful for Description Logic reasoning. Because of the wide variety of implementations, it differs from reasoner to reasoner which inferences, hence, expressivity of DL is supported and, for example, if additionally reasoning for rules or datatypes is available. On that account, it is highly dependent on the problem at hand as well as the complexity of the ontology to operate on, which reasoner is most appropriate. In the current work, the Pellet reasoner [219] is used. There were several factors that led to the choice of this reasoner: firstly, the reasoning engine supports OWL 2 reasoning including all available constructs which is not necessarily the case for other implementations. Further arguments for this reasoner are that Pellet provides reasoning for DL-safe SWRL Rules and is available through an open source license[4].

Querying for Data: SPARQL Protocol And RDF Query Language

Although reasoning already takes a large part of query answering from the query language, apart from the inherent logical possibilities of the different layers of the Semantic Web stack, it is still needed to query and aggregate data and information from triple stores, as not all facts may be directly modeled into the knowledge store either because of brevity or because of limitations in expressivity of the used language. To effectively extract information from an RDF triple store, a query language is needed that is capable of querying RDF. Therefore, the SPARQL Protocol And RDF Query Language [256] has been developed which in short is the same for an RDF triple store as the SQL language for relational databases or XQUERY to an XML database: It facilitates the selection of a specific group of triples from the entirety of the stored RDF triples. As in RDF information is stored using a directed graph, SPARQL is obviously a graph-based query language. As such it allows the definition of graph patterns that should be searched in the graph. SPARQL provides two necessary keywords which are `SELECT` and `WHERE`: the

[4]Project homepage: http://clarkparsia.com/pellet

Table 2.5: Syntactic query elements in SPARQL (excerpt)

SPARQL element	Description
PREFIX	Declares the namespace for elements in the query.
SELECT	Determines which elements should be returned.
WHERE	Query including combinations of triple patterns.
OPTIONAL	Includes further triple patterns in the query that may be optionally available.
FILTER	Allows the definition of filter conditions for the defined query.
BIND	Makes the association of a variable with a (manually defined) value possible.
GROUP BY	Enables the grouping of result sets and the application of aggregate functions.
ORDER BY	Facilitates the ascending or descending ordering of the result.
CONSTRUCT	May be used instead of the SELECT statement to generate an RDF graph as result.
ASK	May be used instead of the SELECT statement to return a boolean answer as result.

SELECT statement allows to define which elements should be returned by a query while the graph pattern itself is defined in a WHERE clause that may be extended with an optional part indicated by the element OPTIONAL. This pattern is then searched in a given graph. Among other functions, SPARQL may construct a sub-graph of the knowledge base as answer for which the CONSTRUCT element is available. It is further possible to use SPARQL in order to query multiple triple stores at once, which can be expressed in the query via the PREFIX element. As representation syntax, SPARQL uses the Turtle [264] RDF serialization syntax.
The SPARQL query language is available in Version 1.0 and 1.1 with the newer edition significantly raising the expressivity of the language with additional keywords and capabilities. In this work, SPARQL Version 1.1 is used to retrieve shorter queries and to also exploit newer capabilities of the query language. Table 2.5 summarizes the most important SPARQL constructs that are used in this work. Initially, SPARQL was designed only for querying an RDF store, while updating the RDF store has only been added later through SPARQL Update: this extension provides possibilities to also update the triple store adding a vital functionality of any query language. While the SPARQL query language has already been a W3C recommendation for quite a while [256] the SPARQL Update extension just became a recommendation recently [263]. Anyway, a full coverage of SPARQL would go beyond this work and the reader is referred to [116] and [68] for more information about the syntax and the usage of this Semantic Web query language.

2.4.2 Ontology Design Methodologies

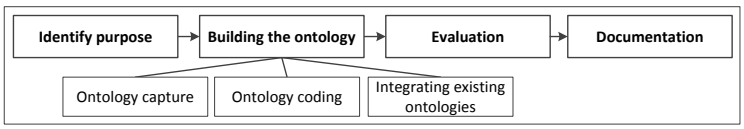

Figure 2.7: Methodology of Uschold & King [242]

The design of an ontology can be supported by several methodologies while there however does not exist a single ontology methodology at the moment which is commonly agreed. The article [84] gives an overview of the most prominent approaches that can be found today. Among the methodologies reviewed in this article, two are considered relevant for the present work, which are the methodology by Uschold and King [242] and the methodology by Grüninger and Fox [104]. Uschold and King provide guidelines for the creation of an ontology that is split into four main steps (cf. Figure 2.7). Mainly, the presented framework organizes the ontology design and creation process by defining a stage where requirements are captured, an ontology construction phase and evaluation as well as documentation step. Regarding the method described by Grüninger and Fox, six stages can be identified for the specification of an ontology (cf. Figure 2.8). Ontology development is firstly motivated by scenarios described as stories or usage

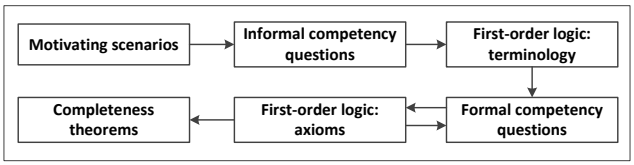

Figure 2.8: Methodology of Grüninger & Fox [104]

examples. From these examples, in a second stage informal competency questions are derived. These competency questions are in a further step formalized with the respective logic underlying the chosen ontology representation language to retrieve formal competency questions that can in turn be used to define formal ontology axioms and concepts for the defined ontology. Finally, the methodology provides an evaluation stage for the ontology specification in which the scope and completeness of created axioms are tested with the help of completeness theorems.
At this point, also the Ontology Development 101 process [179] has to be mentioned as it proposes another simple ontology-development methodology (cf. Figure 2.9). The described guideline is targeted at declarative frame-based systems and again describes several steps that should be followed to successfully design and develop an ontology. While it widely follows the first two approaches, it provides a more practical approach, describing guidelines for the actual implementation of an ontology.
Although not directly used in this work, METHONTOLOGY [83] can be identified as another popular ontology development methodology. The procedure identifies different states and activities that are to be carried out when building an ontology: as observed in [84] it differentiates be-

tween project management activities like planning, control and quality assurance, development-oriented activities like the specification, conceptualization, formalization, implementation and maintenance of an ontology, and supportive activities such as knowledge acquisition, evaluation, integration and documentation. As before, several stages are run through during ontology im-

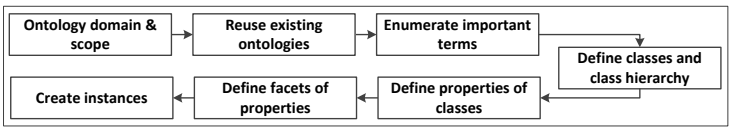

Figure 2.9: Methodology of Noy & McGuinness [179]

plementation covering certain activities. The methodology follows a life cycle centered around evolving prototypes, facilitating the growth of the ontology according to its needs [83].
As none of the just mentioned methodologies is predominantly used for the development of ontologies, often also methodologies covering different aspects are combined to fully describe the ontology design and construction process [43]. This dissertation follows an approach combining ideas and elaborations from several mentioned approaches. The process generally follows the guidelines described in [242] and therefore the shown workflow of the first mentioned source (cf. Figure 2.7) is used as a framework for the following ontology development process:

- **Identify Purpose:** Firstly, as already mentioned, it is typically necessary to motivate the creation of the ontology and describe its future use. As the ontology is created in the context of a smart home system, its purpose can straightforwardly be retrieved from an initial requirements analysis of the system. For this reason, the use cases discussed in this chapter are identified as the first stage of the followed ontology development methodology. Corresponding stages from the methods described in [104] and [179] are taken into account as well and informal competency questions are extracted from the initial smart home use cases that further can be used to identify the purpose and determine domain and scope of the created ontology.

- **Building the Ontology:** The second stage in this methodology is concerned with the creation of the ontology, and according to [242] is split into the three subgroups ontology capture, ontology coding and integrating existing ontologies. In the *ontology capture* phase the identification of key concepts, their description, associated terms and the relationships between different concepts have to be defined. Further, the step *ontology coding* is related to the representation of these identified concepts in a concrete ontology language, e.g., the Web Ontology Language. The last part of this methodology phase is the *integrating existing ontologies* step which is concerned with the evaluation of different already defined ontologies for reuse and integration in the newly created conceptualization. In this dissertation all three substeps of this phase are covered in Chapter 4.

- **Evaluation:** In [242] not much information is given about the evaluation step. In the present work, evaluation is based on requirements and competency questions. Competency questions may act as frame of reference to assess the capabilities of the created

ontology [98]. For this task, the questions that are identified in the requirements analysis are classified according to their generality (i.e., stratified [104]). As a further evaluation step, the natural language competency questions are represented formally with the help of Description Logic queries and the SPARQL query language and it is shown how these questions can be answered with constructs defined in the created ontology. A detailed analysis is provided in Chapter 5 which corresponds to the evaluation phase of the pursued methodology.

- **Documentation:** This final step is also covered in Chapter 4: besides describing design decisions and the detailed analysis of the concepts and their relationships it also highlights the usage of the main concepts of created ontologies and gives examples of possible fields of application. Therefore, the documentation step is implicitly provided by the extensive description of the ontologies' main elements and relationships in the part covering ontology specification.

CHAPTER 3

Ontology Requirements Analysis

In order to identify the requirements of the envisioned smart home system, it is necessary to capture its functionality. In complex technical systems, use cases are common means to gather information about requirements and functionality. In [52], the following definition of a use case can be found: *"A use case is a description of the possible sequences of interactions between the system under discussion and its external actors, related to a particular goal"*. Used as a first concrete document of a project, a use case therefore creates a clear and concise view of a system and its functions without explicitly defining the internal structure. It thus allows to describe the system for a variety of stakeholders and other interested parties that do not necessarily need an extensive technical background in order to understand the dependencies and scenarios described in the use case. The focus of these documents further lies on specifying the behavior of a system with regard to actions executed by an actor. After definition and implementation of the system under discussion, previously defined use cases may additionally be utilized to validate it and ensure that the system meets the postulated requirements.
Considering the smart home system including the multi-agent system and the knowledge base as system under discussion, the actor that interacts with it may certainly be human. However, when particularly focusing on the smart home knowledge base, often another subsystem like the agent system or also time may be identified as actor. This peculiarity regarding use cases results from the system's proclaimed autonomous and proactive behavior.
According to [214], there are different presentation styles for use cases that reach from informal textual descriptions to pseudocode form. In [52], two styles are especially emphasized, termed as "casual" and "fully dressed" use cases. *Casual style* use case descriptions are written in simple paragraphs and describe the dependencies in full sentences allowing to easily read through the use case descriptions written as continuous text. While the casual style may not completely present all conditions that need to be taken care of, it generates a general definition of a use case's focus and usage. On the other hand, *fully dressed* use case descriptions illustrate the use case in a more rigorous way using formal templates for use case description and pointing out trigger conditions, pre-conditions and other template header information [52]. The author of [52] does not favor one over the other description and presents both types of use case represen-

tations as equally reasonable depending on the situation in which they are used: while the size of the project may indicate one particular form also the stage of project development may advocate one representation over the other.
Regarding ontology development, use cases may be compared to the initial step of the methodology described in [104], i.e., motivating scenarios. These scenarios clarify why an ontology is needed in the described context and allows the definition of a set of queries that need to be answered by the ontology. These queries that can be found as initial step of most reviewed ontology design methodologies (e.g., [104, 242, 179]) are termed competency questions. According to a definition provided in [241], *competency questions* may be considered expressiveness requirements in the form of questions that are derived from motivating scenarios, i.e., use cases. In other words, the motivating scenarios are broken down to a set of questions that the defined ontology needs to be able to answer. On these grounds, competency questions denote valuable means of evaluation prior a system's complete implementation to assess that an ontology satisfies the requirements of a particular operational area.
The method described in [104] further suggests a layered approach when defining competency questions with lower-level questions acting as foundation for higher-level questions. The methodology described in [179] also proposes competency questions as a starting point when developing a new ontology. According to this approach, competency questions do not have to completely cover the domain to be modeled and mainly serve as a pointer to important concepts that need to be described in order to cover a particular field.
As a first step to describe the requirements of a smart home knowledge base, therefore, general use cases for a smart home system are considered. In further consequence, from general smart home use case scenarios, competency questions are obtained that are considered a first requirements description for the ontology to be defined. In Chapter 5, these competency questions are then again used to evaluate the designed ontology and ensure that the postulated requirements for a smart home knowledge base are met. The following sections list the derived questions with the respective use cases they can be associated with. Afterwards, a grouping of identified questions into particular domains of interest is presented in Section 3.2. This grouping can be seen as the first step of the ontology development process determining the domain and scope of an ontology, as described in [179].

3.1 Use Case Scenarios

The general smart home use cases presented in this chapter have to a large extent been developed as a requirements analysis for the ThinkHome project (cf. Section 1.2). Other works that emerged from that project, particularly [199], already extensively cover this general set of smart home use cases with respect to general system functionality. As this analysis was jointly conducted with the author of the present dissertation, in this respect necessarily some content of the two works overlaps. However, while in [199] the use cases are presented as "fully dressed" representation, in the work at hand that level of detail is not necessarily needed. Therefore, to keep the elaborations concise, the "casual style" describing single use case scenarios as paragraph style textual descriptions is used in the following sections. This type of presenting the scenarios

is also consistent with the initial step of describing usage examples as proposed in the ontology development methodology of [104]. The scenarios are further used to identify competency questions in order to specify an ontology as knowledge base for the envisioned smart home system. The concise description just outlined is sufficient for the creation of a knowledge base, as the focus is the representation of knowledge for a smart home system rather than a steady system operation that has to cope with different error scenarios. The main outcome from the requirements analysis for an ontology are therefore the competency questions which particularize its needed functionality and can be directly retrieved from the described scenarios. For a detailed smart home use case discussion in the context of the ThinkHome system architecture focusing on a multi-agent-based control system, the reader is referred to [199].

The casual use case type described in [52] provides use case name, scope, level, context and primary actor as main fields. The *name* of the use case describes what has to be accomplished and as such can be seen as the goal to be reached. At the start of every use case investigation stands the *scope* analysis that describes what is inside the system and needs to be cared about and what is outside of the system and has to be interfaced with [214]. These boundaries can be found when actors in the system are identified. Therefore, as a first reference point a *primary actor* needs to be provided. There may be additional stakeholders and actors apart from the primary actor, which are however not necessary for the interaction of the knowledge base with the multi-agent system. Actors may be anything that interfaces with a system and as such can be people, other software or hardware devices as well as networks or data stores [214]. As already pointed out in the introduction of the ThinkHome project (cf. Section 1.2) the system is split into two main system parts, the multi-agent and the knowledge base system. Both parts can be seen as "system actors" while also other system and non-system actors (e.g., actuators, sensors, inhabitants, time) may exist. According to the proposed system architecture, the only part that interfaces with the ontology is the multi-agent system and therefore can be seen as primary actor for the knowledge base with regard to the presented use cases. The *context* of a use case further describes its usual function and occurrence in the process flow of the system and how the particular case contributes to the achievement of global system goals.

The goals of use cases may be ordered on different levels: the *level* defines where the present scenario can be found with respect to its generality or detailedness. Three different goal levels are identified in [52], which are subfunction, user goal and strategic level. Use cases found at the *subfunction* level involve goals that lead to the achievement of a more complex user-goal-level use case. In this respect, competency questions may be seen as subfunction-level use cases as they represent requirements for the knowledge base that need to be fulfilled to reach a particular user goal. Further, a *user goal* is an elementary task that the primary actor wants to be performed by the system. A sufficiently large number of user-goal-level use cases can be seen satisfactorily to fully describe the functionality of a system. Higher level use cases are called *strategic* use cases and are composed of several user-level goals. Strategic use cases mainly show the context in which system services have to operate while representing an outline for lower-level use cases [52].

For the present purpose, user-goal-level use cases are of main interest in order to describe motivating scenarios for the development of a smart home ontology. While in the course of the requirements analysis for the ThinkHome system also two strategic use cases have been defined

(i.e., assure energy efficiency and user comfort, cf. [199]), in the context of accumulating requirements for a smart home knowledge base, these use cases are not of immediate importance as no additional competency questions arise from the consolidation of user-goal-level use cases. For this reason, only user-level use cases are discussed at this point.
The following definitions of scope, level and primary actor are generally valid for all presented use cases and therefore mentioned only once:

Scope: Knowledge base (KB) subsystem
Level: User-goal level
Primary Actor: Smart home system/multi-agent system (SHS)

The use cases are consecutively numbered with the prefix TH standing for ThinkHome. Competency questions are listed in the order they can be extracted from the use case description.

3.1.1 Use Case: TH01 - Assure Air Quality

Context of Use

> *The smart home system (SHS) automatically preserves the air-quality in a building. It periodically receives the current air quality from the automation system and validates it against reference values derived from common standards (e.g., the American Society of Heating, Refrigerating and Air-Conditioning Engineers (ASHRAE) [5]). In this context, the SHS also evaluates if user-defined air quality preferences exist and uses them. When air quality drops below health or manually defined comfort standards, the SHS retrieves means to reach a better air quality from its knowledge base (KB). From this variety of facilities and processes, the SHS chooses the most energy efficient process. The SHS must ensure that the selected process is executable at the moment concerning the current internal and external world states of the home. After choosing the best alternative, the smart home system takes care of the execution of the actions belonging to the selected process.*

Use Case Description

To preserve the air quality, the actor, in this case the multi-agent system, has to perceive the environment. Therefore, the knowledge base needs to provide a possibility to describe sensor values like current air quality or instantaneous occupancy in the building. After receiving a new sensor value, the SHS has to store current sensor values in the KB. The KB further needs to provide an occupancy schedule that informs the SHS about predicted occupancy times in different parts of the building. For that reason, the KB also needs to model the basic layout of the building. In case no individual comfort preference schedule exists or an unknown person (i.e., guest) is present in the building the SHS assures that general guidelines on air quality are met. Therefore, the KB needs to represent standard profiles and preferences following globally recognized standards (e.g., ASHRAE Standard 55-2010). The KB further should be able to represent individual user comfort preferences for air quality. If the SHS detects that air quality is not within the range described in guidelines or individual comfort preferences, it needs to choose a process that improves the air quality. The KB in this case needs to provide suitable

processes. For a process apart from the process type the KB also has to model its associated facilities. If more than one process to reach a better air quality exists, the SHS needs to be able to decide which one is the most energy efficient. Therefore, the KB further needs to represent energy properties of facilities. The SHS needs information if a process is dependent on the weather state. Further, to let the SHS assess the exterior state, the KB needs to supply the current weather situation and the knowledge if the sensed situation is suitable for natural ventilation.

Competency Questions

1. What is the current air quality value in room X?
2. What is the current occupancy value of room X?
3. How can a new value for sensor X in room Y be stored?
4. Which occupancy schedules exist for room X in building Y?
5. How is building X divided into spaces/rooms?
6. What is the minimum air quality that has to be assured according to standards?
7. Which user preferences can be obtained?
8. Which user preferences for air quality exist?
9. Which processes can improve air quality in room X?
10. Which facilities are involved with process X?
11. Is process X dependent on the weather situation?
12. What is the current weather situation?
13. Does the current weather situation facilitate natural ventilation?

3.1.2 Use Case: TH02 - Assure Thermal Comfort

Context of Use

> *The smart home system permanently and automatically aims at ensuring thermal comfort. Again, optimal thermal comfort conditions may be achieved through considering user preferences as well as commonly agreed standards on thermal comfort such as ISO 7730 [135]. According to the mentioned standard, there exist several prevalent factors that influence thermal comfort, with temperature and relative humidity being the ones that are generally measured and controlled. Processes that manipulate these thermal comfort parameters like heating, cooling or humidification are mainly energy-intensive tasks representing a great part of the entire energy consumption of residential homes [15]. Comfort conditions are further dependent*

on the actual and scheduled occupancy and the current state of the home or a specific room. For certain control approaches, it is therefore necessary to take into account building structure (i.e., thermal inertia) and weather conditions. The Smart Home System (SHS) periodically monitors the current temperature and humidity of different parts of a home through the automation system. Under consideration of current and expected future occupancy and related preferences of occupants, it initiates actions that influence the temperature or humidity in a room. In particular, the SHS selects the most appropriate control possibility among a set of different heating/cooling or humidification options, decides on the most appropriate point of time to start the selected process and takes care of the execution of the actions belonging to the selected process.

Use Case Description

When the SHS queries for optimum thermal comfort settings, the KB needs to represent guidelines and personal user preferences regarding thermal comfort. The SHS further evaluates at which times comfort temperature and humidity need to be assured. The KB therefore has to provide an occupancy schedule that informs the SHS about predicted occupancy times in different parts of the building. For that reason, the KB needs to represent the basic layout of the building. For calculation of the optimum thermal comfort preparation time of different rooms, the SHS queries for room characteristics. In this case, the KB has to return properties such as the metrics of a room (size, volume), adjacent rooms (e.g., to take into account heat exchange), the U-value of an adjacent wall or alternatively wall materials and wall thickness as well as how many walls face the exterior. As soon as the preparation time to reach thermal comfort is known, the SHS schedules the most energy efficient processes to reach the desired state. The KB therefore needs to model the processes that can be used to change thermal comfort parameters (i.e., temperature, humidity) in particular areas of the building (e.g., rooms, spaces). If the SHS considers to schedule passive control strategies that do not rely on energy-intensive facilities and involve processes like opening the window or raising the shutters it firstly has to assess if these processes are possible. For the two mentioned processes for example, the KB has to provide the information if there exist operable windows/shutters in a room. Then, also the representation of current weather conditions may be necessary for the consideration of particular processes like opening the window. Further, the orientation of a wall facing the exterior is of importance to take into account solar heat gain when raising shutters. If the SHS recognizes a room occupancy value change from occupied to non-occupied, it needs to know which temperature to choose as setback temperature. In this case, the KB has to store different temperature values depending on situations and user preferences.

Competency Questions

1. What is the standard comfort temperature/humidity?
2. What is the standard comfort humidity?
3. Which user preferences for comfort temperature exist?

4. Which user preferences for comfort humidity exist?
5. Which occupancy schedules exist for room X in building Y?
6. How is building X divided into spaces/rooms?
7. What is the floor area of room X?
8. What is the volume of room X?
9. Which rooms are adjacent to room X?
10. Which walls are adjacent to room Y?
11. What is the U-value of wall X adjacent to room Y?
12. How thick is wall X adjacent to room Y?
13. What are the materials of wall X?
14. Does wall X face the exterior environment?
15. What is the current state of the indoor environment?
16. What is the current temperature value for a room?
17. Which environmental parameters are influenced by process X?
18. Which processes can be used to increase/decrease the temperature of room X?
19. Which processes can be used to increase/decrease the relative humidity of room X?
20. Do current weather situation and room characteristics (i.e., controllable windows, shutters) facilitate passive processes?
21. Which processes are dependent on the weather situation?
22. What are the current and near future weather conditions?
23. Does the current weather situation facilitate natural processes?
24. Does wall X have window openings?
25. Which windows of room X can be controlled by actuators?
26. Do controllable shutters exist for room X?
27. What are the dimensions of window X?
28. What is the orientation of wall X adjacent to room X?
29. What is the current occupancy value of room X?
30. What is the setback temperature for the building/a space/a room?

3.1.3 Use Case: TH03 - Assure Visual Comfort

Context of Use

> *The smart home system permanently and automatically aims at ensuring visual comfort. When rooms are occupied, the SHS compares visual comfort preferences against the current data retrieved from the automation system and initiates adjustments to lighting/shading equipment if necessary. When rooms are unoccupied, minimum energy consumption of the lighting system is ensured by the SHS. The SHS can obtain further information regarding applicable processes and available facilities from the knowledge base (KB). To establish visual comfort, the SHS chooses the most energy efficient control option and takes care of the execution of the actions belonging to the selected process via the automation system.*

Use Case Description

If no user preferences are defined the SHS queries the KB for recommended lighting levels retrieved from guidelines or standards (e.g., IES Lighting Handbook [128]). If the SHS detects the need to influence lighting conditions, the KB further needs to represent processes in the building that are capable of this task. The SHS chooses the process that minimizes energy required to illuminate a room for example through utilization of the current weather situation (e.g., sunlight). If the SHS directly queries the KB for devices (e.g., luminaires, shutter) in case it already knows the available processes, the KB has to return a facility's description and its current and possible states. This way it is possible to, for example, retrieve blind/shutter positions or artificial lighting sources and their modes of operation. If the SHS directly requests a facility to change the environmental condition, the KB additionally needs to provide the association of the facility to a room. Each user in the smart home may have individual preference settings stored on the facility level. In this context, it is also possible to have time-based preferences defined for a room (e.g., different lighting scenes for different times of the day or days of the week). Alongside manual definition it is also possible that the SHS recognizes the schedule through previously detected user habits. As a consequence, the KB has to be capable of representing manually defined and automatically retrieved schedules and their association with a particular room. Occupancy representation on a room level is again necessary, so that the SHS can conduct its energy efficiency strategies.

Competency Questions

1. Which manually defined visual comfort schedules for room X exist?
2. Which recommended preferences for lighting exist?
3. What is the current value of illuminance sensor X?
4. Which processes can be used to increase/decrease lighting of room X?
5. Which facilities are involved with process X?

6. What is the maximum amount of energy consumed by facility X?
7. What is the current position of blinds/shutters in room X?
8. What is the current value of artificial lighting source X?
9. Which modes of operation exist for facility X?
10. Which lighting facilities are installed in room X?
11. Which manually defined comfort schedules for facility X exist?
12. Which automatically created visual comfort schedules exist for room X?
13. Which automatically created schedules exist for facility X and day Y?
14. What is the near future state of the indoor environment?
15. When will room X be occupied in the near future?

3.1.4 Use Case: TH04 - Assure Energy Optimized Operation of Household Appliances and Entertainment Devices (Energy Consumers)

Context of Use

> *The smart home system optimizes energy consumption of household appliances and entertainment devices when they are idling or not required immediately. When a room is unoccupied, the system identifies energy consuming devices, such as hi-fi, TV, printers that are currently unused. The SHS also verifies the occupancy schedule of the room in question. When no occupancy is expected in the near future, the system initiates actions to disconnect selected equipment from the power supply to save energy. It takes care of the execution of the actions belonging to the selected process via the automation system.*

Use Case Description

Firstly, the SHS detects idle rooms and facilities and assesses if a room is expected to become occupied in the near future. For this task, again a global notion of occupancy is essential. The KB in this case has to represent current occupancy as well as predicted future occupancy and user-defined occupancy schedules. If the SHS retrieves predicted occupancy or a manually entered schedule from the KB it can assure if it is safe to turn off facilities in a room in order to save energy. If no occupancy is expected, the SHS can turn off devices provided that they are unused and do not need permanent power supply. Alternatively, considering energy facility control, the SHS may use manually entered device schedules to turn on/off devices according to pre-defined user preferences. For this case, the KB has to be able to provide user-defined schedules on a device level. Also an automatically created device schedule may exist. If the SHS needs to calculate the current and future energy consumption of facilities, the KB firstly needs to provide their possible modes of operation. So that the SHS can perceive the current state of the

environment and possible future energy savings, the KB further has to model the current state of a device how much energy it presently consumes and how much it needs in its different modes of operation. The SHS further needs to get available command options from the KB that realize state changes at devices. The SHS can use the representation of the devices' energy demands in their different states to follow different energy strategies: if the energy demands of different facility states do not differ too much and the device is needed again in the near future, a facility may be put into stand-by instead of completely turned off thus ensuring user comfort. Another precondition for the SHS to consider is if a device needs to be supplied with energy permanently or if it can be turned off safely. This need for energy supply can be determined by the type of the device (e.g., refrigerator) or by a previously entered user schedule (e.g., programmed video recorder). The KB has to provide a representation of energy facilities that is capable of modeling devices and their characteristics in the necessary granularity.

Competency Questions

1. Is room X currently occupied?
2. Which automatically generated occupancy schedule exists for room X?
3. Which manually defined occupancy schedules exist for room X?
4. Which facilities are currently consuming electric energy?
5. Which facilities can be turned off to save energy?
6. Will facility X be used in the near future?
7. Which manually defined facility/process schedule exists for facility X?
8. Does an automatically generated facility schedule exist for facility X?
9. Which modes of operation exist for an energy consuming facility X?
10. What is the current state of facility X?
11. What is the current energy demand of facility X?
12. What is the energy demand of facility X in operation state Y?
13. What control commands does facility X accept?
14. Which facilities need permanent power supply?

3.1.5 Use Case: TH05 - Assure the Efficient Integration of Local Energy Producers (Energy Producers)

Context of Use

> *The smart home system controls electric consumers in the smart home to use locally produced energy in the best way. The scheduling process includes weather information and required device operations to estimate when enough renewable energy will be produced and schedules energy-intensive processes accordingly.*

Use Case Description

The SHS calculates if energy consumption can be met with locally produced energy. Therefore, apart from energy consumers, the KB needs to represent energy power plants and facilities (e.g., geothermal heat pumps, solar panels) that are locally available and produce renewable energy. For energy producers, the KB has to model the maximum power output that can be achieved but also their current energy output. Additionally, it is necessary to store the possible modes of operation and in that regard if a facility can be turned off. In order to let the SHS calculate a successful energy strategy, the KB further needs to supply current weather measurements and also near future weather predictions. With these weather forecasts the SHS can schedule energy-intensive processes at times when a maximum energy output can be expected from local energy producers. Weather predictions may in this respect come from a local weather station or an Internet weather service. As weather forecasts for predictive control need to be highly reliable, the KB should only consider prognoses up to 24 hours into the future. The SHS subsequently can create an operation schedule for specific devices that adheres to times when most free energy is available by local renewable energy producers. Therefore, it is again necessary that the KB represents energy consuming devices and their energy consumption in different states of operation. If local energy producers do not produce enough renewable energy, the SHS queries the KB for external energy providers. The KB returns external energy providers and their energy tariffs to give the SHS a possibility to choose the best variant to purchase energy. If local power plants produce more energy than can be consumed, e.g., because no energy-intensive processes are imminent, the SHS assesses possibilities to feed back the surplus on energy into the energy grid. In this case, the KB also has to take care of recovery tariffs offered by external energy providers.

Competency Questions

1. Which energy producing facility exist in the building?
2. What type of renewable energy is used as input by energy producing facility X?
3. What is the maximum energy output of energy producing facility X?
4. What is the current energy output of energy producing facility X?
5. Which modes of operation exist for energy producing facility X?

6. What is the current weather situation?
7. What will the weather situation be like in the next X hours?
8. Can sunny weather be expected in the next X hours?
9. Can strong wind be expected in the next X hours?
10. Which possibilities exist to assess the current and future weather situation?
11. Which local weather stations exist?
12. Which Internet weather services exist?
13. Which modes of operation exist for an energy consuming facility X?
14. What is the energy demand of facility X in operation state Y?
15. Which operation schedule exists for energy consuming facility X?
16. Which external energy providers provide energy type X?
17. Which energy tariffs are offered by energy provider X?
18. Which feedback tariffs exist in the case that excess energy is produced?

3.1.6 Use Case: TH06 - Assure the Optimal Integration of Energy Providers

Context of Use

> *The smart home system aims at selecting and using energy providers that are sustainable with respect to their energy production and sell energy at the lowest price. From a range of energy providers, the SHS chooses the provider that has the most ecological and economical way of producing energy according to its energy mix and with respect to offered energy tariffs. Energy from this provider is then purchased in order to satisfy the energy demand of the controlled building.*

Use Case Description

The SHS usually acquires energy from a default provider however may switch the energy provider if a better option to purchase energy is available. In this context, if an energy intensive operation is imminent and local energy providers do not produce enough renewable energy, the SHS needs a list of external energy providers that provide the needed energy type. Therefore, firstly, the KB needs to represent energy providers and the type of energy they provide. In the case of electricity providers, or more general, providers that supply secondary energy, the KB further needs to model the energy mix used to provide electric energy. This way, the SHS can choose the "greenest" energy provider available, hence, the one with the largest percentage of renewable energy used for secondary energy production. For this selection task, it is certainly necessary that the KB differentiates between renewable and non-renewable energy forms.

As a second major influence for the SHS to determine which energy provider is optimal, it is necessary to be able to represent the most economical option in the KB, hence, the provider with the best energy selling conditions. For that reason, it needs to cover energy tariffs and their monetary costs. As different tariffs might exist depending on the time of the day or year (e.g., night/day tariff, summer/winter tariff), the KB additionally needs to describe the time a specific energy tariff may be used, hence, its active time. This gives the SHS the opportunity to consider different tariffs for predictive scheduling of facilities.

Competency Questions

1. What is the default energy provider for energy type X?
2. Which external energy providers provide energy type X?
3. What energy mix is used by electricity provider X?
4. Is electricity provider X a "green" energy provider, hence does it only provide energy from renewable energy forms?
5. Which energy tariffs are offered by energy provider X?
6. What is the monetary cost of energy tariff X?
7. What are the active times of available electric energy tariffs?
8. What are the active months of available electric energy tariffs?

3.1.7 Use Case: TH07 - Predictive Operation of HVAC Services

Context of Use

> *The smart home system realizes predictive heating, cooling and ventilation of the building, in case that comfort conditions can be provided without requiring notable amounts of energy. For this purpose, environmental conditions are exploited, for example, sunlight may be used to heat a room and cool night air to lower temperature in summer. The system uses weather information (favorable conditions, such as specific weather situations) to calculate an HVAC schedule. It favors the use of local energy or free cooling and free heating to keep or establish comfort values in the home. In order to use "free" HVAC mechanisms, the system may temporarily overrule comfort parameters within predefined borders (time and range).*

Use Case Description

The SHS requests different room schedules (e.g., occupancy, device usage, preference schedules) to assess if predictive actions may be taken. The KB has to provide automatically created (i.e., predicted) or manually defined schedules to the SHS. If the SHS needs the characteristics of a particular room for its calculations, the KB has to provide these additional parameters (e.g.,

adjacent rooms, wall thickness, space, volume). The SHS then queries for all facilities from the HVAC domain that can be used to change a particular environmental parameter. The KB therefore needs to offer different possibilities on how to change environmental conditions in the room. If the SHS chooses a process that depends on the weather situation, it queries the KB for the weather situation. The KB needs to be able to represent current weather conditions and short-term weather forecasting. This way, the SHS can consider the current situation to work in favor of achieving a particular goal. For example, if the SHS chooses to use solar radiation (i.e., insolation) as positive influence factor on room heating preparation time the KB needs to be able to identify sunny weather. With such a detected weather situation, the SHS may await sunrise and as the sun reaches certain parts of the building throughout the day open blinds and shutters to use solar radiation for natural heat gain. If the SHS considers natural humidification of a room, the KB has to provide a weather forecast that predicts moist weather (i.e., fog, rain). The need for humidification especially emerges when, for example during the winter season, the air moisture in living spaces sinks below the comfort range as a result of space heating. In the summer season again, the SHS can schedule a natural cooling process (i.e., night purge) that reduces the need for energy-intensive artificial cooling of the building. Therefore, the KB needs to identify a cool night weather situation.

Competency Questions

1. Which manually defined facility schedules exist for room X?
2. Which automatically defined facility schedules exist for room X and day Y?
3. What is the predicted value for all different parameters in room X in the next few hours?
4. The environmental conditions of which rooms will need to be changed in the near future (e.g., according to the occupancy schedule)?
5. What is the predicted occupancy schedule for room X in the next few hours?
6. What are the building values for room X?
7. Which processes can influence parameter X in room Y?
8. Which HVAC processes can be started in room X?
9. Which facilities are involved with process X?
10. Is a process dependent on the weather situation?
11. What is the weather situation at the moment and in the near future?
12. Will there be sun for passively heating the building?
13. Does the weather situation allow passive humidification (i.e., is “moist” weather expected)?

14. Can cool weather for passively cooling the building be expected in the next X hours?
15. Which modes of operation exist for a scheduled facility?

3.2 Ontology Dimensions of Information - Universe of Discourse

When a smart building is to be controlled autonomously by a holistic system, there are different domains involved that influence its behavior and therefore need to be covered. A wide variety of parameters needs to facilitate the system to apprehend the environment and act according to prevailing conditions. Generally speaking, the more is known about the environment the system is operating on, the better a system can control it, trying to reach the identified global goals such as energy efficiency and human comfort. Considering the main control tasks identified in Chapter 1, i.e., lighting, heating and cooling, ventilating and humidifying, and the plethora of different competency questions derived from the general smart home use cases, seven dimensions can be identified that are of importance to reach an optimized building operation with regard to envisioned applications:

- **Building Information.** Information from architecture and building physics can be seen as a rarely considered influence factor on building control. If the condition of exterior walls as well as the layout of a room and its position in the building is known in advance, a heating strategy for example can be optimized regarding the starting time when preparing comfort conditions. This dimension therefore needs to cover information like the size and thickness of walls, used materials and layout of the building as well as orientation.
- **Resource Information.** In the present application domain, resource information is considered to cover all facilities and appliances that can be found in a home. As a smart and ubiquitous home is usually filled with devices used to sense and change the environment, an unambiguous classification of devices is important to overcome the integration challenge mentioned in Chapter 1. In modern smart homes apart from appliances from the building automation domain, often also entertainment and household devices as well as local power plants like solar panels or a geothermal heat pump need to be integrated into the control system. Therefore, this dimension has to provide information about the characteristics of facilities to be found in the supervised home, where these are located, how they function and how they can be accessed. All this information can subsequently be utilized by the agents in combination with a concept of the control capabilities (i.e., processes) that exist in a specific room or section of the building (i.e., zone) to autonomously operate the environment on behalf of the user. The dimension further needs provide a classification of a wide variety of different facilities to the system, making it easier to integrate new, yet unknown devices into the smart home.
- **Energy Information.** Further, when it comes to the goal of increased energy efficiency in buildings, it is important to cover energy-related parameters. The energy dimension on the one hand needs to include a representation of energy parameters for facilities and appliances in the building. Therefore, a close connection between the resource information and the energy information dimension can be identified. On the other hand, in order

to efficiently operate a smart home in a smart grid context, it is further necessary to describe energy related concepts that concern energy supply. Therefore, this dimension also needs to cover energy tariffs as well as different energy providers in order to obtain an assessment of energy that also extends outside the building itself.

- **Building Processes Information.** This information allows a smart system to reason about which facilities would be appropriate for changing the situation in the building in order to reach a certain desired state. Therefore, this dimension has to include processes that may be started by the system autonomously as well as their effects on the environmental state.

- **User Behavior Information.** Another dimension needs to cover user behavior and habits. This information is necessary so that the system can autonomously detect recurring patterns and as such can act according to users, their behavior and associated usual usage of facilities and occupation of building parts. As a consequence, regular behavior of users needs to be represented. In this regard, also the future building context can to some extent be retrieved by an agent in order to adjust control strategies accordingly.

- **User Preferences Information.** Demographic information of inhabitants is another domain that may influence the operation of a future smart home system, for example, to optimize settings with regard to age and gender. Further, in this dimension it is important to represent user preferences, i.e., settings that are necessary to create a global notion of user-based comfort. Parameters such as preferred room temperature, noise level or lighting but also facility-centered preferences are among the necessary terms to be modeled. To allow an adequate control of the building even when no manually stored preferences are available, also the representation of standard preferences following widely accepted comfort standards and recommendations is considered at this point.

- **Exterior Influences Information.** The final dimension that needs to be represented for an advanced building control involves weather and climate influences. In this case, it is necessary to cover the most important weather and climate parameters for a smart home control such as wind, rain or temperature and provide definitions of different weather conditions interesting for predictive scheduling of system tasks. In this regard, the represented information may be used to optimize building control and for example exploit exterior conditions in favor of an energy-saving building behavior.

Considering the previously defined competency questions, a classification of questions of different granularities on different levels is possible. According to [104] an ontology can only be well-defined if also more general queries are taken into account. Therefore, it is not only relevant to define questions representing direct queries to the ontology but it is equally necessary to identify higher level questions describing broader goals. For the previously introduced competency questions, roughly two levels are identified. While *low-level* questions are answered directly by the knowledge base through a suitable query mechanism, *high-level* questions represent more general queries that can be answered by combining low-level questions.
For low-level queries a mapping for each question to one of the introduced dimensions is shown in Table 3.1. In the case of high-level questions, it makes sense to exclude them from the domain

Table 3.1: Ontology dimensions and associated competency questions. High level competency questions are collected in a separate class. Single questions are referred to by their respective use case and number (e.g., TH01.01 for the first competency question of use case TH01). Duplicate questions appearing in more than one use case have been eliminated.

Ontology Dimension	**Associated Competency Questions**
Building Information	TH02.06, TH02.07, TH02.08, TH02.09, TH02.10, TH02.11, TH02.12, TH02.13, TH02.14, TH02.24, TH02.27, TH02.28
Resource Information	TH01.01, TH01.03, TH02.16, TH02.25, TH02.26, TH02.29, TH03.03 TH03.07, TH03.08, TH03.09, TH03.10, TH04.13, TH04.10, TH05.01
Energy Information	TH03.06, TH04.04, TH04.11, TH04.14, TH05.02, TH05.03, TH05.04, TH05.14, TH05.16, TH05.18, TH06.01, TH06.03, TH06.04, TH06.05, TH06.06, TH06.07, TH06.08
Building Processes Information	TH01.09, TH01.10, TH01.11, TH02.17, TH02.18, TH02.19, TH03.04, TH07.07, TH07.08
User Behavior Information	TH03.12, TH03.13, TH04.02, TH07.02, TH07.03, TH07.05
User Preferences Information	TH01.06, TH01.08, TH02.01, TH02.02, TH02.03, TH02.04, TH02.30, TH03.02, TH03.01, TH04.03, TH07.01
Exterior Influences Information	TH01.13, TH02.23, TH05.06, TH05.07, TH05.08, TH05.09, TH05.11, TH05.12, TH07.13, TH07.14
High-Level	TH01.07, TH02.05, TH02.15, TH02.20, TH02.22, TH03.14, TH05.15, TH04.05, TH05.10, TH07.06

classification as with these inquiries not necessarily a single dimension is involved. Therefore, these questions are aggregated in the "high-level" class in Table 3.1. The relationship between high-level and low-level questions is analyzed in detail in Chapter 5.

The seven dimensions presented in this section reflect main influence factors on building performance. Without doubt it is possible to put all of this information into one single ontology. However, this would result in a very specialized schema tailored specifically to the presented smart home system, making it difficult to reuse the ontology for any purposes other than smart home control. Further, largely unrelated topics and dimensions would be represented in one single ontology, leading to a mixture of domains and associated terms. A system reusing such

an all-in-one ontology would have to include concepts from all domains, regardless if only one particular domain is needed. Another disadvantage that can be identified is reasoning time: in a single comprehensive ontology that includes a large number of axioms describing all the aforementioned dimensions and their concepts reasoning would be extremely costly taking a substantial amount of time. Therefore a modularized approach reflecting the identified dimensions is followed and thoroughly explained in the following chapter.

CHAPTER 4

Smart Home Ontology Creation

As already briefly discussed in Chapter 1, one of the main benefits of a multi-agent approach in the smart home domain is the inherent flexibility of such a system. The tasks in a smart building are usually performed by devices distributed throughout the environment and these tasks not always follow recurring routines. At the same time, unexpected situations may arise at any moment. A multi-agent-controlled smart home offers considerable advantages compared to other, more traditional system approaches as it provides a goal-oriented behavior of loosely coupled software entities [201]. An autonomous system following this method clearly benefits from a global view of the world (i.e., knowledge base) as in this case it can act on explicitly represented knowledge and flexibly adjust its behavior accordingly [38]. In this context, a shared vocabulary in form of an ontology can be seen as desirable to overcome semantic heterogeneity and describe concepts in the world of interest unambiguously and explicitly [158, 44]. Ontological descriptions are vital for a variety of general tasks as classified in [105]:

1. *For communication*
   - *between implemented computational systems.*
   - *between humans.*
   - *between humans and implemented computational systems.*
2. *For computational inference*
   - *for internally representing and manipulating plans and planning information.*
   - *for analyzing the internal structures, algorithms, inputs and outputs of implemented systems in theoretical and conceptual terms.*
3. *For reuse (and organization) of knowledge*
   - *for structuring or organizing libraries or repositories of plans and planning and domain information.*

With respect to a multi-agent-controlled smart home, these general tasks turn out to be very suitable. Firstly, a multi-agent system can be identified as computer system that is composed of several autonomous parts in need to communicate between each other and having a consensus on what the controlled environment is about: an ontology as a semantically rich conceptualization of the domain can hereby serve as common grounding for agents in the system to base their beliefs on (1.). Further, with inherent reasoning and complex modeling capabilities an ontology allows a more detailed description of the building environment than it is possible with classical data representation methods (i.e., relational databases). The reasoning mechanism can for example be utilized for query answering by realizing common queries in the ontology taking away complexity from the building operation system (2.). Lastly, a classification in ontologies supports the simple integration of new sources of information as they arise. Ontologies are designed for open systems and may be used to integrate multiple information sources into one system, in this way organizing knowledge unambiguously (3.). This generality of information representation cannot be easily achieved in classical storage systems, which are most often tailored to a specific application or system. To model dependencies in the world of interest expressively, the representation as ontology is proposed. This presentation can subsequently act as common knowledge or shared vocabulary for a multi-agent based smart home system.

4.1 Ontology Modularization, Classification and Normalization

To make the modeled knowledge reusable outside the smart home domain and at the same time minimize adaptation efforts for usage in other applications, a split into independent ontology modules is considered for the created ontologies. In this context, the concept of **ontology modularization** may be used to make application-level ontologies reusable for a wide variety of applications in a specific domain. Often, when ontologies are developed for a particular application, they become too specialized and unusable for other purposes, which means that conceptualizations are many times only useful in the specific application they have been designed for. On the other hand, there exist ontologies that are created with the aim to describe a part of the world generally in a way that a great variety of applications can rely on this conceptualization. Ontologies of this kind necessarily cannot be as detailed as the first mentioned type with respect to concept descriptions and axiomatization, as this would make them too inflexible for the envisioned field of application, i.e., describing dependencies between concepts globally. From that viewpoint, ontologies are often either created for the specific use in a particular application or especially developed for sharing information [216].
In this respect, [107] introduces an **ontology classification** and identifies four kinds of ontologies, which are upper and domain ontologies as well as task and application ontologies. *Upper ontologies* or top-level ontologies describe general relationships needed by a variety of different user groups independent of application or domain. In other words, already defined upper ontologies that categorize general relationships and terms valid over all domains (e.g., time, location) should be reused in order to minimize ontology implementation effort and put definitions in the newly created ontology in relation to other already existing ontologies also based on that particular upper ontology. As such, an upper-level ontology can be used to reach semantic interoperability and act as basis for domain and task ontologies [123]. Further, *domain ontologies*

and *task ontologies* specialize the terms of the upper-level ontology in order to describe a particular domain (e.g., weather) or particular tasks (e.g., context retrieval). Domain or task ontologies usually describe a selection of concepts introduced in the upper ontology in detail (i.e., domain knowledge) while omitting the description of unrelated terms. *Application ontologies* may again be seen as specialization of domain and task ontologies and according to [107] depend on both ontology types. As such, this type in further consequence often includes concepts of a domain that are only useful for a specific task and describes dependencies many times solely useful for one particular application. In the reviewed literature, there is however often no clear distinction between domain and application ontologies and the terms are sometimes used interchangeably.
With regard to this classification, firstly, two generic ontologies describing general terms are reused to minimize modeling effort and build on already defined and accepted representations of standard terms. No single, comprehensive global upper ontology is used due to reasons explained in the following section. The developed smart home ontology may further be seen as application-level ontology. To overcome over-specialization and increase the shareability and reusability of the conceptualization however, a twofold goal is identified: while firstly the ontology is obviously tailored to the needs of a multi-agent-controlled smart home system, a division into self-standing modules is considered to make these modules useful for a wider spectrum of applications. As such, the created ontology modules may be categorized as a particular form of domain ontologies, as they are split according to different domains that may be reused in applications different from the smart home use case. A strict classification is however not possible, as the domain modules still include concepts that are defined with the clear focus on an application in a smart home system. This fact does however not lessen the reusability of the domain conceptualizations. The modules are then interconnected through a set of defined properties to a comprehensive smart home ontology (cf. Section 4.1.2). While this extensive specification of influences for smart home operation does not define additional concepts and as such is no specialization of any of the domain modules as postulated in [107], its specialized orientation and the domain-crossing nature clearly classifies it as application ontology [161].
When aiming at the design of reusable and modularized ontologies common Ontology Design Patterns (ODP) as defined in [88] and reviewed in [72] and [207] cannot be neglected. In this context, especially the topic of **ontology normalization** needs to be highlighted. Concerning this topic, in [198] a two-step normalization procedure is introduced while emphasis is laid on the importance of normalized ontologies considering modularization and maintainability. Firstly, the article differentiates between primitive and defined concepts as well as properties, restrictions and axioms. *Primitive concepts* are classes that are solely described by "necessary conditions", hence are described using only subclass relationships to other concepts. The description of *defined concepts* additionally includes "sufficient conditions" that establish an equivalence relation to a restriction or axiom. As such, the reasoning mechanism can automatically infer hierarchies for defined concepts while primitive concepts need to be arranged in a hierarchy manually. *Properties* relate concepts and *restrictions* as well as *axioms* may be used to describe them. The article and related publications covering the same topic (e.g., [71], [207]) identify several normalization guidelines which should be followed when modeling domain ontologies:

- **Homogeneous tree-shaped hierarchies**. One of the main prerequisites for a normalized ontology is to manually only define tree-shaped hierarchies for primitive concepts. A "primitive skeleton" consisting of only primitive concepts should additionally form trees of homogeneous and logically connected terms. Multiple inheritance hierarchies (i.e., "polyhierarchies") should not be asserted by hand but merely inferred by the reasoning mechanism. Following this guideline at creation time of an ontology prevents tangling term definitions (i.e., asserted polyhierarchies) and makes the designed conceptualization much easier to maintain, as for each concept only the main inheritance hierarchy relationship has to be taken care for. All other subclass relations can be inferred through the explicit axiomatization of the concept in form of restrictions and axioms describing its relationship with other concepts in the modeled domain. This characteristic also facilitates an easy extension of the ontology.

- **Disjoint primitive concepts**. Generally, in normalized ontologies, primitive concepts on the same level in a tree should be disjoint to each other and open with respect to child concepts[1]. On the other hand, for value partitions as a special type of primitive concepts the disjointness does not have to be enforced and its child entities need to cover the value partition, hence form a disjoint and exhaustive partition [198] (cf. Section 4.6).

- **Explicit characterization**. It is one of the main goals of modeling an ontology to achieve explicitness, hence to describe the modeled concepts in a way so that additionally to the human reader also a machine process (i.e., reasoner) can make formal inferences according to their general description. As such, it is essential to eliminate implicit meaning that is hidden in labels and names and for that reason only available to a human observer (i.e., tacit knowledge). In that case, an ontology needs to make information explicit to create machine interpretable concepts. Necessarily, to gain interpretability, the concepts have to be described with a number of axioms and restrictions that clearly express interrelations with other modeled constructs.

- **Consistent naming**. While not explicitly stated in the mentioned articles, a consistent naming policy should be applied. This point does not influence the formal reasoning in the ontology, however, it makes the ontology comprehensible for human observers and prevents modeling and alignment mistakes introduced by human error.

All of these requirements are considered in the design phase of the smart home ontology modules to increase their reusability and maintainability. Additionally to the outlined characteristics, [198] proposes different "axes" for manually asserted primitive hierarchies on the one hand and reasoner inferred concepts on the other hand. This normalization step is omitted for practicability reasons, and instead only the aforementioned requirements are considered. While this semi-normalized form still ensures reusability and maintainability to a sufficient extent it at the same time meets the requirements of an application-level ontology. As a further step, additional ODPs are reused where applicable (cf. Section 4.6). In case an ontology is reused that is not normalized, a normalization step is applied beforehand (cf. Section 4.3).

[1] Hence, the list of child concepts is not exhaustive.

4.1.1 Ontology Reuse: Foundational and Generic Ontologies

To prevent that the same dependencies, relationships and viewpoints are implemented in newly developed ontologies over and over again, *ontology reuse* is generally considered good practice. To put ontologies in a greater context and make them more shareable sometimes the inclusion and specialization of an upper ontology is considered. These top-level ontologies are mainly used to integrate different domain-level ontologies and put their definitions in relation to each other. In this context, the upper-level ontology ODP [72] describes how to integrate ontologies in one foundational ontology, with each domain ontology having different relations to concepts defined in the upper ontology [73]. Some of the most relevant foundation ontologies are the Suggested Upper Merged Ontology (SUMO), CYC and DOLCE+DnS Ultralite. The SUMO Ontology [176] is one of the candidates for the IEEE P1600.1 Standard Upper Ontology [227]. As such SUMO generally describes a wide variety of terms and concepts and currently holds around 25.000 terms and 80.000 axioms [188]. OpenCYC [156] is another large top-level ontology. Its core ontology currently includes about 239.000 terms and 2.093.000 triples describing *"all of human consensus reality"* [57]. The CYC ontology is also a candidate for the Standard Upper Ontology. One ontology that does not intend to become the IEEE P1600.1 Standard Upper Ontology is the Descriptive Ontology for Linguistic and Cognitive Engineering (DOLCE). Originally described in [91], it has a cognitive bias and as such focuses on capturing natural language and human common sense. The Descriptions and Situations (DnS) ontology also defined by the author of DOLCE can be used to extend this ontology and represents situations, contexts, theories and norms [92]. There exist a lightweight version that is entitled Dolce+DnS Ultralite [89] which in a simplified way can be seen as a reduced conglomerate of the two previously introduced ontologies. This conceptualization decreases the amount of described concepts to define a lightweight foundational ontology that is capable of modeling physical or social contexts [89].
An inclusion of such a high-level definition of terms in the created smart home ontology would definitely have benefits. For example, the different ontology modules defined for the smart home operation use case describe different types of temperature. An outside air temperature may be seen as a different temperature than an indoor air temperature, a temperature measured in a heating boiler or a preference temperature of a specific user. Implementing the upper-level ontology ODP, all these temperature types could be derived from one single concept "temperature" defined in the foundational ontology making it clear that all of these concepts are subclasses of the same type. While this illustrates an advanced classification of concepts that definitely simplifies integration of ontologies, the use and implantation of upper level ontologies in domain/application-level ontologies are generally controversial [72]. Firstly, because of the commitment to the upper-level ontology, the domain ontology needs to follow the concrete view as it is described in this particular conceptualization. Although upper ontologies are usually kept very general with respect to the axiomatization and definition of terms, this commitment naturally influences the way the ontology has to be modeled as the definition of relevant terms in the upper-level ontology needs to be followed. Also philosophically, the usability of upper level ontologies is questionable as *"the initial project of building one single ontology, even one single top-level ontology, which would be at the same time non-trivial and also readily adopted by a broad population of different information systems communities, has largely been abandoned."*

[221]. Further, considering implementation, the upper-level ontology in its entirety has to be included, which with large generic ontologies like the ones described before tends to become problematic, particularly in the case when only a few classes need to be reused.
Because of their extent and very general nature which opposes the relatively specialized focus of the smart home ontology modules, the inclusion of an upper ontology is therefore not deemed to be rational in the present case: the advantage of aligning the ontologies with one of these general top-level ontologies is compensated with an extremely high overhead on additional triples and concepts. Reviewing the ontologies introduced before, they cannot be seen as candidates for a fitting foundational ontology: their main purpose is to describe general terms in a way so that different ontologies can be put into relation to each other. While this is an essential prerequisite for ontologies used in the classical Semantic Web integration use case, in the present matter, an alignment to such a general purpose top-level ontology would rather be disadvantageous. As the ontologies are defined in a very general way, they include a great amount of concepts not immediately related to the smart home domain. According to [216], there is always a tradeoff to be made between ontologies for use and ontologies for sharing. In this respect, considering the import of an upper-level ontology would increase the size of the smart home ontologies for only little benefit regarding the defined use cases (cf. Chapter 3). This additional complexity would significantly slow down the reasoning process and query responses. Further, while the alignment to such an ontology is useful when performing ontology matching or comparing conceptualizations of different ontologies on the Internet, it is relatively irrelevant for an ontology acting as knowledge base for a multi-agent based smart home system. A mapping could be considered, however is omitted to keep the created ontologies concise and reasoning times low.
On the other hand, there exist generic ontologies which are describing a smaller aspect of reality in a modular manner and therefore are more suitable for inclusion in the present field of application. For two associated domains, such generic ontologies are reused in the created conceptualizations:

- **Location Information**. For location information with respect to weather forecasts (cf. Section 4.6), the Basic Geo (WGS84 lat/long) Vocabulary [41] is reused. This vocabulary, which was developed by the W3C Semantic Web Interest Group is a very simplistic conceptualization that represents global positioning parameters such as latitude and longitude for spatially-located things [41]. The main elements of this vocabulary are the concept `SpatialThing` and its attributes `lat`, `long`, and `alt` following definitions in the WGS-84 reference specification [64]. Unfortunately, the OWL version of the vocabulary identifies the properties `alt`, `lat` and `long` as annotation properties. This cannot be seen optimal as annotation properties are basically ignored by the OWL DL reasoning mechanism. Therefore, before reuse, the vocabulary is slightly remodeled and the properties are defined as OWL datatype properties instead. While the Basic Geo (WGS84 lat/long) Vocabulary is not a W3C standard, according to the project website the vocabulary is widely used for RDF and non-RDF documents.

- **Temporal Information**. For temporal descriptions in the smart home ontology the OWL-Time ontology originally proposed in [186] is used. This ontology which by this time is considered a W3C draft (cf. [254]) allows to generally specify time instants and intervals

and facilitates the ordering of individual time events. For that purpose, the OWL-Time ontology provides a variety of concepts and relationships and offers the possibilities for a very detailed description of temporal information. The central concept of the vocabulary is entitled `TemporalEntity` which can be specialized into one of its subconcepts `Instant` and `Interval`. The vocabulary further defines a set of datatype and object properties that can be used to describe these temporal entities and their relationships in detail. In the smart home ontology, the modules use the standard import mechanism of OWL to include the OWL-Time ontology as it is and as such obtain a standard notion of time.

A detailed account on how the conceptualizations are reused in particular modules of the smart home ontology can be found in the respective sections (cf. Section 4.3, 4.6).

4.1.2 Ontology Modules Overview

With the above introduction and motivation it should be clear at this point that modularization is one of the key requirements for reusable ontologies. Figure 4.1 shows an overview of the different ontology modules that are identified in the smart home use case. It reflects the previously identified dimensions of information (cf. Section 3.2) however consolidates some dimensions into single ontologies. When capturing the main concepts of these different dimensions, it has to be considered that some of them are further apart and as such contain more diverging data, while others are more interrelated. This means, while weather information may be realized as stand-alone exterior ontology, other representations such as the resource and energy dimensions, have strong interdependencies. From the viewpoint of a smart home control system, it is therefore useful to model certain interrelated dimensions as single ontology. For this reason, a consolidation is executed identifying domains encompassing defined concepts of more than one dimension and a representation as independent ontologies is carried out (cf. Figure 4.1). As a benefit, a more concise representation of reality can be achieved, while a basic modularization into separate domains is still provided, guaranteeing the reusability of single descriptions for other domain-related applications.
Regarding information about building structure, simulation studies like [144, 238, 31] show that the quality of the building envelope has a significant impact on the building energy performance. Especially in old stock buildings the architecture and building envelope is generally suboptimal for energy-optimized operation. Independently of the installation of a smart home system either within a refurbished old stock or a new stock building, static knowledge about building construction, orientation and architecture can always be seen as an additional performance parameter that can be included when conducting optimizations at operation time. The ***Architecture & Building Physics*** ontology (prefix: `gbo`) therefore contains information about the building itself, which may for example be retrieved from building design and performance software.
As already discussed in Chapter 1, in a typical smart home there are a lot of energy consuming appliances. Obviously, for a smart home operating in a future smart grid a conception of information about energy facility consumption and energy supply and tariffs is needed. This additional information enables the scheduling of facilities according to energy spot prices and may give feedback to occupants again leading to energy demand optimizations [240, 58]. With heat-

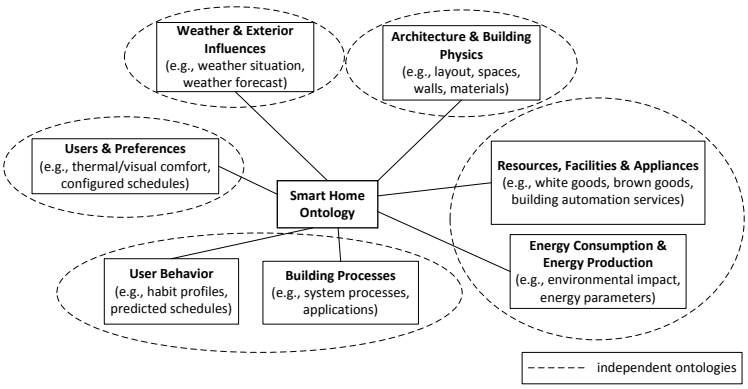

Figure 4.1: Parameter dimensions for smart home control

ing and cooling making up major parts of the general energy consumption of a residential home, significant optimization potential can be found in this domain. While these two dimensions may be represented in separate ontology modules, in the present case one single ontology module is created for both dimensions. The ***Energy & Resources*** ontology (prefix: `ero`) therefore represents facilities, energy providers and associated parameters that influence building performance and are needed to optimize the general energy behavior of the building.
In order to describe context in a smart home, the system further needs to represent building processes and user habits. Processes that can be initiated to change the environment and their energy impact may be used for a system to decide at a certain moment in time, which action is fitting to reach a particular global goal such as energy efficiency. Another part of the building context concerns recurring actions of human actors in the home. Usually the behavior of human actors follows routines [113] that have to be modeled in a comprehensive smart home knowledge base as well to be able to adapt the environment taking into account to expected future situations. Again, the two dimensions of user behavior and building context are consolidated into one single ontology module: the ***User Behavior & Building Processes*** ontology (prefix: `ppo`) is designed to picture the context of the supervised home and its inhabitants. With regard to the classification introduced in [181], this ontology module can also be classified as task ontology.
Generally, also occupants as well as their personalized comfort preferences have to be represented as well. Therefore, for an accurate model of users including demographic data and their preferences in an own module, a ***Users & Preferences*** ontology (prefix: `apo`) is considered.
Another factor that has to be taken in to account is the domain of external influences. Weather and climate conditions are considered to be substantial when approaching an energy efficient building operation [183]. For that reason, also weather observations and states are conceptualized and integrated into the designed smart home knowledge base as own ***Weather & Exterior Influences*** ontology module (prefix: `weo`).

Interrelationships between Ontology Modules

According to [66] an ontology module can be described as *"a reusable component of a larger or more complex ontology which is self-contained but bears a definite relationship to other ontology*

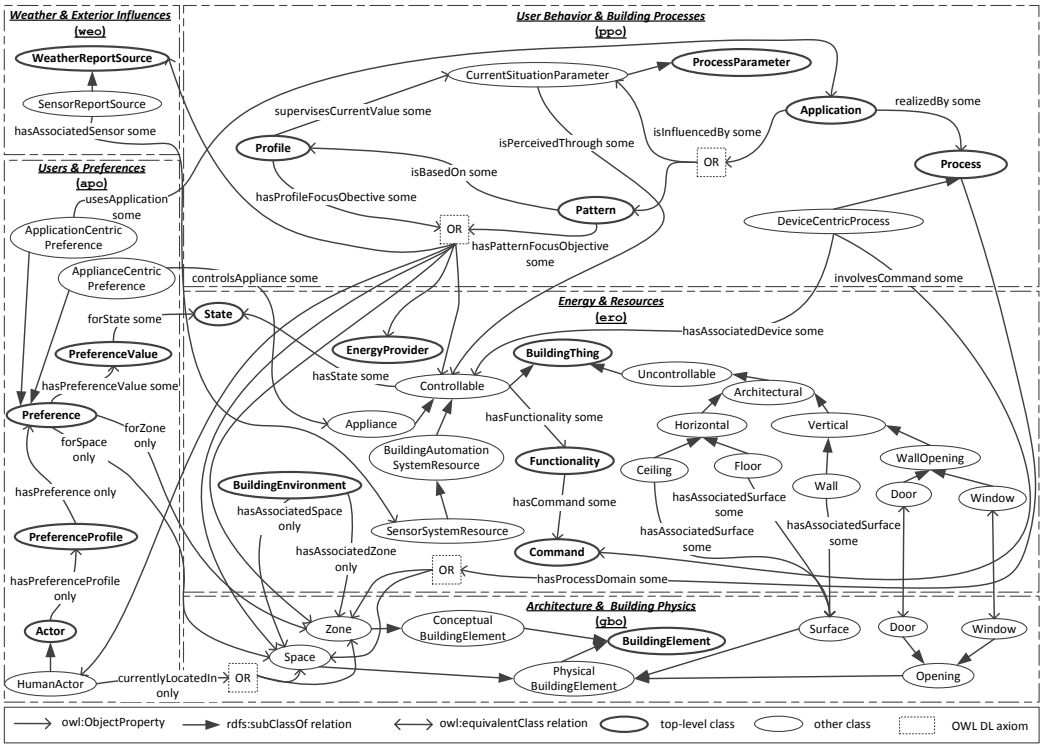

Figure 4.2: Ontology modules and their relating properties

modules". Naturally, the introduced domain modules need to be connected in order to form a comprehensive application ontology for smart homes. For this reason, several interrelating properties are defined: these are realized as `owl:ObjectProperty` relations and may be used to connect the identified five individual modules as shown in Table 4.1. This way, an instantiation of one ontology can be flexibly connected to an individual of another ontology module using globally unique URIs and the defined properties. An example is the relationship between the Energy & Resources module and the Architecture & Building Physics module: While in the first ontology a general notion of building environment may be represented in order to relate facilities to building parts, through the property `hasAssociatedSpace` it becomes possible to associate the building part to a space defined in the second ontology holding more information about dimensions and enclosing construction of a building part. Further, with the property `hasAssociatedZone` a specific room can be related to another partitioning scheme defined in the second ontology dividing the building into zones. Additionally, there are two `owl:equivalentClass` relations defined between the Energy & Resource and the Architecture & Building Physics module. These relations emerge as the two concepts `Door` and `Window` are defined in both ontology modules.
The properties defined for interconnecting ontologies and `rdfs:domain` and `rdfs:range` classes of the related modules are shown in Figure 4.2. The purpose of the properties becomes clear in connection with the detailed information about the single modules in the dedicated

sections. For most of the properties their usage can further be observed in the discussion and evaluation chapter (cf. Chapter 5).

Table 4.1: Interrelating properties between ontology modules

Property Name	**Property Domain** (Ontology)	**Property Range** (Ontology)
`hasAssociatedSensor`	Weather & Exterior Influences	Energy & Resources
`hasProfileFocusObjective`	User Behavior & Building Processes	Architecture & Building Physics, Energy & Resources, Users & Preferences, Weather & Exterior Influences
`hasPatternFocusObjective`	User Behavior & Building Processes	Architecture & Building Physics, Energy & Resources, Users & Preferences, Weather & Exterior Influences
`isPerceivedThrough`	User Behavior & Building Processes	Energy & Resources
`hasAssociatedDevice`	User Behavior & Building Processes	Energy & Resources
`involvesCommand`	User Behavior & Building Processes	Energy & Resources
`hasProcessDomain`	User Behavior & Building Processes	Architecture & Building Physics
`hasAssociatedZone`	Energy & Resources	Architecture & Building Physics
`hasAssociatedSpace`	Energy & Resources	Architecture & Building Physics
`hasAssociatedSurface`	Energy & Resources	Architecture & Building Physics
`usesApplication`	Users & Preferences	User Behavior & Building Processes
`controlsAppliance`	Users & Preferences	Energy & Resources
`forState`	Users & Preferences	Energy & Resources
`forSpace`	Users & Preferences	Architecture & Building Physics
`forZone`	Users & Preferences	Architecture & Building Physics
`currentlyLocatedIn`	Users & Preferences	Architecture & Building Physics

While it would be possible to integrate the ontologies with the standard `owl:import` mechanism, in the present case this mechanism is not used in order to keep reasoning times fast. The benefit of additional inferences that can be drawn by the reasoning mechanism when integrating all of the domains into one ontology is quite low compared to the performance gain when keeping them separate. Nevertheless, if needed, it is always possible to use the `owl:import` statement.

4.2 Architecture & Building Physics

When aiming at the optimization of a building's energy performance, first of all structure and architecture need to be considered. Although the European *Passive House Standard* limits the annual heating demand of a passive house to $15kWh/m^2$, not all newly constructed residential homes satisfy this standard: the European Union merely aims at constructing only new buildings with nearly zero energy demand by 2020 in response to reaching its 20% greenhouse gas reduction goal [79]. Until then, a significant amount of new buildings is still being constructed without too much concern for energy efficiency in design and construction. The assessment of architectural and building physics impacts on the energy efficiency of a building is however especially important at design time as great energy savings can be reached through the usage of a low-energy architecture.
In conventional building design and construction process all of the stages are rather separate, thus often houses that are designed by architects are not in accordance with the visions of energy engineers and thus are not optimized for energy efficiency. However, architecture, orientation and the usage of the building itself to accomplish energy-intensive tasks such as heating and cooling can highly influence its energy demand and need to be considered when aiming at designing environmentally-friendly buildings [51]. Also in [115] key decisions that influence heating and cooling loads are summarized, while the most important are building form, orientation, self-shading, window-to-wall area ratios, insulation levels and window properties. These qualities are mostly fixed by the architect already at design time, seldom visualizing the final energy performance of the building. A thorough energy analysis at the first stages of a building life cycle and before building construction would be vital for the general energy efficiency of a building, however has rarely been conducted with the old stock, and is still not common practice when constructing new buildings. Therefore, usually, an optimized building structure and architecture cannot be assumed. Information about structural properties of such a non-optimal building is crucial for a control system striving to optimize energy consumption. For example, heating setpoint control (i.e., adjusting room temperature and most energy efficiently bring it to a comfortable level according to the user schedule) is highly dependent on building information. In current residential homes often just very simple and inflexible schemes exist that can be programmed by users in advance by stating the planned occupancy times. However, how fast the desired temperature setpoint is reached, hence the *preparation time*, is dependent on several prerequisites: while obviously the current indoor air temperature of a space directly influences the preparation time, also building architecture and associated building physics may be considered. Building material, state of walls (cooled or heated), as well as general layout of the space influence energy-intensive tasks like heating and cooling, should be integrated into smart home

control in order to optimize building performance. To operate heating and cooling in accordance to the state of the building and its structure at the same time promises an improved thermal comfort and optimized overall energy performance. The building information gathered from Building Information Modeling (BIM) can therefore be treated as immediate influence factor on global control strategies and algorithms focusing on building energy efficiency. The following deliberations describe a building model for smart home systems and are partially taken from the articles [148] and [147], while at this point more detail on acquisition and conceptualization of building properties is provided.

4.2.1 Related Work

In projects defining OWL ontologies for smart homes and buildings, the building itself if at all is only rudimentarily modeled. The DogOnt project [34] while focusing on the detailed modeling of the building device landscape allows only the definition of architectural entities like rooms, walls and ceilings mainly used to associate a device in the building with a specific room. Otherwise, building physics and exact architectural dimensions are not mapped. Other smart home control projects like [266] focus more on residents, their context, behaviors and interaction with available devices. Again, information about the building is reduced to a simple room entity that may include facilities and devices. The BOnSAI project [229] defines an ontology that models functionality, hardware, users and context. The defined ontology also fails to describe the architectural and building characteristics beyond the concepts of locations, rooms and floors.
There further exist publications that generally acknowledge the importance of new developments in the Semantic Web sector when it comes to interoperability and building information models like, for example, [9]. Yet other articles specifically focus on transformation of Building Information Modeling into Semantic Web ontologies for exchange of information between different stages of the building life cycle. The authors of [19] describe the process of transforming the EXPRESS schema that underlies Industry Foundation Classes (IFC) into an OWL representation. In their article they show how the underlying model of IFC may be semantically represented with OWL, as well as arising limitations with respect to automatic conversion. In [269], the transfer of building information from a native IFC model to the OWL language is described and the resulting model is in further consequence used to check the conformity of construction projects against technical norms. Bhatt et al. in [30] propose an ontology-based method to assist the design process of smart buildings. In this respect, the authors focus on the representation of space and introduce different perspectives for which they present several ontology modules. To describe the structural aspects of a building, parts of the IFC model are reused and modeled as ontology. Through the different modules and appropriate reasoning, the validation of structural as well as functional requirements of a smart environment becomes possible. The article [37] proposes a so-called Product Modelling Ontology modeled as a generic reusable upper ontology. The authors of [245] introduce an ontology-based model exchange between stakeholders of the building construction process. The article focuses on the improvement of interoperability between Building Information Modeling tools by developing consistent model views using the Web Ontology Language. Therefore, the authors propose a layered approach basing their specific model on already defined standard and domain ontologies. One of the few articles mentioning the utilization of Building Information Modeling in the operational phase of a smart

building is [215], introducing the application of BIM in a smart indoor environment for disabled people. The authors propose the extraction and transformation of building elements and their interconnections from an IFC model into an ontology that subsequently makes inferences on the extracted information possible. Some example inferences are presented, however no further details on the created ontology are given. While these approaches show the feasibility and necessity of the translation from a BIM to the Semantic Web, neither of these translation processes seems suitable to contribute to the envisioned building information ontology. The reason for this inadequacy is that proposed translations are either too general or specialized in a different direction (e.g., conformity checking, interoperability) while lacking concepts important for building energy analysis. This finally led to the decision to develop a new building information ontology from scratch.

4.2.2 Building Information Modeling

In the building planning and construction process there usually exists a variety of stakeholders throughout the different stages ranging from architects to construction or energy engineers and finally the facility manager or building owner. Information between these parties is classically shared with the help of physical or digital documents. Hereby, details about the building project are divided among different documents and to achieve a global view a manual consolidation of information becomes necessary. In order to facilitate an integrated process and interoperability between the various stages, the description of the building with the help of *Building Information Modeling* is gaining popularity. A BIM is a vendor-neutral, digital exchange format that models the construction process in a virtual environment and describes all the required information of a building and its components from planning over construction to operation and decommissioning [153]. The BIM includes the 3D geometric model of a building from a computer-aided design (CAD) system, however, it is more than a format for CAD tools. The objective of a BIM is to preserve information available about the building over the whole building life cycle [11]. This means, a BIM is a digital model that can be shared among different stakeholders throughout the building life cycle and as such, lead to a greater interoperability between different involved parties and interest areas. Information represented in the BIM grows with the advancement of the building, hence the model aggregates information about components, processes and other related data throughout building life cycle phases. In [46], a BIM is characterized as intelligent simulation of architecture that has to be digital, spatial, measurable, comprehensive, accessible and durable in order to reach the goal of an integrated process. A BIM thus digitally represents the components of a building as objects and associates them with a graphical expression and data attributes that describe the components' behavior and facilitate analyses [69]. As stated by [127], there is a clear difference between building information modeling and building graphic modeling: building information modeling describes objects that have knowledge about their properties and behavior, while a CAD system renders the components and visualizes them as 3D figures. Consequently, a building representation containing only a 3D model does not yet classify as BIM: while 3D models mainly describe the geometry of a modeled building and are effectively used for visualization, BIMs include additional component attributes such as wall types, spaces, air handling units, geospatial information and circulation zones [54]. Additionally, also more abstract concepts, e.g., schedules, activities or construction costs are often represented

[142]. Generally speaking, in addition to the three dimensional CAD model, a BIM adds further dimensions to depict the evolutionary process of the building through changes in time (4th dimension) and cost (5th dimension) [160]. A BIM thus includes domain-specific information about modeled constructs covering the whole building process. Ideally, using a BIM in architecture and construction may lead to interoperability among different domains and software tools, however, widespread use and acceptance still lies ahead. According to a survey conducted for the BIM market in the UK [173], about 40 % of the participants are aware and currently using BIM and more than 90 % are aware and considering to use BIM in the future. This survey, although only valid for the UK market, shows the general interest in Building Information Modeling. One advantage of using a BIM is that the fine-grained digital representation of a building project may already lead to an improved, energy efficient design as the transfer of building data into energy analysis tools becomes possible [233]. In order to reuse the information available in a BIM for the operation of a smart environment, the transfer of this information into the control system can certainly improve control strategies, as knowledge about the building that would otherwise not be available can be incorporated in calculations.

BIM Standards

At the moment there is no single BIM standard, but several proprietary and non-proprietary interchange formats exist. Among the wide variety of different formats, the Industry Foundation Classes and the Green Building XML Schema can be seen as two of the most important open exchange formats up to date [65]. IFC [45] is a model that was developed by a global industry and research consortium in order to specify an extensible non-proprietary information format. The framework may be used to create a large set of data representations for exchange between software applications used to document, model and simulate building design and construction [69]. Because of the general nature of IFC, the model schema itself is split into several layers representing diverse building information classes based on the ISO-STEP [191] modeling standard for building information exchange. IFC is designed *"to provide a means of passing a complete and accurate building data model from one computer application to another without any loss of information"* [54]. Therefore, IFC defines a large class hierarchy with a lot of classes being concerned with representation of the geometry of the building in a coordinate system to enable a realistic 3D visualization. Although the collection of classes and modules supports the applicability of the model for several domains, it may not be seen as universal solution for interoperability in the building domain [9]. For example, while the great variety of supported constructs makes it possible to model any kind of building element and shape, it undeniably also raises the complexity of this exchange format. For a smart home system, the detail offered by IFC is however not needed as in that case, a simple representation of spaces, building components and their properties is sufficient so that they can be taken into account for environmental control. Therefore, because of the high overhead induced by the model, the IFC cannot be seen as an appropriate choice for direct application in a smart home system.
The second important exchange format in the architecture and construction industry relevant for building representation in a smart home is the Green Building XML (gbXML) Schema [95]. The model focuses on exchange of energy-related data between construction and energy analysis software [65] and is widely accepted and used in several commercial tools (e.g., Autodesk

Revit[2] and Ecotect[3], Graphisoft EcoDesigner[4], Bentley AECOsim Building Designer[5]). Green Building XML is available as XML Schema Definition (XSD) file which makes it accessible for web-related applications. Because of the use of XML instead of ISO-STEP as data format, the structure of the model is less expressive but also inherently simpler than STEP-based IFC[6]. Energy-critical parameters that may be useful at building operation time are represented in gbXML to a sufficient extent. Data are not intended to be transferred between CAD and energy tools back and forth, which means that the format does not contain enough information to reintegrate the model into a CAD environment. For example, a wall is merely modeled as surface that separates two spaces than a fully-fledged building component. Such simplifications inherently make gbXML a lot easier to apply for the proposed building ontology and associated smart home system, which has no use for information like coordinates and the exact shape of components only needed to transfer the building description back to CAD programs. In this context, an abstraction from certain characteristics of the building and its structure (i.e., quantitative knowledge, cf. Section 2.2) is advantageous in the interest of a concise model only focusing on details necessary for its scope. At the same time, because of its domain being energy simulation, gbXML naturally comes with a lot of useful values allowing simulation and calculation of thermal loads and processes.
Resulting from the information stored in these two prevalent Building Information Models and their respective complexity, the gbXML format deems to be the better choice over IFC, as it is simpler and specializes on data relevant for the calculation of energy consumption. gbXML contains parameters which are extremely valuable for the comprehensive knowledge base of a smart home system, while at the same time simplifying the general architectural and structural building representation.

4.2.3 XSL Transformation of the Green Building XML Schema to the Web Ontology Language

gbXML is available as XML Schema (XSD)[7] that contains and explains all values that can be exchanged with the help of this BIM. The schema itself is specified in a very general way, in order to ensure flexibility of the model. Any file written in XML can be validated against this schema while not all of the defined classes are mandatory: the tool vendors decide which items they support. As already mentioned before, the represented parameters reflect the main field of application of gbXML: the interoperability between building design models and energy analysis tools [95]. Therefore, most of the represented data types can also be seen beneficial for energy-optimizing control strategies in an autonomous smart building.
The great variety of different parameters included in the gbXML file are not treated in detail at this point, however, Table 4.2 gives a selection and short description of some important compo-

[2] http://www.autodesk.com/products/autodesk-revit-family/overview
[3] http://usa.autodesk.com/ecotect-analysis/
[4] http://www.graphisoft.com/downloads/ecodesigner/index.html
[5] https://www.bentley.com/en-US/Products/AECOsim/
[6] Note: there also exists an XML variant of IFC named ifcXML which however is very limited compared with the original schema.
[7] http://www.gbxml.org/currentschema.php

Table 4.2: Important building information components available in the gbXML Schema and their description (excerpt from gbXML Schema ver. 5.10)

gbXML element	Description
`Building`	Contains all of the information of a specific building
`BuildingStorey`	Captures the building storey structure
`Construction`	Represents a combination of one or more layers
`Layer`	A layer is the combination of one or more materials
`Material`	The material of a specific layer
`Opening`	An opening area in a surface (e.g., window, door)
`Space`	A space represents a volume enclosed by surfaces
`Surface`	gbXML equivalent to a wall
`Zone`	Different logical zones are modeled with this concept

nents that are retrieved from the schema. A transformation from an XSD schema to RDF/XML through XSLT stylesheets is relatively straightforward (cf. [116]) and can be seen as possible starting point to retrieve a first set of terms for a shared vocabulary. In the present case of building information transformation into a smart home knowledge base, XSLT is favored over a programmatical approach, as this transition usually only occurs once or very infrequently and with XSLT the entire process remains external to the application, i.e., the smart home control (cf. [116]).
For creation of a basic version of the architecture and building physics ontology, the transformation process includes several steps (cf. Figure 4.3). In case of gbXML, the multi-step approach shown in Figure 4.3 facilitates the translation of XSD schema and XML data file, hence general BIM schema and concrete building instance, into an OWL representation which includes model and instance data in one single RDF/XML file. The transition from gbXML to OWL is adapted from [32] which discusses a general XML to OWL mapping and therefore serves as a good starting point. It is similar with respect to the general process, however differs in how the actual transformation is conducted, as not all elements of gbXML are needed for the anticipated field of application. Further, by designing a custom XSLT transformation it could be tailored to the specific schema to be transformed. The following paragraphs exactly describe how this conversion is realized for different elements of the gbXML schema file. After the thorough discourse of the schema also the transformation of a sample XML instantiation is briefly outlined.

4.2.3.1 XSD Transformation

For transforming the XSD schema, almost all elements of the schema are considered useful. The elements that are not treated, and as such are not incorporated in the generated building ontology are `aecXML`, `UtilityRate` and `GeneralGeometry`, as they serve very specialized purposes and did not seem to be important for the envisioned field of application, i.e., smart home operation. The XSD schema data is transferred to an OWL model according to defined patterns for different XSD elements.

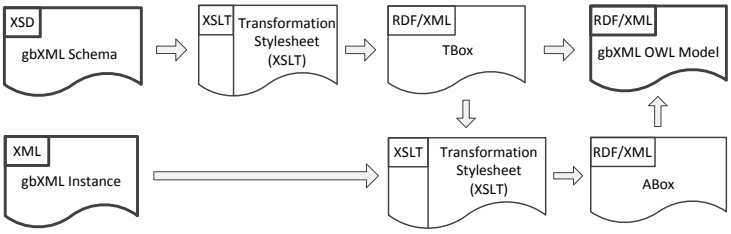

Figure 4.3: gbXML transformation steps (adapted from [32])

xsd:element Firstly, there are several different ways how this schema element type is treated. While some `xsd:element` constructs become OWL properties, others become OWL classes in the created model. The transformation is dependent on their definition in the initial gbXML schema. For example, `xsd:element` statements that do not have child elements but own a *type* attribute are transformed to `owl:DatatypeProperty` elements that have the value of the *type* attribute as range.

gbXML:

```
<xsd:element name="Name" type="xsd:string"/>
```

OWL:

```
<owl:DatatypeProperty rdf:about="#hasNameValue">
  <rdfs:range rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>
```

Also those having only an `xsd:annotation` subelement become datatype properties. In case of an existing direct subelement with value `xsd:simpleType`, also a datatype property is created while a range restriction is added according to the enclosed `xsd:simpleType` element.

gbXML:

```
<xsd:element name="Azimuth">
  <!-- ... -->
  <xsd:simpleType>
    <xsd:restriction base="xsd:double">
      <xsd:minInclusive value="0"/>
      <xsd:maxInclusive value="360"/>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:element>
```

OWL:

```
<owl:DatatypeProperty rdf:about="&ontology;gbBuilding.owl#hasAzimuthValue">
  <rdfs:range>
    <rdfs:Datatype>
      <owl:onDatatype rdf:resource="&xsd;double"/>
      <owl:withRestrictions rdf:parseType="Collection">
        <rdf:Description>
          <xsd:minInclusive rdf:datatype="&xsd;double">0.0</xsd:minInclusive>
        </rdf:Description>
      </owl:withRestrictions>
    </rdfs:Datatype>
```

```
  </rdfs:range>
  <rdfs:range>
    <rdfs:Datatype>
      <owl:onDatatype rdf:resource="&xsd;double"/>
      <owl:withRestrictions rdf:parseType="Collection">
        <rdf:Description>
          <xsd:maxInclusive rdf:datatype="&xsd;double">360.0</xsd:maxInclusive>
        </rdf:Description>
      </owl:withRestrictions>
    </rdfs:Datatype>
  </rdfs:range>
</owl:DatatypeProperty>
```

All other `xsd:element` constructs become `owl:Class` elements in the resulting RDF/XML file. If an `xsd:element` that becomes an `owl:Class` is further a subelement of yet another `xsd:element`, an object property is additionally created for it. In this case, the set of enclosing `xsd:element` items becomes the domain and the current element becomes the range of the property.

gbXML:

```
<xsd:element name="Surface">
  <xsd:complexType>
    <xsd:choice minOccurs="0" maxOccurs="unbounded">
      <!-- ... -->
      <xsd:element ref="Opening" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:choice>
    <!-- ... -->
  </xsd:complexType>
</xsd:element>
```

OWL:

```
<owl:ObjectProperty rdf:about="#containsOpening">
  <rdfs:domain rdf:resource="#Surface"/>
  <rdfs:range rdf:resource="#Opening"/>
</owl:ObjectProperty>
```

Generally, datatype properties are created with prefix “has” and suffix “Value” while object properties are generated with the prefix “contains” in order to create a consistent naming throughout the document and be able to easily differentiate them in the final model.

xsd:attribute All elements of this type become datatype properties in the created OWL model. The *type* attribute of the element again determines the range of the property, while for attributes that occur more than once in the gbXML schema the range is realized as `owl:unionOf` of all different “type” values. In case no “type” attribute but an enclosed `xsd:simpleType` element exist, this element is used to define the range of the created property.

gbXML:

```
<xsd:attribute name="engine">
  <!-- ... -->
  <xsd:simpleType>
    <xsd:restriction base="xsd:NMTOKEN">
      <xsd:enumeration value="DOE2.1e"/>
      <xsd:enumeration value="DOE2.2"/>
      <xsd:enumeration value="EnergyPlus"/>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:attribute>
```

OWL:

```
<owl:DatatypeProperty rdf:about="&ontology;gbBuilding.owl#hasEngineValue">
  <!-- ... -->
  <rdfs:range rdf:resource="&ontology;gbBuilding.owl#engineEnum"/>
</owl:DatatypeProperty>
```

xsd:simpleType The gbXML schema includes a large collection of enumerations, realized as `xsd:simpleType` elements: these become `rdfs:Datatype` enumerations in the OWL model. Further, as already outlined in the preceding paragraph, if the `xsd:simpleType` element is enclosed in an `xsd:element` or an `xsd:attribute` construct it additionally becomes the range of the `owl:DatatypeProperty` created for these elements. In case the `xsd:simpleType` element is an enumeration, the created enumeration datatype acts as range, as seen in the "engine" `xsd:attribute` example. On the other hand, if it includes an `xsd:restriction` element instead of an `xsd:enumeration` as pictured in the "Azimuth" `xsd:element` example before, the datatype property range in the final OWL model is realized as a restriction on the XSD datatype specified in the *base* attribute value of the gbXML `xsd:restriction` element.

gbXML:

```
<xsd:simpleType name="weightUnitEnum">
  <xsd:restriction base="xsd:NMTOKEN">
    <xsd:enumeration value="Pounds"/>
    <xsd:enumeration value="Kilograms"/>
    <xsd:enumeration value="Tons"/>
  </xsd:restriction>
</xsd:simpleType>
```

OWL:

```
<rdfs:Datatype rdf:about="&ontology;gbBuilding.owl#weightUnitEnum">
  <owl:equivalentClass>
    <rdfs:Datatype>
      <owl:oneOf>
        <rdf:Description>
          <rdf:type rdf:resource="&rdf;List"/>
          <rdf:first rdf:datatype="&xsd;string">Kilograms</rdf:first>
          <rdf:rest>
            <rdf:Description>
              <rdf:type rdf:resource="&rdf;List"/>
              <rdf:first rdf:datatype="&xsd;string">Pounds</rdf:first>
              <rdf:rest>
                <rdf:Description>
                  <rdf:type rdf:resource="&rdf;List"/>
                  <rdf:first rdf:datatype="&xsd;string">Tons</rdf:first>
                  <rdf:rest rdf:resource="&rdf;nil"/>
                </rdf:Description>
              </rdf:rest>
            </rdf:Description>
          </rdf:rest>
        </rdf:Description>
      </owl:oneOf>
    </rdfs:Datatype>
  </owl:equivalentClass>
</rdfs:Datatype>
```

xsd:complexType This element describes anonymous superclasses of created named classes, what means that `xsd:complexType` elements generally become `owl:Class` elements in the generated OWL model. There again exist several different ways of how these elements are treated during the XSLT transformation determined by their subelements. In the first case, an *`xsd:simpleContent`* element with including `xsd:extension` is enclosed in the complex type. The *base* attribute of the `xsd:extension` element is then transferred into a datatype property class restriction involving the `hasNativeValue` datatype property with the *base* value as property range and a cardinality (`owl:cardinality`) of one. The functional `hasNativeValue` datatype property is particularly created for this case as it is needed by many elements which just have a value and optionally a unit: therefore, this datatype property serves as a way to describe the value of an element. If in this case child elements of the `xsd:extension` construct exist, additionally an `owl:intersectionOf` restriction is created, while the subelements of the extension (i.e., `xsd:attribute`) are again treated according to the global rule described earlier.

gbXML:

```
<xsd:element name="Conductivity">
  <xsd:complexType>
    <xsd:simpleContent>
      <xsd:extension base="xsd:decimal">
        <xsd:attribute name="unit" type="conductivityUnitEnum" use="required"/>
      </xsd:extension>
    </xsd:simpleContent>
  </xsd:complexType>
</xsd:element>
```

OWL:

```
<owl:Class rdf:about="&ontology;gbBuilding.owl#Conductivity">
  <rdfs:subClassOf>
    <owl:Class>
      <owl:intersectionOf rdf:parseType="Collection">
        <owl:Restriction>
          <owl:onProperty rdf:resource=
                          "&ontology;gbBuilding.owl#hasNativeValue"/>
          <owl:qualifiedCardinality rdf:datatype="&xsd;nonNegativeInteger">1
          </owl:qualifiedCardinality>
          <owl:onDataRange rdf:resource="&xsd;decimal"/>
        </owl:Restriction>
        <owl:Restriction>
          <owl:onProperty rdf:resource="&ontology;gbBuilding.owl#hasUnitValue"/>
          <owl:qualifiedCardinality rdf:datatype="&xsd;nonNegativeInteger">1
          </owl:qualifiedCardinality>
          <owl:onDataRange rdf:resource="&xsd;string"/>
        </owl:Restriction>
      </owl:intersectionOf>
    </owl:Class>
  </rdfs:subClassOf>
</owl:Class>
```

Also for each direct `xsd:attribute` subelement of `xsd:complexType` a datatype property restriction is created. In case a defined *type* attribute value refers to a standard XML Schema datatype, it becomes the range of this datatype property. If otherwise a custom datatype declared in gbXML is referred, or there is no *type* attribute, `xsd:string` is declared as range in the describing axiom. A special handling exists for the `xsd:ID` and `xsd:IDREF` types: as

these types are not defined in RDF semantics[8], the resulting range is also set to `xsd:string`. This way, the relation that exists between ID/IDREF elements defined in the XSD file is lost, however can be re-established through dedicated rule extensions in the ontology (cf. Section 4.2.4). If the *use* attribute of the element has the value "required", the restriction is realized as `owl:cardinality` statement describing that this attribute should occur exactly once. If the *use* attribute has the value "optional" a restriction with `owl:maxCardinality` one and no `owl:minCardinality` is created, denoting a property that may occur optionally.

gbXML:

```
<xsd:element name="Campus">
  <!-- ... -->
  <xsd:complexType>
    <xsd:choice minOccurs="0" maxOccurs="unbounded">
      <xsd:element ref="Name" minOccurs="0"/>
      <xsd:element ref="Description" minOccurs="0"/>
      <!-- ... -->
      <xsd:element ref="Surface" minOccurs="4" maxOccurs="unbounded"/>
      <!-- ... -->
    </xsd:choice>
    <xsd:attribute name="id" type="xsd:ID" use="required"/>
    <!-- ... -->
    <xsd:attribute name="ifcGUID" type="xsd:string" use="optional"/>
  </xsd:complexType>
</xsd:element>
```

OWL:

```
<owl:Class rdf:about="&ontology;gbBuilding.owl#Campus">
  <rdfs:subClassOf>
    <owl:Class>
      <owl:intersectionOf rdf:parseType="Collection">
        <owl:Class>
          <owl:unionOf rdf:parseType="Collection">
          <!-- ... -->
            <owl:Restriction>
              <owl:onProperty rdf:resource=
                              "&ontology;gbBuilding.owl#containsSurface"/>
              <owl:onClass rdf:resource="&ontology;gbBuilding.owl#Surface"/>
              <owl:minQualifiedCardinality rdf:datatype="&xsd;nonNegativeInteger">4
              </owl:minQualifiedCardinality>
            </owl:Restriction>
          <!-- ... -->
          </owl:unionOf>
        </owl:Class>
        <owl:Restriction>
          <owl:onProperty rdf:resource="&ontology;gbBuilding.owl#hasIdValue"/>
          <owl:qualifiedCardinality rdf:datatype="&xsd;nonNegativeInteger">1
          </owl:qualifiedCardinality>
          <owl:onDataRange rdf:resource="&xsd;string"/>
        </owl:Restriction>
        <!-- ... -->
        <owl:Restriction>
          <owl:onProperty rdf:resource="&ontology;gbBuilding.owl#hasIfcGUIDValue"/>
          <owl:maxQualifiedCardinality rdf:datatype="&xsd;nonNegativeInteger">1
          </owl:maxQualifiedCardinality>
          <owl:onDataRange rdf:resource="&xsd;string"/>
        </owl:Restriction>
      </owl:intersectionOf>
    </owl:Class>
```

[8] http://www.w3.org/2011/rdf-wg/wiki/XSD_Datatypes

```
  </rdfs:subClassOf>
  <!-- ... -->
</owl:Class>
```

If there exist other subelements for `xsd:complexType` like the elements `xsd:choice` and `xsd:attribute`, an `owl:intersectionOf` statement is generated to picture the relationship between the created constructs. The subelements of the `xsd:choice` element are in this case connected via an `owl:unionOf` statement.
In case the `xsd:element` statements enclosed in `xsd:complexType` have a subelement hierarchy that involves other constructs than `xsd:annotation` or `xsd:simpleType` (cf. "Surface" `xsd:element` example), it represents the connection to a class as opposed to a simple datatype property. Therefore, in this case an existential object property restriction is created, with restrictions again corresponding to *minOccurs* and *maxOccurs* attribute values of the enclosed elements.

minOccurs, maxOccurs The attributes `minOccurs` and `maxOccurs` are mapped to the corresponding OWL constructs `owl:minCardinality` and `owl:maxCardinality` (cf. "Layer" `xsd:element` example). There exist two special cases in the transformation: first, if the maximum amount of occurrences is unrestricted, the XSD includes the value "unbounded" for the *maxOccurs* attribute. In this case, no maximum restriction is created in the OWL model. Otherwise, if *minOccurs* has the value "0", the element is deemed to be optional and therefore no minimum restriction is created in the OWL model.

xsd:choice The `xsd:choice` element becomes a union element in an anonym OWL superclass, as this element only occurs inside `xsd:complexType` constructs.

gbXML:

```
<xsd:element name="Layer">
  <!-- ... -->
  <xsd:complexType>
    <xsd:choice minOccurs="0" maxOccurs="unbounded">
      <xsd:element ref="Name" minOccurs="0"/>
      <xsd:element ref="Description" minOccurs="0"/>
      <xsd:element ref="Cost" minOccurs="0" maxOccurs="unbounded"/>
      <xsd:element ref="InsideAirFilmResistance" minOccurs="0"/>
      <xsd:element ref="MaterialId" maxOccurs="unbounded"/>
    </xsd:choice>
    <!-- ... -->
  </xsd:complexType>
</xsd:element>
```

OWL:

```
<owl:Class rdf:about="&ontology;gbBuilding.owl#Layer">
  <rdfs:subClassOf>
    <!-- ... -->
    <owl:Class>
      <owl:unionOf rdf:parseType="Collection">
        <owl:Restriction>
          <owl:onProperty rdf:resource=
                          "&ontology;gbBuilding.owl#containsInsideAirFilmResistance"/>
          <owl:onClass rdf:resource=
                       "&ontology;gbBuilding.owl#InsideAirFilmResistance"/>
          <owl:maxQualifiedCardinality rdf:datatype="&xsd;nonNegativeInteger">1
```

```
          </owl:maxQualifiedCardinality>
        </owl:Restriction>
        <owl:Restriction>
          <owl:onProperty rdf:resource="&ontology;gbBuilding.owl#hasDescriptionValue"/>
          <owl:maxQualifiedCardinality rdf:datatype="&xsd;nonNegativeInteger">1
          </owl:maxQualifiedCardinality>
          <owl:onDataRange rdf:resource="&xsd;string"/>
        </owl:Restriction>
        <owl:Restriction>
          <owl:onProperty rdf:resource="&ontology;gbBuilding.owl#hasNameValue"/>
          <owl:maxQualifiedCardinality rdf:datatype="&xsd;nonNegativeInteger">1
          </owl:maxQualifiedCardinality>
          <owl:onDataRange rdf:resource="&xsd;string"/>
        </owl:Restriction>
        <owl:Restriction>
          <owl:onProperty rdf:resource="&ontology;gbBuilding.owl#containsCost"/>
          <owl:onClass rdf:resource="&ontology;gbBuilding.owl#Cost"/>
          <owl:minQualifiedCardinality rdf:datatype="&xsd;nonNegativeInteger">0
          </owl:minQualifiedCardinality>
        </owl:Restriction>
        <owl:Restriction>
          <owl:onProperty rdf:resource="&ontology;gbBuilding.owl#containsMaterialId"/>
          <owl:onClass rdf:resource="&ontology;gbBuilding.owl#MaterialId"/>
          <owl:minQualifiedCardinality rdf:datatype="&xsd;nonNegativeInteger">1
          </owl:minQualifiedCardinality>
        </owl:Restriction>
      </owl:unionOf>
    </owl:Class>
    <!-- ... -->
  </rdfs:subClassOf>
  <!-- ... -->
</owl:Class>
```

4.2.3.2 XML Instance Transformation

While the OWL model is directly transformed from the gbXML schema to RDF/XML, the building instance to be converted comes from a concrete XML exchange file, which can for example be retrieved from a CAD system or an energy analysis tool. The test building created for this work is modeled in Autodesk Revit Architecture 2010 and enhanced with building physics properties in Ecotect Analysis 2010[9]. Again, as already in the schema transformation, some elements of the building instance that solely have relevance for CAD graphical representation (e.g., planar geometry, cartesian point, coordinate) are not needed for the building operation phase: therefore they are omitted, in order to keep the created building model concise. All gbXML element instances that become individuals (i.e., class members) in the OWL model are generically modeled as member of the `owl:Thing` class, while element instances that become simple datatype properties are generated accordingly. Subsequently, individuals can be classified into the appropriate subconcept of `owl:Thing` by the OWL reasoning mechanism.

gbXML:

```
<Surface id="obj000" surfaceType="ExteriorWall" constructionIdRef="con098"
         exposedToSun="true">
  <Name>Obj000</Name>
  <AdjacentSpaceId spaceIdRef="zon001" />
  <RectangularGeometry>
```

[9] http://usa.autodesk.com/

```
    <Azimuth>90.000003</Azimuth>
    <!-- ... -->
    <Tilt>90.000003</Tilt>
    <Height>3.000000</Height>
    <Width>6.400000</Width>
  </RectangularGeometry>
  <!-- ... -->
  <Opening id="obj001" openingType="OperableWindow" windowTypeIdRef="win059">
    <!-- ... -->
  </Opening>
    <CADObjectId>OBJ:00000</CADObjectId>
</Surface>
```

OWL:

```
<owl:Thing rdf:about="&ontology;gbBuilding.owl#Surface_ID=obj000">
    <rdf:type rdf:resource="&owl;NamedIndividual"/>
    <hasSurfaceTypeValue rdf:datatype="&xsd;string">ExteriorWall</hasSurfaceTypeValue>
    <rdfs:comment rdf:datatype="&xsd;string">Name of Surface: Obj000</rdfs:comment>
    <hasNameValue rdf:datatype="&xsd;string">Obj000</hasNameValue>
    <hasConstructionIdRefValue rdf:datatype="&xsd;string">con098
    </hasConstructionIdRefValue>
    <hasIdValue rdf:datatype="&xsd;string">obj000</hasIdValue>
    <hasExposedToSunValue rdf:datatype="&xsd;boolean">true</hasExposedToSunValue>
    <containsAdjacentSpaceId rdf:resource=
                             "&ontology;gbBuilding.owl#AdjacentSpaceId_ID=d6e717"/>
    <containsCADObjectId rdf:resource=
                             "&ontology;gbBuilding.owl#CADObjectId_ID=d6e923"/>
    <containsOpening rdf:resource="&ontology;gbBuilding.owl#Opening_ID=obj001"/>
    <containsRectangularGeometry rdf:resource=
                            "&ontology;gbBuilding.owl#RectangularGeometry_ID=d6e719"/>
</owl:Thing>
```

This example shows a specific surface in the modeled building and its representation in the created RDF/XML file. In case enclosed subelements can be classified as members of OWL classes created during schema transformation (e.g., `Opening`, `RectangularGeometry`, `AdjacentSpaceId`) they are connected to the shown instance through object properties while being represented as separate individuals as well. In other cases, like the `Name` element simple datatype properties are created. The classification of individuals is accomplished through range and domain restrictions of associated object properties. For example, as shown before, the `containsOpening` object property is defined to solely have surfaces as domain and openings as range. These restrictive domain and range declarations allow the DL reasoner to conclude that all instances which actually have a `containsOpening` relation are of type `Surface`.

4.2.3.3 Summary

The described schema mappings are summarized in Table 4.3. Firstly, customized datatype enums defined in gbXML are produced. Then, needed datatype and object properties are created, anonymous classes and axioms describing named classes are generated and finally, concepts are made disjoint to each other. The mappings of the transformation significantly differentiate from the ones suggested in [32] in order to simplify the retrieved OWL model and reduce the amount of created classes. At the same time, through the definition of a tailored process the possibility of adjustment according to the gbXML schema arises. The proposed approach allows to create a much conciser model than it would be possible using a more generalized mapping procedure. It permits both, solely transforming the XSD schema and thus only creating the TBox of the

Table 4.3: General mapping from XSD schema elements to OWL

XSD	OWL
`xsd:element` with no subelements and no *type* attribute	`owl:Class`
`xsd:element` containing elements other than `xsd:annotation` or `xsd:simpleType`	`owl:Class`
`xsd:element` as subelement of another `xsd:element`	`owl:Class` and associated `owl:DatatypeProperty`
`xsd:element` containing no subelements and *type* attribute	`owl:DatatypeProperty`
`xsd:element` containing only one `xsd:annotation` subelement	`owl:DatatypeProperty`
`xsd:element` containing a direct `xsd:simpleType` subelement	`owl:DatatypeProperty`
`xsd:attribute`	`owl:DatatypeProperty`
`xsd:simpleType`	`rdfs:Datatype`
`xsd:complexType`	`owl:Class`
`xsd:minOccurs`, `xsd:maxOccurs`	`owl:minCardinality`, `owl:maxCardinality`
`xsd:choice`	`owl:unionOf`
`xsd:annotation`	`rdfs:comment`
`xsd:extension` *base* attribute	`owl:DatatypeProperty` "hasNativeValue"

ontology or additionally including a concrete XML instantiation of the XSD schema and thus generating TBox and ABox. The retrieved structure of the generated OWL model is very simple due to limited expressivity of the XSD schema and the necessity of an automated approach. For example, while XSD elements are interpreted as OWL concepts, and in some cases additionally as OWL object properties, XSD attributes are always realized as OWL datatype properties (cf. Table 4.3). This results in a compact translation which preserves most of the information defined in the original gbXML schema.
In a second step, a specific XML instantiation of the schema (i.e., a concrete building project) is translated into OWL individuals with another stylesheet in order to populate the created OWL model. This XSLT transformation takes the general OWL model created in the first step as well as a Green Building XML file as input and produces the ABox of the ontology. In this respect, the OWL model that serves as input can be modified involving a complexity that goes beyond information retrieved through the schema transformation: as long as the basic classes created in the gbXML schema translation step are existent in the OWL model, an arbitrary complex version can be used as input file for the ABox transformation still permitting the successful creation of individuals and their association with appropriate classes. This makes the extension of the

knowledge store possible while not interfering with the ABox creation procedure. Almost all of the information stored in the gbXML file describing the building project is identified to be relevant for energy efficiency considerations and thus transferred to OWL, while excluding only special schema elements and CAD coordinate instance values which deemed to be unessential for the operational phase.
As a last step of the process, the two generated files (TBox and ABox) are merged into one OWL file resulting in a fully populated OWL building model. This fundamental OWL representation of building data retrieved from gbXML may already be used as basic model for building information or can be seen as basis for the creation of a sophisticated building information ontology representing building physics and architectural parameters.

4.2.4 OWL Conceptualization of Building Physics

As already motivated in the previous paragraphs, when considering the optimization of energy consumption in a building, the general structure of the building, its orientation and building physics play an important role that cannot be neglected. In this case, Semantic Web modeling can be used on the one hand to take away complexity from the system relying on it while on the other hand describing the relations between parameters in a machine-interpretable form. Modeling the building and its characteristics with the help of the Web Ontology Language brings several advantages: the possibilities that arise through the use of a complex, logic-based language like OWL can aid the creation of a model of the world that accurately reflects reality and represents knowledge about the building environment. The inherent logic can be used to infer missing information in order to complete the representation of the building. Particularly for autonomous building systems it is essential to represent information available about the building and its characteristics as complete as possible. As this information can seldom be retrieved to a full extent at the stage of building usage, it is desirable to reuse information about the building already collected in the design and construction phase through architects, planners and building physicists as shown in previous sections. The described inclusion of information collected in gbXML forms a fundamental basis for the envisioned semantic layer. The idea of the created architecture and building physics ontology is to integrate the information retrieved from gbXML into a smart home in order to benefit from already known information about the building. This information is manifold, but as one of the main use case goals of the envisioned smart home system is to provide an energy-efficient operation (cf. Chapter 3), the representation of building physics parameters as well as relations of rooms, walls, windows and doors are most important. Then again while the storage of the building orientation is important, detailed information about the building model position in a CAD coordinate system is unnecessary for the proposed utilization and can therefore be neglected.
As the information retrieved from gbXML is extracted from an ordinary XML file following the gbXML schema, the initial OWL hierarchy is flat: this results from the fact that subsumption hierarchies as they may be defined in OWL cannot generally be presumed in XML. Further, because gbXML describes a format for interoperability between development tools of the AEC domain, most of the elements are described in a very general way, what means, that many describing elements are optional. For example, the definition of the Material element in gbXML includes parameters like density, thickness, conductivity or R-value among others, however the

only required parameter is an ID. This great amount of optional elements makes the definitions retrieved from gbXML rather unsuitable for the automatic deduction of a class hierarchy. As previously described `xsd:complexType` elements of the XSD schema are still transformed into OWL axioms. Because of the open-world assumption these axioms and their large amount of optional dependencies represented as `owl:unionOf` relation are not necessarily useful for the OWL reasoning itself, however, besides formally describing the relationships to other concepts and datatypes, they serve as a reference point for a human observer to comprehend the meaning and intended usage of a specific OWL concept. For reasoning performance considerations, it would be possible to store these axioms as `rdfs:comment` annotations and as such exclude them from DL reasoning altogether. While for a human observer this definition would be sufficient to comprehend the meaning of a concept, information would then not be available to the reasoning algorithm anymore. On that account, it is decided to keep the axioms in concept descriptions.
The following elaborations show how the created ontology can enhance information retrieved from the gbXML building information model with the help of inherent logical reasoning of the applied Semantic Web language and the design of logical rules for advanced inferences.

4.2.4.1 Upper Level Building Domain Conceptualization

As a first step, an asserted OWL subsumption hierarchy is manually created in order to classify the retrieved gbXML parameters. The whole hierarchy is deemed to represent a shared vocabulary or upper ontology for Building Information Modeling. As shown in Figure 4.4, the top level concepts include the most general groups of data represented in gbXML while subhierarchies define partitions of subgroups. Some of these upper ontology elements for gbXML are directly taken from the gbXML schema, while others are created in order to group retrieved classes. The *campus* concept in gbXML includes facts about the exterior environment of the building as well as other information specifically relevant for energy analysis. Further, one or more *buildings* are defined for a campus site. This element also stems from gbXML where it is mainly used to represent everything that is known about the building, from general facts like address and description to its geometric shape. In the context of the created shared vocabulary, the campus class is therefore used to describe the compound of one or more buildings and possibly the surroundings while the building class represents information about the building itself. Each building has several associated building elements (`BuildingElement`), a class defined to group general relevant components that together make up a building, such as building storeys, materials and surfaces (i.e., walls). `Equipment` is another top-level class created to group certain concepts as gbXML also allows to describe different types of equipment (e.g., air loops, lighting system) for energy simulation. The `Parameter` top-level concept finally represents the main hierarchy of the created shared vocabulary: in this case, all parameters that may be specified for buildings and equipment are grouped according to their usage type (e.g., geometric, temporal, building) as well as building life cycle phase. The relevance of each parameter for a specific phase can be classified through the *ParameterSignificance* group. Finally, the concept `IdentifierElement` is an auxiliary element to group ID elements that are created through the XSLT transformation of gbXML. Generally, this element can be used to represent ID attributes for XML-based

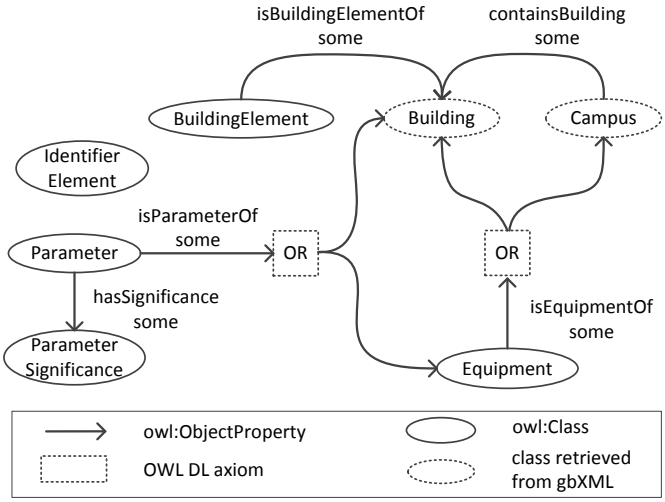

Figure 4.4: Top-level concepts and relating properties of an upper level building domain conceptualization based on gbXML

schemas. Typically a lot of information in OWL ontologies is represented in properties. The created transformation from gbXML already creates a wide variety of properties relating specific gbXML elements to each other. However, again, the possibilities of automatically creating properties from an XML schema are limited to the expressivity of the schema. In order to define connections between the defined top-level elements, a well-defined set of object properties is created, including connections which are not defined in gbXML and thus cannot be retrieved directly through transformation. In this respect, the `isBuildingElementOf` relation directly relates a building element to a specific building on the campus. Each equipment may be part of a campus or a building and therefore another property `isEquipmentOf` is defined. It is further necessary to relate each parameter represented in this ontology to a building or equipment which is realized with the `isParameterOf` relation. Several parameters retrieved from gbXML are just relevant for the building performance analysis phase, e.g., parameters for specific load calculation methods, while others like the blower door value or various energy-related parameters about equipment also prove to be useful for the operation phase of the building. Therefore, a relation of parameters to the hierarchy representing parameter significance is created by means of the property `hasSignificance`.

4.2.4.2 gbXML Classification

In order to classify concepts retrieved from gbXML, they are added to one of the subgroups defined in the upper level building ontology and shown in Figure 4.5. While an automated approach would be possible, in the current process the concepts are manually asserted to these hierarchies.

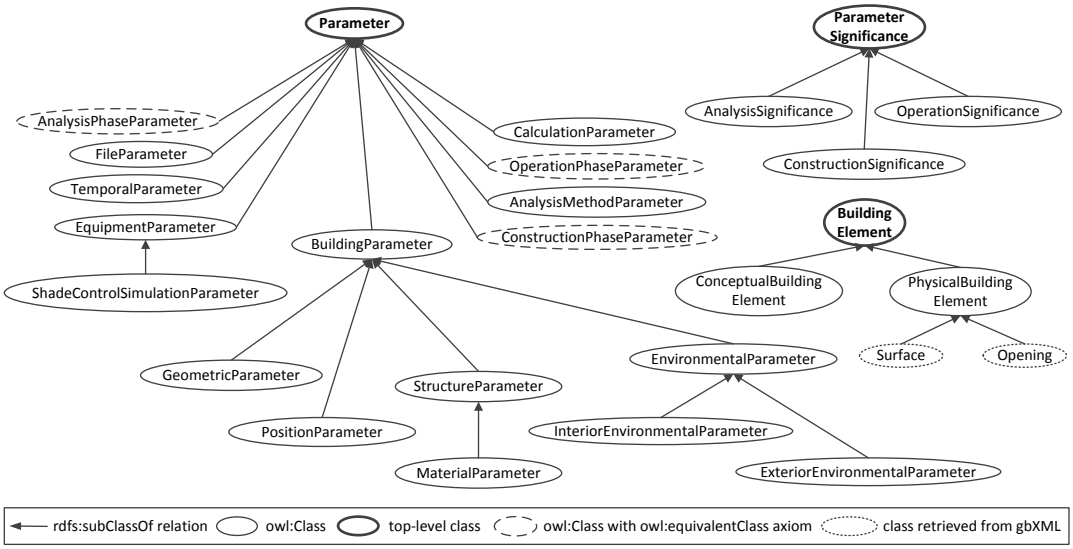

Figure 4.5: Building ontology module shared vocabulary hierarchies

Building Elements All physical as well as conceptual elements of gbXML that are related to the description of the building and its structure are summarized in this category. Examples for *physical elements* obtained from gbXML are surfaces (i.e., walls), construction and layer of these surfaces as well as the material that constructions are made of. Further, also openings included in surfaces (i.e., windows, doors) as well as spaces, as representation of volumes enclosed by surfaces (i.e., rooms) are defined as pyhsical building elements. Then again, *conceptual elements* are elements that do not necessarily represent real-world entities and can be seen as auxiliary elements that are modeled in gbXML to express some concepts needed for simulation or to add an additional expressivity to the model. Examples would be zones represented as members of the `Zone` concept which describe a partition of the building independent of rooms, or the `SpaceBoundary` concept which is designed in gbXML to represent the connection between surfaces and their adjacent spaces.

Building Information Parameters The parameter subhierarchy is mainly split into two types: life cycle related and application related. While application related parameters are directly imported from gbXML and are therefore represented in a manually asserted hierarchy, the life cycle related classes are defined in the shared vocabulary to additionally categorize the represented parameters according to the life cycle stage for which they are relevant.

Life cycle related At the moment there are three *life cycle* groups reflecting different `ParameterSignificance` subgroups (cf. Table 4.4). Life cycle related concepts are defined using `owl:equivalentClass` axioms, and therefore, logical reasoning can be ap-

Table 4.4: Life cycle related parameter concepts and their corresponding equivalence axioms

Parameter Concept	Equivalence Axiom
AnalysisPhaseParameter	Parameter and hasSignificance some AnalysisSignificance
ConstructionPhaseParameter	Parameter and hasSignificance some ConstructionSignificance
OperationPhaseParameter	Parameter and hasSignificance some OperationSignificance

plied and application related parameters with proper axioms can be inferred to be members of one or more of these groups. This additional classification allows to instantly identify parameters that are solely important for a specific life cycle phase, as many parameters retrieved from a Building Information Model are solely used for simulation (i.e., in the building analysis/design phase). Those parameters can then be classified as not being relevant for building operation. For example, while a geometric parameter like the area of a room has a significance in all three phase, the results of a simulation captured in the `Results` class are only important in the analysis phase. With the `hasSignificance` object property and the appropriate axioms, `Area` is therefore inferred to be a subgroup of `AnalysisPhaseParameter`, `ConstructionPhaseParameter` and `OperationPhaseParameter`, while `Results` can only be identified as subclass of `AnalysisPhaseParameter`.

Application related While one gbXML concept can be subclass of more than one life cycle related parameter class, generally there only exists one main application for a parameter. Therefore, in order to better categorize different parameters represented in gbXML, a detailed *application related* parameter hierarchy is provided in the designed shared vocabulary as shown in Figure 4.5. All concepts identified during the gbXML transformation that represent building or equipment parameters are added to the particular group which indicates their main category:

- **Calculation parameters** This class includes general parameters that may be used for different simulation-related parameters needed for calculating performance and cost of equipment. In the case of gbXML, costs of building elements (`Cost`) and classes needed for the representation of algebraic expressions (`Equation`, `IndependentVariable`, `DependentVariable`) are modeled as calculation parameters. Parameters in this group are mainly significant in the building design phase.

- **Building parameters** Parameters that are related to the building itself and its surroundings are represented in this concept. *Environmental parameters* as first direct subgroup are all parameters that are describing the interior or exterior environment of a building. Exterior environmental parameters are again mainly used for simulations and for example

include simulation parameters for outdoor conditions (e.g., cooling degree days, weather) and environment (e.g., vegetation, biomass density). For the interior environment, again a large amount of simulation parameters is defined in gbXML, while in this case it becomes clear why a classification according to phase significance is needed: values for illuminance or temperature defined in gbXML only have a relevance at building design time while operating values for these parameter types are defined in the ontology defining sensors (e.g., light sensor, temperature sensor) and their current states (cf. Section 4.3). On the other hand, static values like blower door value or people heat gain in combination with current values (e.g., people number count of a specific room) also can be used to optimize the building at operation time. The *geometric parameter* group comprises classes that are needed to describe the items relevant for representing geometric forms in the building model. As such it mainly includes concepts derived from CAD programs, e.g., height and length of a building element or the level of a particular building storey. The ***position parameter*** group is created to contain information about the site's global position such as the location class describing the placement using height above the sea level (i.e., elevation), latitude and longitude. Finally, *structure parameters* as subclass of building parameters covers information items that depict the general structure of the building. *Material parameters* as a subclass of structure parameters encloses those values and classes describing used materials in detail (e.g., reflectance, absorptance, U-value).

- **Equipment parameters** This group is concerned with parameters regarding the design and operation of equipment in the represented building. As several elements in the gbXML schema are solely used for *shade control simulation*, an own subconcept is provided for this group of parameters. While equipment parameters in general may be significant for all phases of the building life cycle, shade control simulation parameters are only significant for the analysis phase.

- **Analysis method parameters** This concept encloses items that are solely significant within a specific design method used for building analysis.

- **Temporal parameters** This class groups elements relevant to model time concepts in the building information model. The parameters are needed for example to describe schedules for energy simulations and as such are also only significant for the analysis of a building at design time.

- **File parameters** In this subconcept of building parameters auxiliary parameters that are used to describe the gbXML file itself and properties like its creator, modification date and name of the creation program are represented.

Building Parameter Significance This top-level concept currently identifies three different stages: analysis, construction and operation. The parameter significance concept is provided because a BIM generally spans several stages of the building life cycle and as already illustrated not all parameters available for energy analysis also necessarily have an influence on the operation of a building. As Figure 4.5 shows, single building life cycle stages are modeled as subclasses.

This modeling decision makes it possible to define a more detailed partition of building parameters by defining additional subhierarchies if needed. it would for example allow to specify that a particular parameter is only significant for a short period of time in the analysis phase enhancing the expressivity of the model.

Building Equipment Additional to building elements, the gbXML schema also includes elements describing building equipment for the HVAC and lighting domain which are summarized in this concept. While for a thorough representation of building facilities and applications the Energy & Resources ontology (cf. Section 4.3) is provided, equipment information gathered in gbXML is still included. The building information model in this case provides elements to describe parameters of air loop equipment, hydronic loop equipment and the lighting system as well as general internal equipment. The elements are defined in a very generalized form in order to classify any type of equipment while no detailed equipment hierarchy is described. In the designed smart home ontology parameters defined in gbXML can still be regarded useful to enrich the resource ontology with static information retrieved from the building information domain.

4.2.4.3 Reasoning in the Building Ontology

Additionally to describing the information retrieved from gbXML with the Web Ontology Language also subtle Description Logic reasoning is used to enhance retrieved facts and create additional information. In this case, two different forms of reasoning are exploited: common OWL as well as SWRL reasoning. This section describes some of the advantages that arise through different forms of reasoning when using information from a building information model for the realization of a building ontology.

OWL Reasoning In the case of the defined ontology for Architecture & Building Physics, standard OWL reasoning is mainly utilized for individual reasoning. As a large amount of elements retrieved from the concrete model become individuals in the ontology, they need to be classified into their corresponding concepts generated from the gbXML schema. In order to keep the conversion short and simple and make it unnecessary to manually assert class dependencies in the transformation process, these individuals are defined as members of the common superclass `owl:Thing`. To classify them as members of an appropriate subclasses of `owl:Thing`, the reasoning mechanism of OWL is used. In this respect, the defined `rdfs:domain` and `rdfs:range` descriptions fixed during the transformation are useful to identify the corresponding class. For the two individuals `Concrete_Insulation` and `Insulation_Material` in Figure 4.6 for example, a variety of connections to other created individuals are defined through object properties. For each of these properties domain and range are identified in order to define associated classes. In this regard, the `containsR-value` object property is defined as follows:

```
<owl:ObjectProperty rdf:about="&ontology;BuildingOntology.owl#containsR-value">
  <rdfs:domain rdf:resource="&ontology;BuildingOntology.owl#Material"/>
  <rdfs:range rdf:resource="&ontology;BuildingOntology.owl#R-value"/>
</owl:ObjectProperty>
```

Therefore, the connection with this property already allows to classify the insulation material instance as member of the `Material` class (cf. Figure 4.7) as well as the concrete R-value instance as member of the `R-Value` concept. These classifications are not only dependent on single properties as dependencies like the one illustrated can be found throughout the whole ontology. Similarly to the described example, all other inferred `rdf:type` relations shown in Figure 4.6 can be deduced.

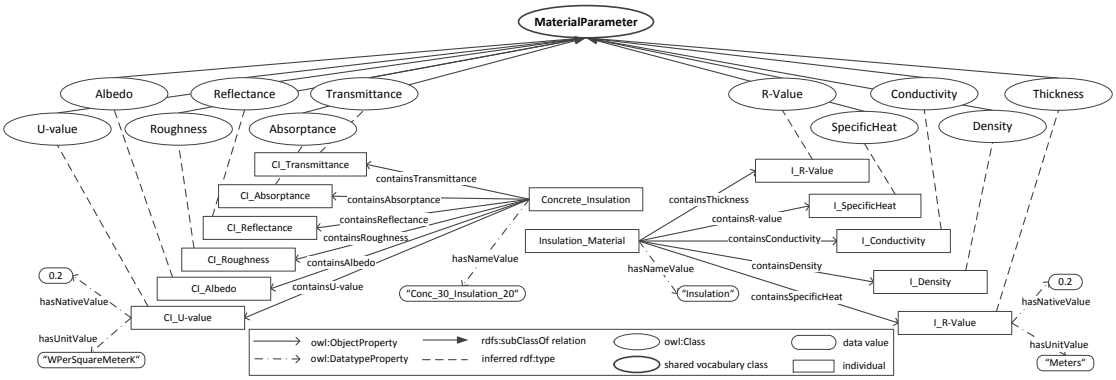

Figure 4.6: Inferred class relationships of individuals associated with a construction (`Concrete_Insulation`) and a material (`Insulation_Material`).

SWRL Rule Extensions As already explained in Section 2.4.1, SWRL rules are used to extend the reasoning capabilities of OWL. Because of the fact that the data format used by gbXML is ordinary XML and as such is not as expressive as the OWL language, logical rules are used in the constructed building ontology to extend the information representation and also simplify dependencies derived from gbXML. One main application for rules in this particular case is the mechanism that is used by XML to define references between elements: An element with a unique ID is referenced via the XML schema simple type IDREF and thus creates a simple form to describe relations between objects in an XML document. In the created OWL ontology it is however desired to be able to directly relate concepts to each other without the need for matching ID/IDREF pairs in a user-defined query. In this case, SWRL rules may be used to connect concepts that are described with a reference relation in the original gbXML document. Listing 4.1 shows an example that models the connection between a construction and its associated layer.

Listing 4.1: SWRL Rule to connect entities referenced to each other via ID and IDREF attributes in the original gbXML document.

```
hasDefinedLayer:

        Construction(?x1), Layer(?x4),
        containsLayerId(?x1, ?x2),
        hasLayerIdRefValue(?x2, ?x3),
        hasIdValue(?x4, ?x3) -> hasDefinedLayer(?x1, ?x4)
```

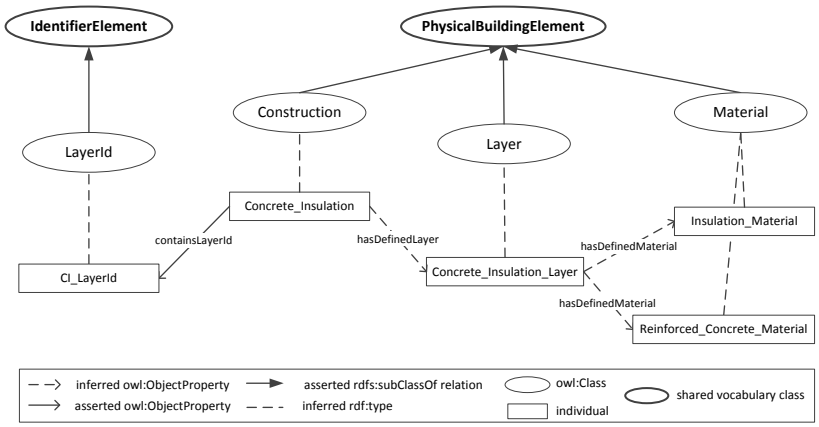

Figure 4.7: Example showing the connection between a particular construction, its layer and two associated materials

Basically, the rule validates if the values of the `hasIdValue` datatype property of a `Layer` instance and the `hasLayerIdRef` datatype property of a `LayerId` individual connected to a specific `Construction` instance match. If this connection is detected, a new relation `hasDefinedLayer` is added to the knowledge base, describing the fact that this particular construction has an association to the respective layer. A similar rule can be defined for the connection between layers and materials which for a specific construction in further consequence may lead to dependencies as shown in Figure 4.7: in that case, the construction has one associated layer, which itself is made up of two different materials, insulation and concrete. While in this case the two materials belong to one single layer, naturally it is also possible to split them in two distinct layers[10].
A further example for the need of rule extensions is the definition of inverse relations. Considering the necessity to represent adjacent walls for a particular space, in gbXML surfaces are generally related to spaces via the `AdjacentSpaceId` element. The other direction, hence the relation from spaces to surfaces, may be realized with the help of the `SpaceBoundary` element (cf. [223]). While the schema provides the possibility to define both relation directions, none of them is mandatory and not all of the currently available software solutions also use them. For example, only the first relation exists in the test building gbXML instance created by the Ecotect software tool.

Listing 4.2: SWRL Rule to determine the adjacent walls of a room

```
hasDefinedAdjacentSurface:

        Space(?sp1),Surface(?su1),
        hasIdValue(?sp1,?id1),
        containsAdjacentSpaceId(?su1,?aid1),
        hasSpaceIdRefValue(?aid1, ?id1) -> hasDefinedAdjacentSurface(?sp1, ?su1)
```

[10]For both defined relations corresponding inverse properties are defined (`isDefinedLayerOf`, `isDefinedMaterialOf`) which are omitted in the figure for brevity.

Listing 4.3: SWRL Rule to define two adjacent spaces

```
hasDefinedAdjacentSpace:

        Surface(?su1), Space(?sp1), Space(?sp2),
        hasDefinedAdjacentSurface(?sp1, ?su1),
        hasDefinedAdjacentSurface(?sp2, ?su1),
        hasIdValue(?sp1, ?id1),
        hasIdValue(?sp2, ?id2),
        notEqual(?id1, ?id2) -> hasDefinedAdjacentSpace(?sp1, ?sp2)
```

In this case, the application of an ontology suits perfectly in order to describe and deduce missing relations. As it is a precondition for a sophisticated building model to represent appropriate relations to walls, ceilings and floors for defined spaces [142], a rule to infer this relation through the reasoning mechanism of the ontology is provided (cf. Listing 4.2). This rule extends the gbXML building information ontology with the property `hasDefinedAdjacentSurface` describing the surfaces (e.g., walls, floor, ceiling) that are adjacent to a specific space. That way, the second direction to relate space and surface individuals can be inferred by the reasoning mechanism even without the existence of a `SpaceBoundary` element.
The third example to illustrate rule extensions in the building ontology identifies if two spaces are adjacent to each other. In newer versions of gbXML[11] this relationship may be created with the `oppositeIdRef` attribute of the `SpaceBoundary` element which again is not mandatory. Therefore, the rule illustrated in Listing 4.3 which builds on the previously shown rule (cf. Listing 4.2) can be used to identify two adjacent spaces by essentially examining two spaces for a common, shared surface. In order to assure that the two identified spaces are indeed different spaces, the rule additionally makes use of the `swrlb:notEqual` SWRL built-in [253]. In case the rule is satisfied by two individuals in the knowledge base, it adds an additional relation (`hasDefinedAdjacentSpace`) to express that these two particular spaces are adjacent to each other. Again, as already with the last example, the rule can be seen as addition to the ontology that completes knowledge about the building not explicitly stated in the source, the building information model.
Another rule that builds on the adjacent surface rule is pictured in Listing 4.4. In the original gbXML model, surfaces are associated with the campus instead of the building element. To overcome this peculiarity, the shown SWRL rule modifies the `containsSurface` object property retrieved from gbXML in a way that each surface adjacent to a space of a particular building is connected to the building via the `containsSurface` property.

Listing 4.4: SWRL Rule to create a direct connection between buildings and surfaces

```
containsSurface:

        Surface(?x1), Building(?x3), Space(?x2),
        containsSpace(?x3, ?x2),
        hasDefinedAdjacentSurface(?x2, ?x1) -> containsSurface(?x3, ?x1)
```

As this rule redefines an existing object property retrieved from gbXML instead of creating a new one, additionally to defining the rule, the `rdfs:domain` of the property needs to be adjusted: according to the original gbXML transformation, the domain is limited to elements of type "`Campus`", however, in order to satisfy the constraints of the created rule the domain of

[11] Version 5.00 and higher.

this object property needs to be extended to "`Building` **`or`** `Campus`".
The provided examples show how the reasoning mechanism of OWL and SWRL rules may be used to extend the knowledge about a building by computationally inferring missing relations between individuals retrieved from a concrete gbXML instantiation.

4.2.5 Ontology Metrics

Table 4.5 shows a summary of the created ontologies in this domain. For the envisioned Architecture & Building Physics ontology on the one hand a simple shared vocabulary is created that encloses a well-defined set of concepts and object properties. There are no datatype properties and also no instances defined in this ontology, as it merely serves to create a general classification for elements and parameters interesting in the building domain.

Table 4.5: Metrics of the Architecture & Building Physics ontology module

		Shared Vocabulary for Architecture & Building Physics Ontology	**Architecture & Building Physics Ontology**
TBox	Named Classes	31	263
	Object Properties	5	238
	Datatype Properties	0	207
ABox	Instances	0	0

Further, Table 4.5 shows the metrics of the complete Architecture & Building Physics ontology including the concepts and properties of the shared vocabulary. The major part of the created classes represents different types of building parameters. The rest of the concepts is used to represent the building and its elements. The large amount of object and datatype properties arises from the automatic extraction of classes and properties from the gbXML building information model. Additionally, not shown in the overview, a small amount of SWRL rules is provided in order to prove the usability of a rule extension in this context.

4.3 Energy & Resources

In current residential homes, occupants are often not aware about how much energy is consumed at a certain point in time. While smart metering is about to become extensively introduced in the European Union [119], in the general case a smart meter is only a digitized version of current mechanical meters. As such it is capable of giving real-time feedback about the electricity use of a building to residents and allow energy providers to remotely switch off or limit the energy use of single energy consumers [244]. Hence, in this case information about energy consumption is limited to a general, building-wide value not giving feedback of consumption on the device level.
Generally, as pointed out in surveys and projects like [240, 137] it can be seen as a prerequisite

of a smart home to make energy consumption of single devices available either for direct feedback to occupants or for sophisticated, autonomous control mechanisms. While the feedback of a smart meter can already be used to give the occupants a general idea of how much energy is consumed, it proves rather difficult to isolate major energy consumers from measures taken by smart meters. There exist projects showing the feasibility of this approach for small test environments (e.g., [23]), however especially in buildings with many electrical appliances such an approach will most certainly fail to identify the variety of different appliances: consumers that show the same steady state profile for example make it hard to distinguish different electrical appliances. Similarly, also an association of a device to a specific room or zone is in this case only possible if each of the devices show individual energy-consumption load profiles. In future smart homes, therefore it is beneficial to measure the energy consumption at the source i.e., at the devices themselves. Currently, there exist different possibilities to reach device-level energy metering. On the one hand, it is possible to install a smart metering infrastructure on a socket level. One popular example in this case would be the Plogg[12] smart plug environment: this system allows the monitoring of devices and metering of generated or consumed kWh which can be read out wirelessly using the ZigBee protocol or Bluetooth [137]. On the other hand, installed building automation systems may be used to monitor the energy demand of single devices or device groups by adding specific components for energy measurement (e.g., KNX energy modules [3]). Alternatively, energy measuring devices of a dedicated field bus like M-Bus [76] may be used to retrieve electricity, gas, heat or water consumption figures and provide them in machine-readable form to a building control system. In future homes, Internet-connected devices may also natively support the reading of their current energy demand.

Another important dimension with respect to energy information is energy production. In this connection, energy may be locally produced by renewable energy generation equipment or provided by external energy suppliers. While local energy producers such as solar panels or a geothermal heat pump as well as their states and functionality naturally need to be depicted in a smart home, it is also necessary to provide a representation for energy suppliers and the energy spot price. For a truly smart home, it needs to be possible to react on available excess energy in the energy grid. As already mentioned in [202] a shortcoming that building automation systems are facing today is that the promising integration of household appliances (*white goods*) and consumer electronics (*brown goods*) is not happening pervasively, if at all. The reason is that the consolidation of these devices is not trivial at the physical layer (e.g., wired-wireless), nor at the network layer (communication), neither at the application level (data semantics). Additionally, such an extension of the building automation system scope obviously increases the overall system complexity. In this case, especially the integration of household appliances and entertainment devices into a homogeneous system is far from being trivial. Even after integration of all devices is achieved, operating them energy efficiently is another big challenge due to different usage paradigms, operation modes and interfaces. For full resource conservation, it is mandatory to include major energy consumers such as household appliances in novel control strategies of the automation system.

With the great variety of integrated devices in the home, it becomes important to picture them according to their type, functionality and possible states. Such a representation not only gives

[12] www.plogg.co.uk

a unique view on the facilities infrastructure, but may also be used to integrate heterogeneous building networks into one single system. Naturally, apart from control information, also the energy consumption and energy production need to be pictured: this information enables a global control system to consider it for energy optimization strategies. Relying on an ontology semantically describing building automation resources and household appliances, newly attached facilities can easily be categorized and integrated into the system without complete information about functionality and control capabilities.
Many devices also waste energy when they are currently not in use e.g., during their stand-by times. Reducing these idle-times may lead to large savings if done on a building level [163]. To become aware of idle times, the SHS should be aware of occupancy information as well as a notion when different facilities are likely to be used (e.g., a coffee machine is likely to be used between 6 and 9AM) represented in another part of the smart home ontology (cf. Section 4.4). Especially in the case of brown goods like TVs and other leisure facilities it is feasible to unlink them from the power grid during times when the home is not occupied and to just bring them back into stand-by mode when usage is expected. For the energy-intensive operation of white goods like dishwashers or washing machines it is beneficial to know how tasks should be scheduled with respect to the energy supply side. This way, peak loads on the power grid can be reduced and at the same time the environmentally friendliest energy provider can be chosen.

4.3.1 Related Work

In recent years, a majority of home and building automation (HBA) and smart home research concentrated on the pervasive integration of all kinds of devices found in a building. This integration is in most cases not directly possible because of the heterogeneity of different HBA standards, communication protocols and related data semantics. While current gateway-based integration solutions like OSGi [185], OPC UA [155] or oBIX [180] often provide an encoding in XML, e.g., to describe building automation devices and communication types, there also exist efforts to overcome heterogeneity in building automation systems based on ontologies [200]. In this context, DomoML [224] is one of the first ontology-based projects to propose an ontology-based household device model. While it successfully provides a taxonomy of different household appliances, DomoML fails to address the energy behavior of the facilities. The second appliance integration project is called DogOnt [34] and can be seen as an advancement of the taxonomy described in DomoML. It reuses certain parts of the DomoML ontology, tries to overcome limitations of DomoML and gives a compact and complete ontological representation of household goods including states, functionality and a basic notion of the building environment. Additionally, DogOnt starts to address the issue of heterogeneity in home and building automation as it abstracts from particular types of installed facilities and generalizes them with the help of an ontological representation. Again, energy-related issues are not specifically addressed in the suggested ontology and also the integration in a multi-agent system as well as energy efficiency considerations are not in explicit focus. Another existing project in the area of semantic representation for smart buildings is VantagePoint [177]. This project puts its focus on the visualization of information and knowledge about the system, while resting upon an ontological knowledge representation. The paper mainly concentrates on the ontology for visualization but also describes challenges arising from the integration of a vast amount of sensors available in a

smart home. For the field of wireless HBA technologies, the authors of [97] propose an ontology that can be used as middleware between wireless sensor networks and enterprise systems. In this case, the ontology reasoning mechanism is used to filter data necessary for further processing while preventing information flooding in the wireless sensor network. Further, there also exist works that propose the application of ontology-based techniques at building design time: The authors of [209] describe a requirements ontology in order to support an automatic design process of building automation systems.

While these preliminary works show plausible approaches in various sub-parts of a smart home control system, almost all of them do not address the important topic of energy efficiency. One exception is the paper of Gomez et al. [97], which however does not realize a sophisticated facility-based representation of energy information as it is needed for a smart home system. Another project that focuses on energy information is the framework presented in [99]. Although the authors' approach covers a similar topic as the one that is described in this proposed Energy & Resources ontology, they do not sufficiently discuss the context of a smart home system: while the ontology can be regarded as sophisticated, the focus of the pictured domain is unclear. Some concepts that are not necessary in the smart home domain are represented in detail (e.g., power plants' location and owner) while other more important concepts are missing (e.g., energy providers, energy tariffs). Also the integration of the ontology with the pictured central intelligence part called *Smart Home Manager* is only explained to some extent and the reader is left with open questions regarding the implementation and intelligence of the proposed system. The SESAME-S project described in [81] illustrates a system for semantic smart metering relying on ontologies. The authors identify three ontologies, namely an automation, a meter data and a pricing ontology. While for the pricing ontology important terms like energy type and providers as well as tariffs are mentioned, a detailed description of the ontological model is not provided. Also reasoning capabilities of the proposed system are only mentioned and not described in detail. Another article focusing on energy management in residential homes is [225]. Although the proposed system does not rely on multi-agents or ontologies it considers the main influence factors on building energy consumption and production in a smart home which are local energy consumers and producers as well as remote energy service providers. For the designed home energy management system, the article proposes a home network model to represent devices, networks and services. While the authors describe this high-level classification and show an example instantiation of their home network model detailed information about the used representation technology as well as detailed structure of this model is regretfully missing.

The following sections discuss the semantic representation of energy and facility information in smart homes. Different parts of this ontology have already been published in several scientific publications, most notably in [152], which also serves as foundation for subsequent elaborations. The authors of [100] further propose the usage of the energy part of the following module in a framework utilizing a variety of different ontologies to cover the smart home domain. In the present work, apart from energy consumers and producers, also the description of external energy providers and their conditions is outlined and therefore an additional viewpoint is provided that has not been thoroughly considered by any of the above mentioned projects.

4.3.2 Structural Overview

Taking a look at the top-level concepts of this resource representation (cf. Fig. 4.8), it can be seen, that the ontological descriptions can be split into two logical dimensions which are highly interrelated, the resource and the energy dimension. Both dimensions are covered with a set of subhierarchies, describing relevant concepts. For the created ontology, the energy information and resource information are represented in one ontology because of the interrelationships. For elaboration and discussion of the Energy & Resources Ontology, however, the two dimensions are being treated separately. Concepts and properties that represent the relations between the dimensions are highlighted where necessary.

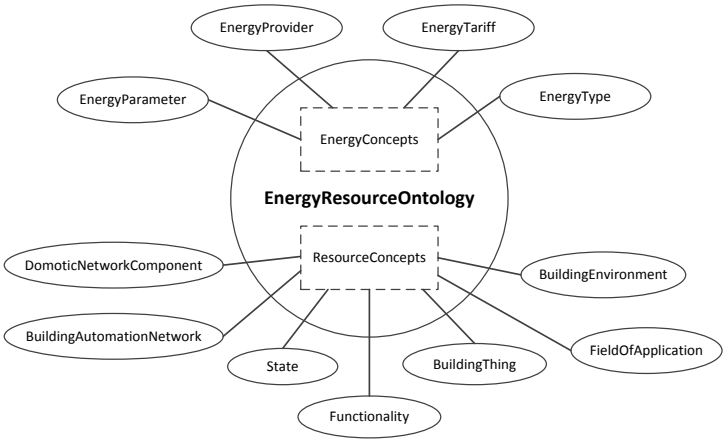

Figure 4.8: Energy & Resource ontology top-level concepts (selection)

4.3.3 Resource Description

For an energy efficient operation of a building, a wide variety of different influences have to be considered. Most importantly, for a smart home to operate on the environment, a homogeneous view on existing home automation devices and household appliances, hence *resources* needs to be provided. In the described domain resources are understood as available facilities and their characteristics. Therefore, information that is included in this particular ontology spans white goods, brown goods and building automation networks hosting lighting, shading as well as heating, ventilation and air conditioning (HVAC) devices. As the automation networks can be of different types, protocols and manufacturers, it is valuable to represent them as concepts in an ontology: this way, their definition can be generalized, which in turn supports the transparent integration of components across different networks into one central control system. In addition, also local plants and installations used for energy production like solar collectors or a thermal heat pump need to be represented in this section as well. Hence, it is desired to depict a complete model of the energy consuming and producing landscape available in the building [224].
To represent crucial information about installed equipment like possible states of operation and

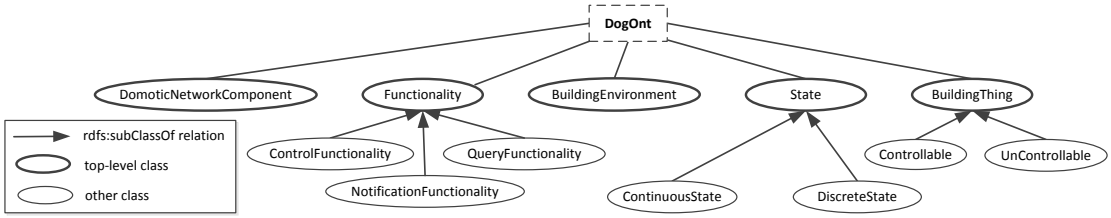

Figure 4.9: Top-level concepts of the DogOnt ontology [34]

their location in a building, the DogOnt ontology, designed for the DOG OSGi Domotic Gateway [34, 33] is found to be a suitable choice, as it thoroughly describes the controllable environment for a smart home. DogOnt is mainly used to locate devices and very simple reasoning as its main function is device modeling. Therefore, the ontology is not exhausting the capabilities of the OWL language with respect to automated inferences. Yet, because of the sophisticated modeling of household appliances as well as sensors and actuators with respect to possible states and functionalities, this ontology is reused. Currently (November 2013), the latest version of the DogOnt ontology is 1.0.8. However as a foundation for the resource part of the Energy & Resources ontology, DogOnt version 1.0.5[13] is used which was the current version at the time when integration of DogOnt into the Energy & Resources ontology was decided and initiated. Because of the changes and additions that are conducted in the integration process, changes made in newer versions of DogOnt are not considered. An overview of the main classes defined in DogOnt can be seen in Figure 4.9 and as such the most important branches representing facilities in the smart home can be identified as [34]:

- *BuildingThing*: This subbranch of the ontology hierarchy describes all controllable as well as uncontrollable things that can be found in a typical smart residential home. Large parts of the hierarchy are reused and adapted from DomoML-env [224] to fit the needs of the DogOnt field of application.

- *BuildingEnvironment*: This concept hierarchy allows a rudimentary representation of the building to represent sensors and equipment in relation with the building layout. In this ontology, defined for interoperability this representation solely serves as simple possibility to relate modeled devices to an actual location in the building.

- *State*: The state hierarchy models stable conditions that controllable devices can adopt. In this respect, it is differentiated between continuous and discrete states, identifying if a specific facility can take arbitrary values of a defined interval (continuous) or can only take on certain values from a defined set (discrete).

- *Functionality*: This hierarchy represents functionalities that can be used to model capabilities of modeled devices.

[13] http://elite.polito.it/files/releases/dog/dogont/DogOnt-1.0.5-MaisonEquipee/DOGOnt.owl

- *DomoticNetworkComponent*: Here, network components like for example routers or gateways of different manufacturers and technologies (e.g., KNX, ZigBee) can be represented.

The DogOnt ontology thus supports five different modeling dimensions: environment modeling with the Building Thing and Building Environment hierarchies, device modeling with the Controllable subbranch of the Building Thing concept, functionality modeling with the Functionality hierarchy describing device-to-device interaction, state modeling with the State subhierarchy and network modeling with the Domotic Network Component hierarchy depicting the interoperation between domotic components of different building automation networks [35]. These dimensions to a large extent represent important domains when describing facilities for an autonomous smart home system. Because of this fact, and as reuse of already existing ontologies is desired, the main hierarchies of the DogOnt ontology are taken over for the purpose of serving as resource part in the designed smart home ontology.
Considering reuse of this ontology, a direct import of this ontology through `owl:import` is not conducted. First, as some of the provided subhierarchies include information not relevant for the envisioned field of application smart home control they may be omitted. Some parts also conceptually rather belong to other defined modules (e.g., Architecture & Building Physics) and therefore also may be excluded from this part. The second main reason why a direct import is not considered is that the ontology is not normalized. As main focus in this work is put on reusability, it is desired to bring the ontology into a normalized form before reusing it.
The following list describes hierarchies and items that are left out in order to make the representation more concise:

- *Furniture*: Large parts of the subhierarchy are excluded, as a great part of uncontrollable equipment as it is described in DogOnt is not needed for the smart home control use case. Apart from the changing of thermal behavior of a room, it is not necessary to depict facilities (e.g., bidet, desk, nightstand) that cannot be controlled. Further, if the location of uncontrollable furniture is to be considered because of the thermal behavior of a room or the building, it most certainly should be described as part of the defined Architecture & Building Physics Ontology (cf. Section 4.2) as it rather belongs to static information about the building that does not change frequently. However, there are three concepts that need to be kept: `Awning`, `Shade` and `Shutter`. These devices may have associated actuators that manipulate their state and therefore need to be represented in the described smart home ontology. As the kept classes do not represent "Furniture" per se, the superclass is renamed to "Equipment" for a clearer terminology. Further, the concept `Radiator` is added to the set of uncontrollable equipment, in order to represent the equipment that is associated with a specific radiator actuator capable of manipulating its state.

- *Architectural*: Also this part is reduced to necessary concepts. The definition of `Wall` for location of a sensor, `WallOpening` for doors and windows and their associated actuators or sensors as well as the concepts `Ceiling` and `Floor` are kept, while other concepts are removed.

- *BuildingEnvironment*: This branch is kept to determine the location of sensors and facilities in the home. The associated building environment as described in the subclasses

can be related to the Architecture & Building Physics Ontology by using the properties `hasAssociatedSpace` and `hasAssociatedZone` (cf. Figure 4.2).

As stated in [198], a normalized ontological representation significantly raises reusability and is therefore considered as key requirement for large ontologies. In DogOnt often concepts are just distinguished by their name. More important than the name is the axiomatic description of a concept so that the logical reasoning mechanism has more information abou the value types represented by a certain class. Therefore, just with a richly axiomatized classification it is assured that the DL-reasoner has enough information to make correct inferences. Regarding ontology normalization, the DogOnt ontology extensively uses asserted polyhierarchies while in a normalized ontology these should be created automatically by the reasoning mechanism. However, when providing the correct set of axioms, an OWL-DL reasoner is able to maintain the polyhierarchic structure avoiding human error [71].
As a consequence, the DogOnt representation is first refactored into a semi-normalized form by reformulation of comparatively weak ontology parts, while making a tradeoff between a fully normalized ontology and practicability. First of all, naming is adapted to better reflect represented classes and individuals (cf. Table 4.6), as consistent naming is a necessary pre-condition for reusable ontologies. In this process, all classes are renamed to singular terms apart from the concepts named `BrownGoods` and `WhiteGoods` for which no singular terms exist. Another naming change is conducted for the `FreeStateValue` concept which is renamed to `NotOccupiedStateValue` to better reflect its oppositeness to `OccupiedStateValue` for the human reader. For object and datatype properties, there are no naming changes. For hierarchies, the key design focus of the reformulation further is the keeping of the original conceptualization as defined in the DogOnt ontology as far as possible but, for example, to

Table 4.6: Class names as defined in DogOnt 1.0.5 and their renamed counterparts in the Energy & Resources Ontology

DogOnt 1.0.5	**Energy & Resources Ontology**
`HousePlants`	`BuildingAutomationSystemResource`
`ElectricalSystem`	`ElectricalSystemResource`
`HVACSystem`	`HVACSystemResource`
`SecuritySystem`	`SecuritySystemResource`
`Acoustic`	`AcousticSystemResource`
`Actuator`	`ActuatorSystemResource`
`Control`	`ControlSystemResource`
`Lighting`	`LightingSystemResource`
`Sensor`	`SensorSystemResource`
`AntiIntrusionSystem`	`AntiIntrusionResource`
`FireSytem`	`FireSecuritySystem`
`FloodSystem`	`FloodSecuritySystem`
`GasSystem`	`GasSecuritySystem`

eliminate asserted polyhierarchies and only let them be automatically asserted by the reasoning mechanism. In some few cases also hierarchies are changed to better represent the dependencies between concepts: for example the subclass relation between `MainsPowerOutlet` and `OnOffOutput` is changed so that the latter is seen as specialized mains outlet that may be switched on and off. To make the reasoner-driven classification of polyhierarchies possible, it is further necessary to extend the axioms of the DogOnt ontology in a way that the reasoner can make the desired inferences. This leads to the introduction of the auxiliary concept hierarchy `FieldOfApplication`. This subbranch describes the different fields of application for installed equipment (e.g., temperature, visualization, security) and therefore helps to characterize the nature of an appliance or building automation resource. With such additional descriptions, it becomes possible to determine the capabilities of a specific building automation equipment. The multi-agent system operating on the ontology can subsequently gather possible utilizations of the appliance.
In order to automatically infer the polyhierarchies defined in the DogOnt ontology, the describing axioms of a substantial amount of concepts had to be changed. Particularly, it is necessary to define `owl:equivalentClass` properties for classes, to make the reasoning mechanism capable of automatically inferring desired dependencies.

Listing 4.5: Definition of the `TemperatureSensor` class

```
ero:TemperatureSensor rdf:type owl:Class ;
        rdfs:label "TemperatureSensor"^^xsd:string ;
        owl:equivalentClass
            [ rdf:type owl:Class ;
              owl:intersectionOf
                  ( ero:BuildingAutomationSystemResource
                    [ rdf:type owl:Restriction ;
                      owl:onProperty ero:hasFieldOfApplication ;
                      owl:someValuesFrom ero:ElectricalFieldOfApplication
                    ]
                    [ rdf:type owl:Restriction ;
                      owl:onProperty ero:hasFieldOfApplication ;
                      owl:someValuesFrom ero:TemperatureFieldOfApplication
                    ]
                    [ rdf:type owl:Restriction ;
                      owl:onProperty ero:hasFunctionality ;
                      owl:someValuesFrom ero:GroupFunctionality
                    ]
                    [ rdf:type owl:Restriction ;
                      owl:onProperty ero:hasFunctionality ;
                      owl:someValuesFrom ero:QueryFunctionality
                    ]
                    [ rdf:type owl:Restriction ;
                      owl:onProperty ero:hasFunctionality ;
                      owl:someValuesFrom ero:TemperatureMeasurementFunctionality
                    ]
                    [ rdf:type owl:Restriction ;
                      owl:onProperty ero:isSensorOf ;
                      owl:someValuesFrom [ rdf:type owl:Class ;
                                           owl:unionOf
                                               ( ero:BuildingEnvironment
                                                 ero:BuildingThing
                                               )
                                         ]
                    ]
                    [ rdf:type owl:Restriction ;
                      owl:onProperty ero:hasState ;
```

```
                    owl:onClass ero:TemperatureState ;
                    owl:qualifiedCardinality "1"^^xsd:nonNegativeInteger
                  ]
                )
            ] ;
        rdfs:comment "Sensor that detects the temperature of the atmosphere"@en .
```

The full axiomatic description of the `TemperatureSensor` class in the Energy & Resources module is exemplified in Listing 4.5. In DogOnt a temperature sensor is defined to be an item that has some temperature state, has a group and temperature measurement functionality and is a subclass of the concepts `HVACSystem` and `Sensor`. To resolve the asserted polyhierarchy, in the ontology, the sensor is defined to have a relation to two different fields of applications, `TemperatureFieldOfApplication` and `ElectricFieldOfApplication`. The first field of application makes it possible for the reasoning mechanism to infer that the temperature sensor is an `HVACSystemResource`, while the second field of application is used for classification as a resource for an electrical system (`ElectricalSystemResource`). Regarding the second inference, the reasoning mechanism can further infer that the temperature sensor is a more special version of electrical system resource, a `SensorSystemResource` also defined through an `owl:equivalentClass` axiom (Listing 4.6).

Listing 4.6: Definition of the `SensorSystemResource` class

```
ero:SensorSystemResource rdf:type owl:Class ;
        rdfs:label "SensorSystemResource"^^xsd:string ;
        owl:equivalentClass
            [ rdf:type owl:Class ;
              owl:intersectionOf
                  ( ero:BuildingAutomationSystemResource
                    [ rdf:type owl:Restriction ;
                      owl:onProperty ero:hasFieldOfApplication ;
                      owl:someValuesFrom ero:ElectricalFieldOfApplication
                    ]
                    [ rdf:type owl:Restriction ;
                      owl:onProperty ero:hasFunctionality ;
                      owl:someValuesFrom ero:QueryFunctionality
                    ]
                    [ rdf:type owl:Restriction ;
                      owl:onProperty ero:isSensorOf ;
                      owl:someValuesFrom [ rdf:type owl:Class ;
                                           owl:unionOf
                                               ( ero:BuildingEnvironment
                                                 ero:BuildingThing
                                               )
                                         ]
                    ]
                  )
            ] ;
  rdfs:comment "Class for grouping all electrically powered sensors"@en .
```

As this example shows, normalization on the one hand leads to a better axiomatization of the modeled concepts, which in turn can be seen as a more profound description of the domain. On the other hand, the reasoning mechanism can be used to automatically infer certain dependencies, taking away complexity from the manual description of the ontology.
This way, all of the originally defined polyhierarchies of the DogOnt ontology can be automatically inferred by the reasoning mechanism. A different hierarchical representation can only be found in the security hierarchy which is slightly modified. In that case, the original DogOnt

ontology defines several security system concepts (e.g., fire, flood, smoke), while associated sensors are modeled as subclass of these concepts. Generally speaking, a sensor is not a special kind of security system and therefore it should not be modeled as subclass but merely as part-of relation. For this reason, in the Energy & Resources ontology the connection between security system and sensor is created via the `hasSensor` object property, better reflecting the connection between these two concept types.
In addition, also other recommended normalization steps are performed. One of these steps is to make primitive[14] concepts disjoint. While the adding of disjoint axioms for primitive sibling classes in the original DogOnt ontology leads to several unsatisfiable classes, the definition of defined classes for polyhierarchies makes it possible to add disjointness in the Energy & Resources ontology without further problems. As the extensive modeling of household facilities and the building automation part makes the DogOnt ontology a good candidate for integration into the proposed Energy & Resources ontology, its normalized version is used as part of the resource representation. However, the DogOnt ontology only focuses on energy consuming devices, while energy producing facilities and local power plants (i.e., *sources*) are not modeled. Therefore, the ontology needs to be extended to also represent this dimension which is modeled as `HomePlant` subhierarchy of `Controllable`. This hierarchy defines concepts for local power plants and energy producing facilities, e.g., geothermal heat pumps, solar panels, wind turbines or small local hydroelectric plants.
In DogOnt, building information is only rudimentarily treated, in order to specify the location of a sensor or facility. Information about building physics and architecture is neglected, while this information is essential if considering a comprehensive smart home knowledge base for a multi-agent system. While in the case of the designed smart home ontology, static building information is represented in an own building ontology (cf. Section 4.2), it is still important to have a basic representation of the building also in the Energy & Resources ontology: this way, the module may for example be used as stand-alone ontology without the detailed building physics knowledge represented in the Architecture & Building Physics ontology. Therefore, elements that can be used to locate sensors and associate them to parts in the building are kept. When connecting the two ontology modules, certain concepts can be defined equivalent (`owl:equivalent`) as delineated in Figure 4.2.
Referring to Figure 4.1, the normalized and extended DogOnt ontology therefore may represent the resource ontology which interfaces with the more detailed building representation branch described in Section 4.2 containing information such as used material or the orientation of the building. The resource ontology is further extended with classes and properties describing the energy dimension, adding an important viewpoint for an autonomous smart home control.

4.3.4 Energy Description

On the other hand, energy representation is an important source of information about energy demand and energy supply of the smart home. In order to represent energy consuming facil-

[14] primitive classes are classes that are described only using necessary restrictions (e.g., subclass axioms) and no sufficient restrictions (i.e., equivalence axioms)

ities (i.e., *sinks*) in the proposed energy resource ontology, the modified DogOnt ontology is chosen as a starting point. As the authors of DogOnt put the focus of their knowledge base on the semantic integration of domotic components concentrating on building automation service interoperability [34], the representation mainly focuses on modeling domestic and domotic devices. Energy related issues like the demand of mapped home automation systems and appliances are however not considered in the original approach. While the description of an Energy & Resources ontology was firstly presented in [150], the authors of DogOnt have apparently simultaneously been working on their idea of representing energy properties as extension to the original DogOnt classification model (cf. [36]). However, in contrast to the energy ontology presented in this work, their approach does not go far enough: in DogOnt the authors focus solely on energy consumption, while leaving out energy production or energy supply, which however can be seen as important aspects when considering an energy efficient home system. For the interoperability use case, the representation of energy consumption can be seen as sufficient, while for a building-wide energy efficiency optimization also energy supply has to be considered.
The top-level concepts of this second part of the Energy & Resources ontology contain the following classifications (cf. Figure 4.8):

- *EnergyProvider*: All remote energy suppliers providing some form of energy relevant for the residential home.
- *EnergyTariff*: The tariffs that are charged by an energy provider to supply a certain energy type.
- *EnergyType*: The different energy types that are available and are either supplied by energy providers, or used as source of energy to produce some secondary energy.
- *EnergyParameter*: All information needed to model energy demand and supply as well as energy costs.

One of the purposes of this module is to allow a software system to base its operational decisions on the status of the connected facilities in the smart home. A second purpose is the representation of energy providers and energy tariffs, information that enables an ecological and economical use of different energy forms such as electricity or district heating, with respect to renewability and energy costs. The definition of these main supply-side concepts in the knowledge base of the smart home therefore allows yet unconsidered improvements with respect to energy consumption. This information is especially valuable when envisioning the integration of the ThinkHome system into a smart grid, as the ontology can provide the momentarily best option for energy consumption or recovery. Considering other ontology modules like the Users & Preferences or User Behavior & Building Processes ontology that keep energy schedules for different occupancy states and scenarios (e.g., day, night, weekends, holidays) it becomes possible to anticipate consumption and thus mitigate peaks. The modeling of this information needs to be open to changes, as it is not only likely that new and probably unknown devices are added to the smart home, but also that information about energy providers and their tariff schemes might change frequently.

4.3.4.1 Types of Energy

Firstly, in order to provide the system with a general notion of energy, a categorization of energy types is needed. The knowledge base has to reflect the domain of smart homes with respect to modeled energy types that are relevant in this domain. As a first classification, a distinction between *final energy*, *energy sources* and *useful energy* is realized (Figure 4.10). First of all these classes reflect the varied usage of energy types viewed from the providing side as well as from the consuming side. Mainly what is relevant for a smart home are two viewpoints: the production and the consumption viewpoint. For the *production viewpoint* which focuses on energy providers but also local energy producers, it is necessary to be able to represent different energy sources that can be used to produce secondary energy or can be directly delivered to a household. The concept `EnergySource` therefore is used on the energy provider side to classify different energy providers and describe the used energy mix or provided primary energy. Energy, as it is received in the building is entitled `FinalEnergy`. Hence, final energy is energy as it can be consumed in the home by power facilities. Further, there exist facilities that can convert final energy into `UsefulEnergy`, which can be defined as *"final energy use minus assumed conversion losses at the home"* [112]. For example, `Gas` as individual member of `FinalEnergy` can be transformed to `Heat` a member of the `UsefulEnergy` concept by a gas heater. With other words, useful energy describes energy types that are inconvertible with respect to usage in a residential home. All three forms of energy together give a comprehensive picture of energy consumption in residential homes, for example allowing a clearer picture of substitution of electricity for heating fuels [112].
The classification of energy sources follows the general definition, whereas two distinctions are made. The first is into *primary* and *secondary sources* of energy. Electricity, for example, is a secondary energy source (`ElectricEnergySource`) because it is not freely available in nature and therefore has to be gained through some primary energy source. Individuals classified to be members of the `ElectricEnergySource` class are used in order to distinguish between different energy tariffs of one single energy provider (cf. Section 4.3.4.2).
The second differentiation in the energy source hierarchy into *renewable* and *non-renewable sources* is particularly necessary for resource-optimal energy consumption in a smart home. It is important for an environmentally-friendly building to consider which type of fuel is used for energy production as well as what method of energy conversion is applied and further, how much the environment is polluted because of a specific energy generation process, transmission or end-use [51]. Therefore, it is inherently more environmentally friendly to rely on renewable energy sources. These two last mentioned concepts are finally super-concepts of the actual energy sources.
The concepts to represent final and useful energy have a closed set of member individuals for energy types that can be consumed in the smart home. A differentiation is made with respect to their transformability in the home: For example, `Heat` or `Cold` as members of `UsefulEnergy` can be consumed in the residential home, but are non-transformable, i.e., they cannot be used to generate another type of energy that is of interest for the automatic control of a smart home. On the other hand, `Electricity`, `Gas` or `Wood` are examples of final energy that still needs to be transformed into useful energy by a transformation facility.
The concrete members of the concepts `FinalEnergy` and `UsefulEnergy` that are defined

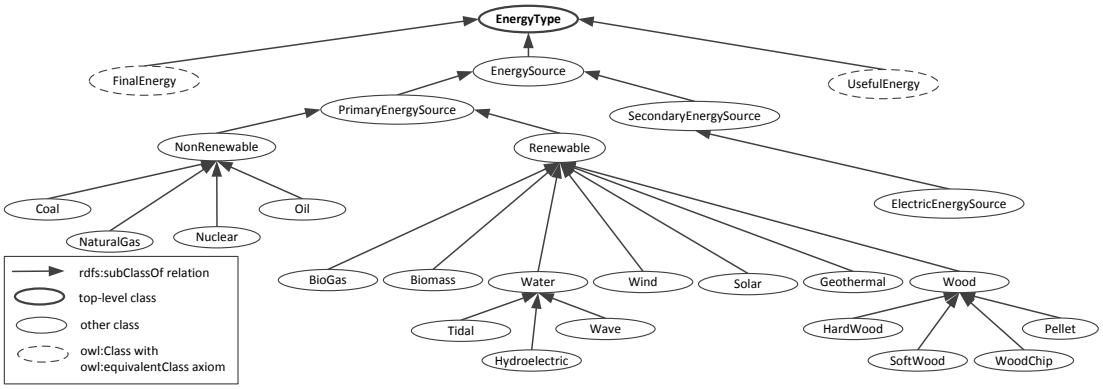

Figure 4.10: Energy forms hierarchy of the Energy & Resource ontology. Shown subconcepts of the `NonRenewable` and `Renewable` classes have alias names to highlight different energy forms.

in the Energy & Resources ontology can be seen in Table 4.7. These groups can always be enlarged or narrowed, according to which end energies should be covered by the smart home system. Because of the two viewpoints on energy, once external for the energy production side and also internal for the energy consumption side, some energy types occur twice in the knowledge base: first as energy sources and additionally also as members of one of the two concepts for final and useful energy. The importance of energy for a self-controlled home needs a comprehensive representation of the term and therefore the two mentioned views need to be covered. The production and the demand side's energy conception need to be represented differently as energy sources exist that may be used by an energy provider to generate secondary energy but can also be directly provided for consumption in the smart home as final energy. Natural gas, for example, may be part of the energy mix of an electricity provider used for production of electric energy in a gas-fired power plant. This can be seen as the energy supply view. At the same time gas can be provided to the household for consumption and therefore it is additionally a member of the `FinalEnergy` concept. This can further be identified as energy consumption view. The benefits of the realization of final energy and useful energy types as OWL individuals instead of concepts are further discussed in Section 4.3.4.3.

Table 4.7: Members of `FinalEnergy` and `UsefulEnergy` concepts

FinalEnergy	UsefulEnergy
Coal	Heat
ElectricEnergy	Cold
Gas	Light
Wood	Water
Oil	

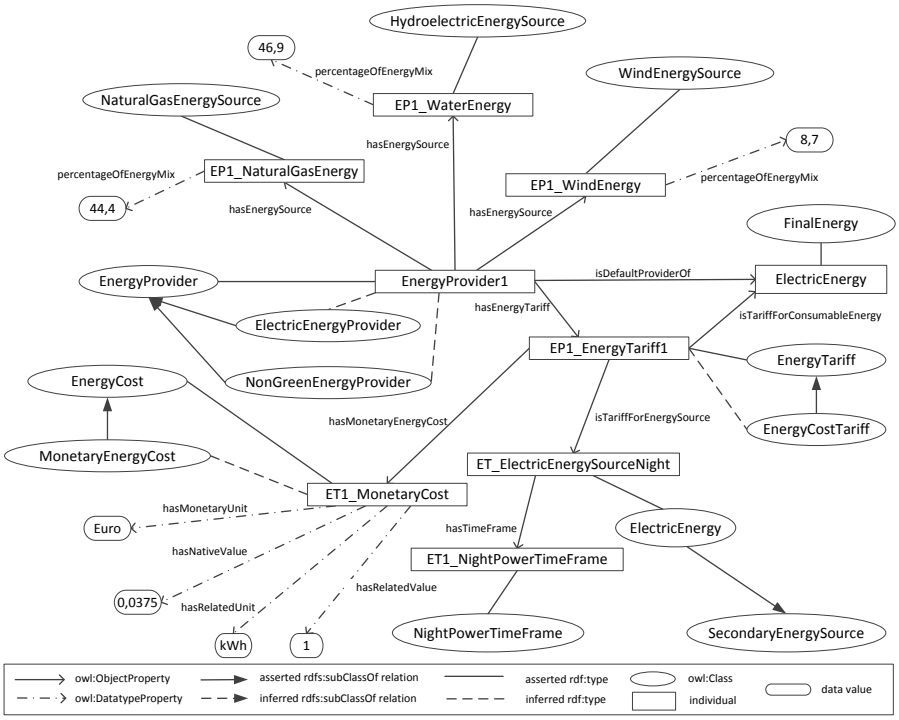

Figure 4.11: Energy provider instance

4.3.4.2 Remote Energy Providers and Tariffs

At the dawn of the smart grid and the vison of decentralized energy production, micro-energy providers and shiftable loads, knowledge about different energy providers and their tariffs are a valuable addition to a smart home information representation. According to [190] *"one of the key aspects to a smarter grid is the ability to make decisions on how to operate the power system on both, the supply-side and the demand-side"*. Therefore, an autonomous smart home optimization taking into account energy providers and their tariffs and acting according to peak load periods is desired.

It is assumed that like in the electric energy market also other markets (e.g., gas, water) will be liberalized in the future and therefore a variety of energy suppliers is classified in the proposed knowledge base. This conceptualization comprises deliverers of different energy forms as well as the distinction between *green* and *non-green energy providers*. Reasoning on this hierarchy with a DL-reasoner may again be used to automatically generate inferred hierarchies and classify individual members of different energy provider classes as green and non-green types. The following example explains how inferences are conducted.

In Figure 4.11, the instance-centric view of an example energy provider (`EnergyProvider1`)

generating electricity through hydropower, wind and natural gas is modeled. In the Energy & Resources ontology, the connection of an energy provider to its primary energy forms, thus, its energy mix, is depicted via the object property `hasEnergySource`. Because of the fact, that one of the three energy sources in the energy mix is a non-renewable energy source (cf. Figure 4.10), the logical reasoner can infer that the energy provider is of type `NonGreenEnergyProvider`. If at a future point in time the energy provider changes its energy mix and stops using natural gas as energy source, the classification can be automatically corrected by the DL reasoning mechanism and the provider further listed as green energy provider. Further, the datatype property `percentageOfEnergyMix` can be used to describe the share of a specific energy source in the energy mix of the modeled energy provider.
The energy that this energy provider supplies is electric energy. This can be directly modeled as `providesEnergyType` relation, however may also be inferred by the logical reasoner through a property chain [116], a functionality of the OWL 2 language. Property chains allow the connection of object properties in a way that they can be serially related to each other. In the case of energy providers, the property chain for the `providesEnergyType` relation has the following representation[15]:

```
ObjectProperty: providesEnergyType
  SubPropertyChain: hasEnergyTariff o isTariffForConsumableEnergy
```

With this property chain and the inference that can be made through the reasoner, it is assured that only energy is supplied that is also available in one of the energy provider's tariffs[16]. The `isDefaultProviderOf` object property finally allows to determine if an energy provider is the default provider for the particular energy type it provides through one of its tariffs.
In addition to the way how suppliers generate energy, it is needful to know the different types of energy tariffs. Of course not every company has the same rates for energy supply and also different tariff switching times can exist which have to be considered in the knowledge base for reasoning on economic deliberations. In Figure 4.11, the nighttime electricity tariff of the represented energy provider is shown exemplarily. Energy tariffs are connected via the `hasEnergyTariff` object property. Tariffs have a relation to their monetary energy cost, which itself is realized as individual, representing how much is charged by the energy provider for the supply of a specific unit of energy. Further, it can be seen that the energy tariff individual links energy sources with consumable energy types. With the OWL object property `isTariffForConsumableEnergy`, a relation to an consumable energy type individual is realized, while the property `isTariffForEnergySource` allows the tariff to be related to the concrete energy source that is supplied by the energy provider. With this representation, for example, different tariff intervals can be taken into account while it still remains possible to refer to the energy type of the tariff as one single individual. With the reuse of the OWL-Time ontology [186], the characterization of tariffs according to their active times becomes possible. Energy tariffs in the ontology may be classified as `EnergyCostTariff` or `EnergyRecoveryTariff`, subclasses of the `EnergyTariff` concept: cost tariffs are tariffs that the energy provider charges for supplied energy. Recovery tariffs are refunded by an energy provider for recovery of electric energy into the grid through local energy producing

[15] Listing in Manchester Syntax for OWL 2 [259]
[16] Figure 4.11 only shows asserted property relations.

facilities (cf. Section 4.3.4.3). Because of its monetary energy cost relation, the reasoning mechanism can identify the example energy tariff as `EnergyCostTariff`.
Together, the concepts `EnergyProvider` and `EnergyTariff` form the main concepts of the supply side, modeling external energy supply for the smart home.

4.3.4.3 Local Energy Production and Consumption

Apart from remote energy providers, it is undoubtedly also necessary to model local energy production and consumption in the Energy & Resources ontology. In a modern home, local renewable energy producing facilities can help to reduce dependency on exterior energy that needs to be purchased from energy providers. Further, a clear picture of energy consumed by facilities in different parts of the home in combination with occupancy schedules facilitates an optimized operation with regard to energy peak loads.
Again, an example is used to explain the representation of the described facts in the proposed smart home knowledge base (Figure 4.12). The specific example appliance is a DVD recorder

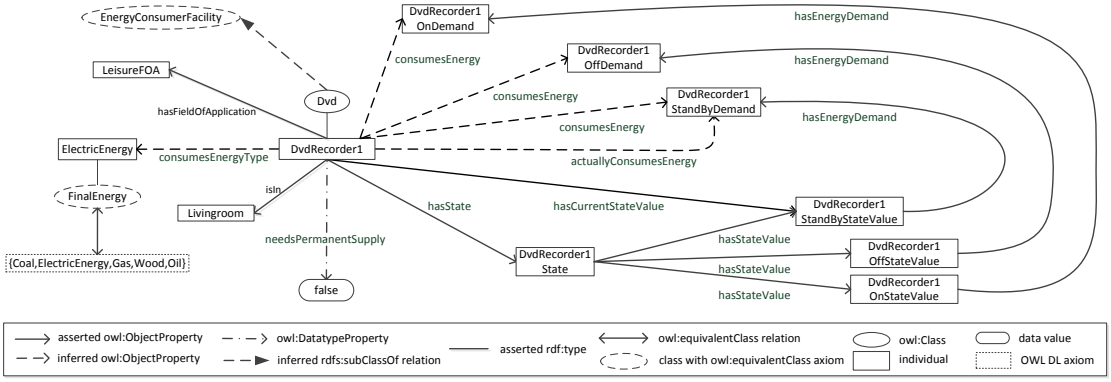

Figure 4.12: Energy facility with current energy demand

located in the living room of the building and modeled as a member of the `Dvd` concept. For brevity of the example only relations and concept memberships needed for the explanation of the energy extension are shown.
To classify energy facilities in the building domain, three additional concepts are added to the `BuildingThing` hierarchy: `EnergyFacility`, `EnergyProducerFacility` and `EnergyConsumerFacility`. These concepts are realized as defined classes enabling the reasoner to automatically infer subhierarchies. With respect to the smart home domain, energy facilities can be defined as all appliances which either consume or prodce energy and are controllable. The installed equipment can range from a washing machine or dishwasher to a solar panel or wind turbine on the roof. The two other defined classes are subclasses of the energy facility class and split represented facilities into two groups, consumers and producers. A facility in this case does not have to fit only one type: for example a geothermal heat pump at the

same time produces thermal energy while it consumes electric energy. Such energy facilities can accordingly be reasoned into both sub-groups.
The distinction between energy consumers and producers is realized with the help of equivalence axioms (cf. Figure 4.13): energy consumers are defined to have a `consumesEnergy` object property relation to some energy demand individual, as well as a value that determines if the represented facility needs permanent power supply, realized as `needsPermanentSupply` datatype property. This property may for example be used do determine if a facility can be disconnected from power in order to save energy. Energy producers are classified with the help of two object properties: first, they are defined to produce some energy (`producesEnergy`) and second they have an energy source (`hasEnergySource`) that is used to generate provided energy (e.g., geothermal energy source, solar energy source). The original definitions of the `Dvd` concept and other facilities retrieved from DogOnt are extended with properties in order to classify as subclass of either of these concepts.
For a system autonomously acting on behalf of the resident, firstly, the energy consumption per state of operation is of special interest. This information can be retrieved through smart plugs as described in [117] providing detailed information about the attached facility. If this possibility is not available, the information may be manually retrieved either through measuring or from the technical specification of a facility. For representing the dependency in the Energy & Resources ontology, the object property `hasEnergyDemand` is created, that associates each state value of a facility with a specific energy demand. Considering facilities with continuous states and state values, there exist two possibilities: on the one hand, the continuous state may be discretized into a finite set of discrete state values (25 percent, 50 percent, etc.) and for each of the state values an approximate energy demand is stored. On the other hand, the energy demand is measured and updated according to the current continuous state value. In the second case, only one (current) energy demand individual exists.
As an extension, subproperties are defined for the object properties relating the facilities to their supply or demand. In Figure 4.14 the subpropery hierarchy is shown for the `consumesEnergy` object property. Two subproperties are created to differentiate between current energy demand and maximum energy demand of a specific facility. An equal hierarchy is defined for the `producesEnergy` property. While the maximum energy demand of a facility needs to be manually asserted, the current energy demand can be inferred with the shown property chain. For the energy production side, this property chain does not exist, as the energy production of a specific facility is dependent on several influencing factors (e.g., insolation, wind) and therefore the real energy supply needs to be measured just in time to have an accurate vision. The second property chain shown in the figure can be used to infer the `consumesEnergy` relation from single energy demands of the different facility states. This chain similarly exists for the `producesEnergy` property.
Considering the DVD recorder example illustrated in Figure 4.12, the `DvdRecorder1State` individual related to the DVD recorder by the `hasState` relation, holds information about the states that can be entered by this facility. The different modes of operation are represented as state values connected via the `hasStateValue` object property. The example shows that the possible values of the recorder are on, off and stand-by. Further, the state values have `hasEnergyDemand` relations to individuals representing the concrete energy demand in a

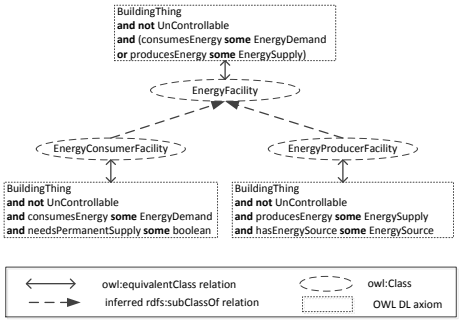

Figure 4.13: Energy facility hierarchy with `owl:equivalentClass` axioms

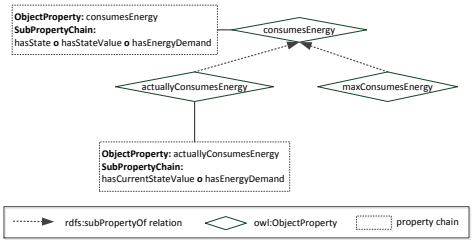

Figure 4.14: `consumesEnergy` object property hierarchy with property chains

specific operative state. The OWL object property `hasCurrentStateValue` further models the current state of the facility and needs to be changed by the system according to state switches: In the pictured case, the DVD recorder currently resides in *stand-by* mode. For the facility there is no information manually asserted about how much energy it consumes. This information can rather be inferred by the logical reasoning mechanism. The `consumesEnergy` relation that connects the DVD recorder individual to the demand representations is retrieved through the already described property chain illustrated in Figure 4.14. At the same time, also the `actuallyConsumesEnergy` relation which is a subproperty of `consumesEnergy` can be automatically inferred by the specific property chain as well.
Yet other facilities may take different state values but have an energy demand that is not dependent on the current state of the facility. An example would be a window actuator: this actuator can take the state values ”open“ and ”close“ but the energy demand is the same regardless of the current state of the actuator. In this case, both state values may refer to the same energy demand individual representing the permanent demand of the particular device.

4.3.4.4 Connecting Energy Demand Side and Energy Supply Side

The connection between energy demand and energy supply side is realized via an ontology design pattern called *class-individual mirror* and described in [4]. The example in Figure 4.15 shows the application of the pattern for energy providers, producer and facilities in the ontology. In the graph, two energy providers and two facilities are illustrated. The energy providers and the solar panel facility together represent the energy supply side while a washing machine is illustrated at the energy demand side. As already described before, the facilities can be automatically inferred to be either consumer or producer facilities. Also for the energy provider hierarchy equivalence axioms are created in order to exploit automated reasoning. Figure 4.15 exemplarily shows the axiom of the `ElectricEnergyProvider` concept. This axiom assures that only energy providers that provide energy of type ”electric energy“ are classified as electric energy providers. On the energy demand side, a washing machine is modeled as specific energy consuming facility. The graph demonstrates how the final energy types explained before

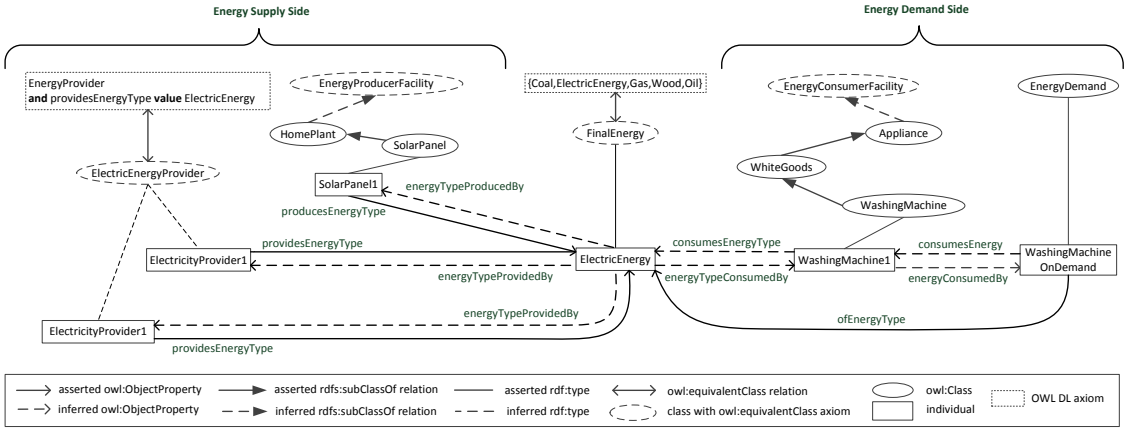

Figure 4.15: Connection between energy providers and energy facilities

may act as pivotal elements between the two conceptual dimensions: It can be seen that `ElectricEnergy` as individual member of the `FinalEnergy` concept holds information about the energy supply and the demand side and therefore connects the two different viewpoints in the ontology. In the example, the electric energy providers and the electric energy producing facility are connected to the `ElectricEnergy` element through the `providesEnergyType` and `producesEnergyType` relations. While in the example case these properties are manually asserted, they may again be inferred by the reasoning mechanism through appropriate property chains (cf. Listing 2.2). Additional to these connections, there exist inverse counterparts in the ontology (`energyTypeProvidedBy`, `energyTypeProducedBy`) that can be inferred by the reasoning mechanism through the definition of `owl:inverseOf` statements in the knowledge base. A similar situation is found on the energy demand side, where the energy consumer (i.e., washing machine) is modeled. An energy consumer, as already mentioned, has usually some energy demand, and therefore also has a type of energy it consumes. This type of energy can indirectly be inferred from the `ofEnergyType` object property defined for energy demands. Through this relation it is defined that the washing machine consumes electric energy, and thus the `consumesEnergyType` property can be inferred pointing to the `ElectricEnergy` individual as well. The benefit of using the class-individual mirror pattern in this case is that the pivotal element holds information from the energy supply side and the demand side. If, for example, energy providers for a specific facility have to be retrieved, this can happen through solely querying the `ElectricEnergy` element, as it already holds information about which energy providers or producing facilities supply the demanded form of energy.

As these elaborations show, energy representation, as module of the proposed smart home knowledge base, keeps a wide variety of useful information for the energy efficient operation of a building. All of this information can further be utilized to realize sophisticated use cases and scenarios of which some are discussed in Chapter 5.

4.3.5 Ontology Metrics

Table 4.8 shows relevant metrics of the Energy & Resources ontology in comparison with the original DogOnt ontology. While some concepts of DogOnt were removed, roughly 124 new concepts are defined. Additionally, the amount of object properties and datatype properties has increased. The major part of the additional concepts comprises classifications related to energy consumption and provision like energy types, energy providers, energy tariffs and energy facilities. A vast amount of the classes inherited from DogOnt is concerned with representing commands, functionalities, states and state values as well as a hierarchy including all types of elements and things in the building.

Table 4.8: Metrics of the DogOnt ontology version 1.0.5 compared to the Energy & Resources ontology module

		DogOnt 1.0.5	**Energy & Resource Ontology**
TBox	Named Classes	400	524
	Object Properties	25	72
	Datatype Properties	30	40
ABox	Instances	0	16

As the Energy & Resources ontology also makes use of individuals to describe energy types and building automation networks the final version also includes several instances. To describe activity intervals for energy tariffs, the import of the OWL-Time ontology should be considered.

4.4 User Behavior & Building Processes

The process ontology which is the topic of this section does on the one hand hold information about processes that can be started by a smart home system and further, also represents profiles and patterns. These concepts can in further consequence be used to depict contexts in the home and to anticipate situations on behalf of the user. To achieve the two main goals of energy efficiency and user comfort, user behavior and building context play major roles. In this section, it is proposed to recognize context, and especially user behavior and habits through a profile-based control. In this respect, *user profiles* can be used to record user behaviors and deduce habits from recurring observations which in turn are represented as *user patterns*. To detect representative profiles and patterns, different mechanisms may be used as described in [132]: firstly, user profiles may be generated from single data values. These profiles then represent the user behavior over a specific time period. Following, a certain quantity of profiles is used to create patterns representing user habits with the help of a pattern generator algorithm. These patterns are furthermore used in the smart home system to predict future user behavior and thus identify optimization potentials that arise accordingly. Besides reducing energy consumption, the proposed process provides the possibility to infer an individual notion of comfort from usage profiles.
Certainly, the smart home system additionally needs to know the set of processes that can be

started for each zone (e.g., turning on electric heating, lowering blinds, opening windows), in order to be able to choose the best alternative to reach a desired comfortable condition. Therefore, to cover the user and building context in a smart home system, *process knowledge* is another dimension of context knowledge and plays a major role in recognizing which actions are available and feasible at a specific point in time in a particular zone of the building. Also services that can be optimized with the help of user profiles, as for example reaching the comfort setpoint temperature need to be described in the knowledge store while their dependency to processes has to be modeled as well.
A representation of this knowledge in an ontology can significantly support an autonomous system to establish comfortable conditions in the most energy efficient manner. Figure 4.17 shows an overview of the top-level concepts and their relations as they are defined in the created process ontology. The following sections explain the general structure of profiles and show how properties and dependencies are reflected. This part of the knowledge base was initially described and published in the article [146] which also serves as basis for the following sections.

4.4.1 Related Work

In a smart home, to achieve optimal, satisfactory performances directly depends on the level of context awareness implemented in the underlying control system - a review of context awareness in AAL (Ambient Assisted Living) is given in [42]. The more aware the system is concerning what happens within the dwelling or facility, the more chances it has to carry out suitable operations. The "suitability" of actions in this case has to be deliberated in terms of energy minimization, comfort maximization, and user adaptiveness. In this respect, one of the key aspects to face is how to effectively represent and store the surrounding context. The multiple, coexisting applications manage changing contexts created by interleaving human activities and implemented services. In such a complex scenario, the approach of Sato proves to be especially appropriate. The author proposes that the context should be represented through *"a pattern of behaviour or relations among variables that are outside of the subjects of design manipulation and potentially affect user behaviour and system performance"* [212]. Therefore, behavioral profiles and habit patterns seem to be the correct objects for context abstraction. The list of works that follow this approach is extensive and some examples shall be mentioned. A first introduction of a profile management framework together with an overview of profile technologies for smart homes is shown in [101]. Different kinds of profiles (e.g., presence, temperature, light) are proposed for a wireless sensor structure in [17]. Profiles are used to explain inhabitants' behavior, thus energy consumption and comfort requirements can be optimized based on predictions. In this respect, Mori and Takada have published a considerable amount of works (e.g., [167, 235, 166]) in which profiles are deployed in a similar way, capturing behaviors, abstracting home activities (e.g., eating, sleeping), detecting accidents and providing users with a dedicated environment. In this context, also the CASAS project deserves attention: the authors of [195] propose a set of components to discover patterns in user activities that in further consequence can be used to schedule automated system actions. Likewise, to support intelligent home network services, a user pattern analysis framework is selected and depicted in [143].
Further, in the field of home automation, the work of Mozer is widely known as one of the

early attempts of adaptive environments [171]. In his neural network house, diverse types of profiles are deployed to control temperature, light, ventilation and water heating. In the SOPRANO project an open, flexible and semantic service platform for AAL is proposed. Context is represented with a dedicated context ontology that in the SOPRANO case acquires the current situation of a user with the help of interconnected sensors [145]. The main focus of this project lies on the detection of the situational context of a person in the supervised area. Also the approach found in [50] proposes the use of ontologies and reasoning for activity recognition in AAL applications. In this case, a Description Logic-based reasoning mechanism is used to recognize different user activities by binding sensor observations with context models and creating ontologies that describe activities of daily living. The main focus of this project lies on recognizing activities and patterns for use in an agent-based system in the AAL domain, however, an application with respect to energy optimization is not planned.
The conception of behavioral context is a precarious aspect, since a wrong approach ultimately leads to difficult control scenarios and unsatisfactory results. Conflicts, human interleaving and service overlaps must be carefully taken into account already at design time of the whole smart home concept. For instance, for the definition of profiles there are usually different possible views of observing the interaction between users and the diverse elements and domestic phenomena. On the one hand, profiles that store specific user preferences may be very suitable for certain AAL applications, however usually yield to complex scenarios. On the other hand, profiles addressing devices or rooms simplify context recognition for the system. For example, in order to control the heating of the living-room, it is easier for the system to directly work with the comfort temperature profile of the room than managing the diverse comfort temperature profiles of each inhabitant. In the present work, context representation through room-based profiles is introduced. At the same time, user preferences are treated in an own ontology module (cf. Section 4.5) which in this case may be seen as auxiliary information to take personal preferences of occupants into account and allow the optimization of system strategies even if habit profile information is not available.
In any case, although the success of system performance mainly relies on a good application design, some conflicts are unavoidable and accordingly have to be expected and solved. In this respect, in [154], conflicts in multi-agent systems are treated smartly by the prior establishment of the context applying user profiles, activity profiles and object profiles. In contrast, approaches like the one introduced in [110] propose complex action-related profiles and structures in which conflicts do not seem to be properly covered.

4.4.2 Profile-based Smart Home Control

In a smart home, implemented services, e.g., to reach thermal comfort of a room, are usually realized by control algorithms embedded in dedicated controllers (i.e., agents). In profile-based applications, profiles and associated patterns based on profiles are representing context data to be interpreted and further used by controllers to adjust the environment according to user behavior. Each controller, and as such, agent of a multi-agent-controlled system utilizes the required patterns in algorithms and rules specifically defined for the intended application (cf. Chapter 5). Although the application of habit information is ultimately specific for each service, the process from sensor data collection to the context reading follows a standard scheme irrespec-

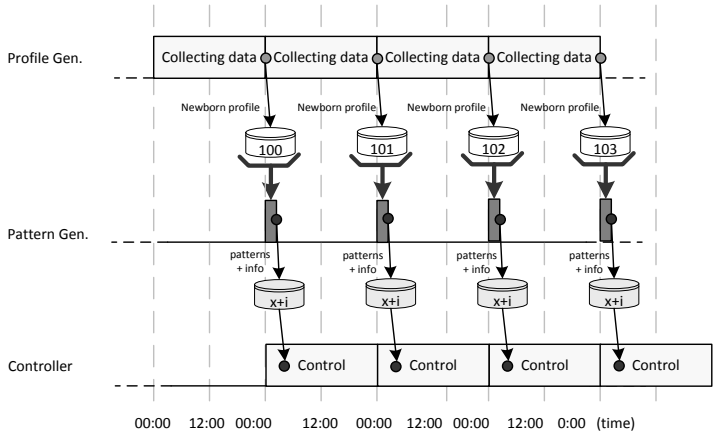

Figure 4.16: Profile and pattern processes related to time (generation, storage, analysis and deployment) (cf. [129])

tive of which kind of profile or pattern is referred. This process is described in detail in [129] and sketched in Figure 4.16: data is collected from sensors and initially processed by a *profile generator* (PFG). This PFG consists of a buffer that stores, and in case of need, averages values from the incoming information and saves them in a persistence store (i.e., an ontology) in form of a profile, i.e., an array representing a time series. This task is repeated regularly with a rate fixed by the profile scope, always adding "newborn profiles" to the store.
A second element entitled *pattern generator* (PTG) from time to time retrieves a set of stored profiles from the ontology and tries to identify trends within the set. Components like the PTG that perform a habit discovery process are commonly based on statistical, data mining or clustering-based techniques [130]. A thorough discussion of different clustering techniques for pattern generation can again be found in [129]. The outcome of this process is from one up to several patterns representative for recurring habits and user actions detected in the observed set of profiles. Once patterns are identified, they are again in the ontology ("x" in Figure 4.16). In addition, analytical information of each discovered pattern is provided by cluster validity methods. This extra information is essential for context reading and interpretation as it illustrates the representativeness and reliability of each found pattern. The information needs to be represented in the ontology as well (indicated as "i" in Figure 4.16). In further consequence, controllers and associated context agents may retrieve represented patterns from the knowledge base for specific applications and subsequent decision making.
In order to reach optimized energy efficiency and user comfort, profile-based control can easily be integrated into classic HBA installations. There are no special requirements or constraints to consider, but some recommendations of good practice. As already explained previously (cf. Chapter 1), building automation, networks and operation are classically divided into three levels, the field, the automation and the management level. Because communication in field and automation networks is often realized with constrained protocols, the focus on these layers rather lies on efficient command transfer than on excessive data transmission. Profiles and patterns

however represent a considerable amount of traffic as profile descriptions have to be sent back and forth between PFGs and PTGs as well as knowledge base and as such it is preferable to place relevant components, i.e., PFG, PTG, ontologies and profile-based controllers on the management level on which data is mostly transferred over Ethernet.

Comfort and Energy Optimization through Profile-based Control

One of the main challenges of smart home systems is to provide desired comfort situations while simultaneously achieving an energy cost minimization. Profile-based control in this case can become a tool that fulfills this requirement by implementation of the following four capabilities [129]:

- **Adaptiveness:** Each user of the system represents different tastes concerning how home applications and devices are parametrized and how the environment should be conditioned. Apart from the focus on specific applications, these requirements also include global management and system-user interactions. For instance, on the one hand profiles store habitual comfort temperatures and lighting levels that support the manual adaptation performed by users. On the other hand, they may also be used to detect trends that measure the level of user dissatisfaction or disagreement regarding the implemented services [131]. This means, with the proper set of profiles, the system is able to check its own performance, as well as to learn how users usually solve situations or adjust services.
- **Predictive Control:** Specific applications profit from forecasting capabilities, in order to prepare home devices and appliances for an upcoming scenario. Since patterns store habitual behaviors and usages, predictions can easily be obtained. Some common services that may benefit from habit-based predictive control are lighting, HVAC and DSM.
- **Global Reasoning:** Some profiles are common in diverse applications, becoming shared objects among them. A simple example is useful to illustrate the purpose of profiles in this context: a user may usually leave the house at Friday, 9:00am, to go for a weeklong vacation and when doing this, always first waters the plants and takes a shower which leads to an unusual increase in water consumption. In this case, the combined reading of *occupancy* and *water usage* patterns predicts that the house will be unoccupied longer than two days. Thus, the system may automatically reduce heating to safety rates, switch off non priority electrical sockets, unroll blinds and schedule the water heater to only be active on Saturday night, when power tariff is reduced avoiding unnecessary tank heating.
- **Tailored Feedback:** Habit-based feedback is mandatory for optimizing energy use [85] and improving the system's adaptiveness [131]. Based on habit patterns, the user is informed and warned just in time about possibly wrong habits, resource misuse, resource malfunction or similar energy-related problems.

4.4.3 Definition of Processes and Applications

As already mentioned in the beginning of this chapter, it is imperative to model actions and behaviors in order to represent the context in a smart home. In the ontology module created for

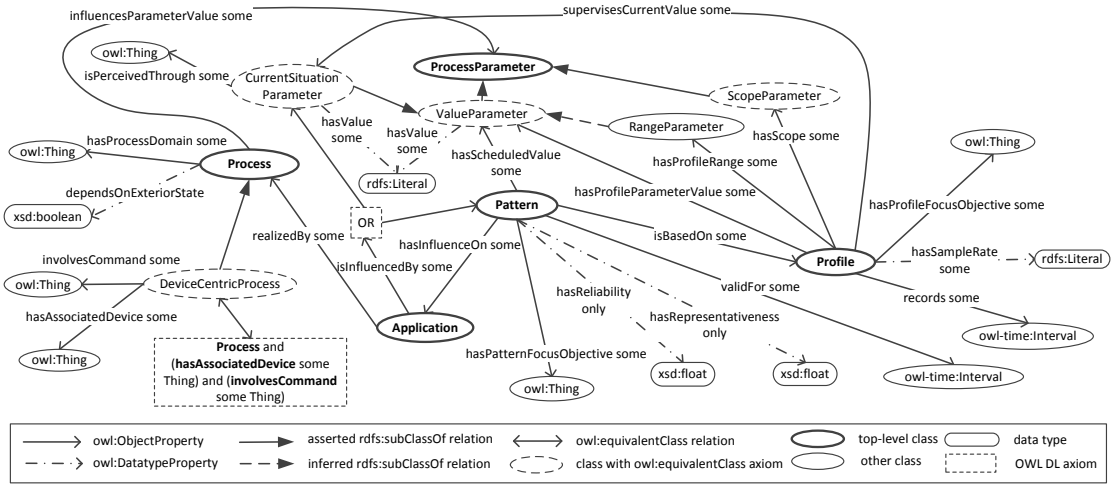

Figure 4.17: Top-level concepts and relations of the behavior and process profiles ontology

this purpose, a *process* is an atomic action that can be carried out by the system autonomously (cf. Figure 4.17). An individual process may be to turn on the heating or open the window. Such processes can in further consequence be used by the multi-agent control system to infer which actions are basically available to change the environmental situation in the building. A specific process instance is mostly associated with a device that can be manipulated by this process (e.g., window actuator, radiator actuator), and which is represented in the ontology with the `hasAssociatedDevice` object property. A process may further trigger a certain command which is defined for the automated equipment (e.g., open command, turn up command) depicted by the `involvesCommand` property in the ontology. By executing the associated command in the home automation system (cf. Section 4.3), a specific environmental parameter can be influenced (`influencesParameterValue`). Further, the ontology provides the possibility to depict how environmental parameters are influenced by individual processes through the subproperties `increasesParameterValue` and `decreasesParameterValue` of the `influencesParameterValue` object property.

Environmental parameters are represented in the `CurrentSituationParameter` class. This concept contains the parameters that are sensed through a particular sensor and describe the current environmental situation. It can be inferred to be a subclass of the `ValueParameter` class as in the case the ontology is used as stand-alone version, its members usually have an associated value connected through the `hasValue` datatype property . In the present case however, the ontology is used as module of a comprehensive smart home ontology and therefore, the current value of an environmental parameter is stored in a dedicated module together with the corresponding sensor that senses it (cf. Section 4.3, Chapter 5). Hereby, only the link to the Energy & Resource ontology needs to be created in this module through the `isPerceivedThrough` object property.

If a process is influenced by the exterior situation (e.g., opening the window for natural cooling)

can be represented through the datatype property `dependsOnExteriorState`. The decision if an exterior situation is suitable for a particular process is then made in the control system operating on the ontology (cf. Chapter 6). Finally, a process has a domain which is realized via the `hasProcessDomain` relation and which represents the space or zone of the building that is influenced by the execution of a specific process. An example of how processes are used in the operation of the smart home system is given in Section 6 in which the usefulness of process descriptions with regard to an autonomous smart home operation is outlined.
In contrast to processes, *applications* as represented in the ontology module are high-level goals that can be seen as objects of global reasoning in a multi-agent-controlled smart home system. Applications therefore have a broader context: while processes picture actions that can be started by the autonomous system, applications describe general goals that can be achieved by executing certain actions in a specific area of the building. As such, applications may be seen as processes described in a broader context. In this regard, an application relates to processes via the `realizedBy` object property which in turn may be used to achieve a goal. Therefore, processes can be seen as elementary building blocks for applications. The ontology thereby describes all possible processes that can be used to achieve a goal. One of these processes can subsequently be chosen by the multi-agent system to achieve the described goal. The decision which processes are feasible and optimal to reach a certain state however lies in the responsibility of the multi-agent system. One example of an application and thus agent goal is "to reach a desired setpoint temperature in the living room": while it is just a broader definition of a process, several atomic processes defined in the ontology can be used to realize this goal. The application may for example be influenced by a heating process (e.g., turn heating on), a cooling process (e.g., turn air condition on) or a (possibly natural) ventilation process (e.g., open window). In this case, it is the responsibility of the multi-agent system to decide which of the processes is feasible considering the circumstances (e.g., weather conditions) and appropriate with regard to energy optimization to establish the desired condition most energy-efficiently at a certain point in time (cf. Chapter 6).
Further, an application may be influenced by (`isInfluencedBy`) a specific pattern and a current situation value, (e.g., instantaneous occupancy) modeled as process parameter. If an application is for example influenced by a setpoint pattern, the system strives to reach the desired setpoint that is depicted in the pattern and derived from setpoint level profiles. If the pattern describes space usage like patterns belonging to the `OccupancyPattern` class, this information can be taken to influence the necessity to reach a setpoint. For example, it is possible to model the fact that the comfort temperature only needs to be reached when occupancy is immediately expected. In unoccupied periods, the system has a more relaxed schedule and may freely optimize heating energy consumption while still satisfying certain limits to prevent building damage (e.g., mold).

Table 4.9: Applications according to [129] and their relations to patterns in the User Behavior & Building Processes ontology

Application	`isInfluencedBy` **Pattern**
GeneralCommandForLightingGroupsApplication	LightLevelPattern LightStatusPattern
SelfCheckingLightingManagementApplication	GeneralLightingCommandPattern LightingUserReadjustmentPattern
EfficientLightingApplication	LightLevelPattern LightStatusPattern OccupancyPattern
GeneralCommandForShutterGroupsApplication	ShutterLevelPattern
UnsupervisedControlForShutterGroupsApplication	ShutterLevelPattern
ThermalComfortSupportApplication	ComfortRelativeHumidityPattern ComfortTemperaturePattern OccupancyParameter
SelfCheckingShutterManagementApplication	GeneralShutterCommandPattern ShutterUserReadjustmentPattern
UnusualPresenceDetectionApplication	Energy & Resources
AlarmOversightDetectionApplication	AlarmStatusPattern OccupancyPattern
ReachSetpointTemperatureApplication	ComfortTemperaturePattern OccupancyPattern
ReachSetpointRelativeHumidityApplication	ComfortRelativeHumidityPattern OccupancyPattern
SelfCheckingTemperatureManagementApplication	TemperatureUserReadjustmentPattern
SelfCheckingRelHumidityManagementApplication	RelHumidityUserReadjustmentPattern
DHWRecirculationApplication	HotWaterConsumptionPattern
CustomizedBillingApplication	DwellingConsumptionPattern
ElectricPeakShavingApplication	DeviceConsumptionPattern DeviceStatusPattern DwellingElectricityConsumptionPattern ElectricitySpotPricePattern
CheckEnergyBehaviorApplication	DeviceConsumptionPattern

Table 4.9: Applications according to [129] and their relations to patterns in the User Behavior & Building Processes ontology

Application	`isInfluencedBy` **Pattern**
SmartHomeCustomizationApplication	ComfortRelativeHumidityPattern ComfortTemperaturePattern DeviceConsumptionPattern DeviceStatusPattern DwellingElectricityConsumptionPattern GeneralLightingCommandPattern LightingUserReadjustmentPattern LineConsumptionPattern LineStatusPattern OccupancyPattern RelativeHumidityUserReadjustmentPattern ShutterUserReadjustmentPattern TemperatureUserReadjustmentPattern WindowStatusPattern

Additional to interconnections, hierarchies are created for processes and applications in order to classify different categories. Processes are on the one hand divided according to the comfort parameter they influence (e.g., airing, humidity, lighting, temperature) and how they influence these parameters (e.g., cooling process, heating process). On the other hand, a subclass `DeviceCentricProcess` is modeled to categorize processes that are performed by specific devices: a process that has an associated device and involves a command to control the device can then be categorized as `DeviceCentricProcess` by the OWL DL reasoning mechanism. Its associated OWL equivalence axiom is exemplarily shown in Figure 4.17.
For applications, a hierarchy is created that follows the list of applications proposed in [129]. All mentioned applications and their pattern dependencies are represented in the ontology as well: the respective applications and their associated patterns are shown in Table 4.9 which resembles the table in [129] p. 49, while several naming adaptations are made to keep it consistent with the other ontology modules (e.g., "shutter" instead of "blind" for shading devices). An additional application subconcept is entitled `SelfCheckingApplication` and includes applications that measure user satisfaction according to adaptations made by users after automated settings have been applied.

4.4.4 Definition of Profiles and Patterns

According to the revised literature, and trying to find a profile definition as simple and flexible as possible, behavioral profiles (and habit patterns by extension) are defined as *time-related data structures* (i.e., univariate time series) that abstract the users' behavior or usage of specific devices, objects and spaces in the home environment [132]. A *profile* may be seen as the most basic building part for a profile-based system: it represents an image of process-behavior over a certain specified time. According to [132], a profile has several characterizing values. It

belongs to a *type*, addresses an *object*, takes values within a *range*, entails different *fields*, covers a *scope* and is identified by a specific *date* or *interval*. Profiles can strictly be differentiated from processes: a process is some action that the system (or a user) can take in order to change a certain environmental value in the building. A profile on the other hand is a record of the effect of processes. A process can lead to the creation of a new profile, and a profile in fact implies which process needs to be started next to reach a desired state.
There also exists a relation between profiles and applications in the smart home in a way that applications may be supported by regularly occurring patterns retrieved from profiles. A *pattern* is calculated by a pattern generator from a group of individual habit profiles. These patterns can conversely be used to reach the goal described by the application in a more efficient way, which is extensively discussed in [129]. A pattern may describe the immediate future states for a building facility or the building environment according to recorded and analyzed past user behavior. It is calculated outside of the ontology by an own component responsible for pattern generation (i.e., pattern generator) and stored back into the ontology with a reference to the profiles used for the creation of this specific pattern and prediction values for a certain environmental parameter (cf. Figure 4.17). It is valid for a specific period which could be a day per week or any other period for which a regular user behavior has been detected. Usually, the granularity of scheduled values follows the sampling rate of the profiles used in pattern creation. The following paragraphs cover the different characterizing values for profiles and patterns as identified in [132] and show how these values are represented in the created ontology for user behavior and building processes.

Profile type In the ontology, profile types are represented as OWL subclass hierarchy. The top-level concept `Profile` is subdivided into several classes created to characterize different profiles. The created classification again follows the one proposed in [129]:

- `ConsumptionProfile`: Related to consumed variables or costs, they usually consist of ratio scale profiles of system outputs but may also include interval scale profiles, e.g., profiles capturing the consumption of electrical power, water or gas.

- `UsageProfile`: This class represents profiles that monitor double valued system states, e.g., on/off, passive/active, true/false, open/close, present/absent. Due to the influential role in most of home applications, its subclass `OccupancyProfile` is one of the most important representatives.

- `SetpointLevelProfile`: This type stands for interval scale system input information, e.g., setpoint temperatures, lighting levels. For the convenience of the pattern discovery algorithms, a split into into *usage* and *setpoint/level* profiles is realized.

- `AbstractAimProfile`: This is the class of profiles that supervises nominal scale variables, which usually address complex user activities (e.g., sleeping, having meal, cleaning, sports). This type of profiles may thus be used to represent user preferences in the original approach. In the described ontology, user preferences are represented in an own module (Users & Preferences, cf. Section 4.5) and therefore, the abstract aim profile is only being modeled at this point in order to stay consistent with [129].

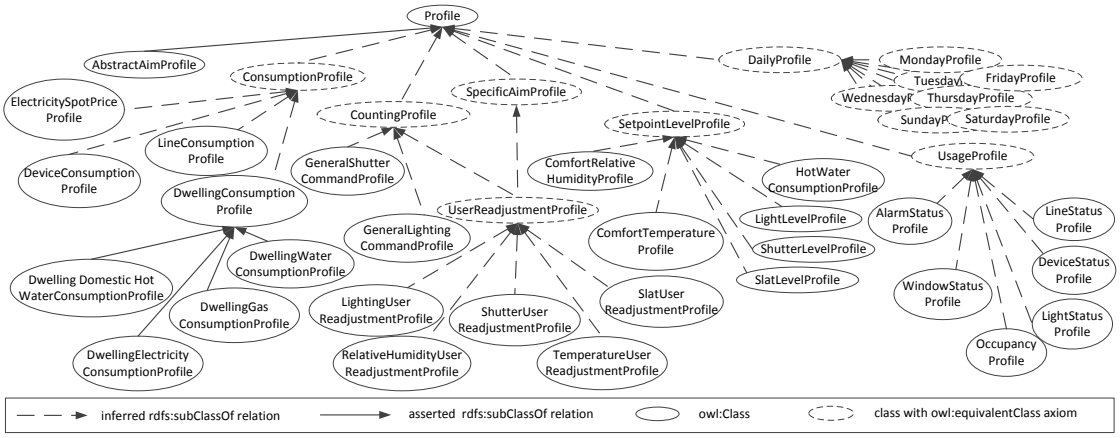

Figure 4.18: Profile hierarchy, main classes and sample habit profiles as represented in the ontology. For simplicity of the figure, only inferred subrelations of profiles to the main concept `Profile` are illustrated.

- `SpecificAimProfile`: Finally, profiles that are specific to certain facilities and purposes are represented with this class. Additional to the five main profile types, also subclasses reflecting the profile scope (i.e., daily, monthly, weekly) are defined.

In further consequence, all habitual profiles for a home environment mentioned in [129] are modeled in the ontology. Additionally, also profiles mentioned in connection with application definitions (cf. [129] p. 49) are added, so that all described dependencies between applications and patterns (cf. Table 4.9) can be represented. As the main profile types except for abstract aim profiles are defined with the help of equivalent axioms, OWL DL reasoning can be used for categorization of these habit profiles into one of the five main types (Figure 4.18).

Pattern type Similar to profiles, also for patterns an OWL subhierarchy is provided. The main categorization that is realized identifies pattern types according to their validity and their application. With respect to validity, several time-based concepts are provided (e.g., weekend pattern, weekday pattern, seasonal pattern) that allow the PTG to categorize patterns according to the time interval when they should be considered by the control system. Regarding the application, there may exist different pattern types for each habitual profile (e.g., comfort relative humidity pattern, comfort temperature pattern, occupancy pattern) reflecting the profiles that this pattern is based on. In this case, the categorization follows the general profile classification already presented. The relation between patterns and profiles is modeled as object property (`isBasedOn`).

Profile/pattern object Objects of home profiles may be of different types and may be related to a profile or a pattern via the `hasFocusObjective` property, and its subproperties

hasPatternFocusObjective and hasProfileFocusObjective: focus objectives can be devices or house elements (e.g., lamps, sockets), groups or lines of such elements (e.g., first floor heaters, bathroom lamps, ring circuit for switch sockets), spaces, zones or sub-zones (e.g., corridors, bedrooms, storerooms), dwellings or houses, when considering the facility as a whole, individual users or user groups or also external influences such as the current electricity price per electricity provider or weather forecasts from a particular report source. Because of the wide variety of value types possible for this property, the range of the created object property is not limited and can be anything (cf. Figure 4.17). Information about the characteristics of connected objects needs to be represented in the respective ontology modules (e.g., Architecture & Building Physics ontology, Energy & Resources ontology, Weather & Exterior Influences ontology) and as such, this object property and its subproperties can be seen as connecting properties between elements of different ontology modules (cf. Figure 4.2). In case of patterns, the property is modeled to be inferred by a simple property chain:

```
ObjectProperty: hasPatternFocusObjective
  SubPropertyChain: isBasedOn o hasProfileFocusObjective
```

This property chain assures that the focus objective of the pattern always resembles the object of the profiles this pattern instance is based on.

Profile range The object property hasProfileRange describes an interval of allowed values for this profile. The connected RangeParameter concept specifies minimum and maximum range value through associated datatype properties modeled as subproperties of the hasValue property (hasMinValue, hasMaxValue, cf. Figure 4.19). These values do not represent absolute restraints but can be seen as barriers for the usual range of this profile. For example, if the PTG recognizes values outside of this range in a profile, it can identify this profile as outlier and disregard it in the pattern generation. This in turn leads to stronger and more reliable patterns as profiles not representing usual user habits do not influence the created habit patterns.

Profile scope According to [129] p. 13, the profile scope is *"the total length/duration of an individual profile, constituted by all the profile fields together"*. The scope of a profile therefore describes how often a profile is generated (e.g., daily, weekly, monthly, yearly), and is delineated in the ontology with the hasScope property. The scope itself is realized as a process parameter (ScopeParameter) with a defined set of individuals (cf. Figure 4.17): for an accurate representation of context in the smart home system, daily, weekly and monthly scopes are considered necessary and sufficient. At the same time, individual profile subclasses are modeled for different scopes (e.g., DailyProfile, MonthlyProfile) for which membership can be determined according to the scope parameter they are referring to.

Profile/pattern interval A profile records observations in the home for a specific period and likewise, a pattern is valid for yet another period. Therefore, two separate relations are modeled as object properties: records relates a profile to the time period for which it is created, for example a 24 hours period. Further, validFor models the connection between a pattern and

the time period for which it is valid, as it may be possible that a detected pattern is only useful to optimize system operation for a specific time of the year (e.g., July to September).

Pattern representativeness/reliability The representativeness and reliability of a pattern are indicators of how well a generated pattern reflects reality with respect to the profiles it is based on. If there exists a clear pattern that can be found throughout profiles used for pattern generation, these values are high. Otherwise, if there is little coherence between profiles or if there exist a lot of intermediate profiles that do not resemble the detected pattern, the representativeness and reliability tend to be rather low. The values are calculated for each pattern by the pattern generation algorithm. In the ontology, these two characteristics are modeled as datatype properties while their possible values are restricted to the interval from 0 to 1. In simplified terms, these assessments can be interpreted as the probability that a specific pattern will reoccur.

4.4.5 Instantiation and Reasoning

To give an example of the utility of an ontology-based data representation for habit profiles in order to describe building context the achievement of thermal comfort with the help of profiles is discussed. At this point, the elaborations only serve an illustrating purpose to explicate dependencies between concepts and their values while the integration and usage in a complete use case scenario is pictured in Chapter 5. Figure 4.19 shows full parametric dependencies for a specific profile as represented in the context ontology module. Further, Figure 4.20 shows an application instantiation (i.e., to reach the setpoint temperature), together with a sample process realizing this application and influencing patterns. To keep the figure concise, full dependencies are only shown exemplarily for one process and pattern. Certainly, more processes not depicted in the figure (e.g., turn down the radiator, open the window, turn on the air condition) may also influence the portrayed application. Similarly, only one profile that forms the basis for the shown pattern is pictured as example while a pattern is usually created from a set of profiles.

Profile The PFG may create daily temperature profiles for the living room in a building. A concrete instance is the "temperature profile of the living room on 11.02.2013" (cf. Figure 4.19) which can be represented as a member of the `ComfortTemperatureProfile`. This subclass of the top-level class `Profile` can be inferred to be a child of the more specific `SetpointLevelProfile`, one of the five main classes identified for profiles in Section 4.4.4. The profile instance is addressing a certain room (`hasFocusObjective`) which may be a simple string value or, as in the present case, represented in an own building ontology, describing its characteristics. The profile has 24 associated values connected via the `hasProfileParameterValue` object property stored for each hour of a day. Exemplarily, Figure 4.19 shows the profile value recorded between 20:00 and 21:00 on this day. The sampling rate of one hour is represented in an own datatype property (`hasSampleRate`) as 60 minutes. The values of the profile parameters usually range between a minimum of 15 and a maximum of 28 degrees Celsius which is represented with the help of the `hasProfileRange` object property and an associated individual. By definition, a daily profile covers the temperature curve of one day (`hasScope`) and this example represents the situation specifically measured

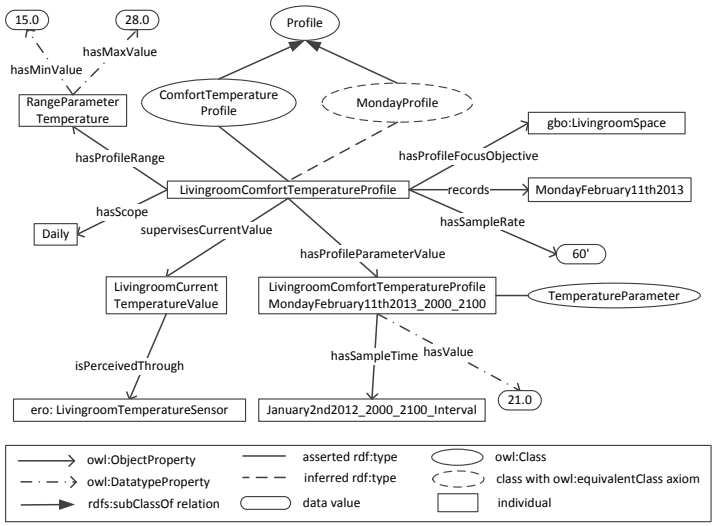

Figure 4.19: Example of a profile instance as represented in the ontology

on February, 11th, 2013 (`records`). The profile is further connected to a particular current value through the `supervisesCurrentValue` object property which is perceived through a particular sensor stored in the Energy & Resource ontology.

Pattern After a set of profiles has been analyzed by a PTG, the component may detect a specific pattern recognized for some Mondays in a row and therefore creates a Monday pattern which is based on (`isBasedOn`) a set of Monday profiles. Again, in Figure 4.20 only the relation to one Monday profile is shown exemplarily. The pattern has a reliability and representativeness of 0.93 indicating that it accurately reflects reality. Further, it has hourly scheduled values of desired temperature setpoints (`hasScheduledValue`) which have been calculated by the PTG (only the value for one interval is shown). The pattern is a prediction for all Mondays in the month of February which is indicated by the `validForPeriod` relation to a validity time interval (i.e., February). While the pattern is manually asserted to the class `MondayPattern` it can additionally be inferred to be of type `ComfortTemperaturePattern` as well as `WinterPattern`, according to the profile type it is based on (cf. Figure 4.19) and its valid time.

Application The pattern may further be used by a concrete application instance, namely the `LivingroomReachSetpointTemperature` application, which may be performed by a setpoint temperature controller to adjust the heating based on the habitual temperatures found comfortable by users and represented in comfort temperature patterns. Firstly, the capabilities of the `owl:inverseOf` property can be exploited: from the manually asserted relation `hasInfluenceOn`, the standard OWL DL reasoning mechanism can infer that there also exists an inverse relation `isInfluencedBy` which is defined as `owl:inverseOf` to `hasInfluenceOn` in the ontology. This way, when considering communication with the

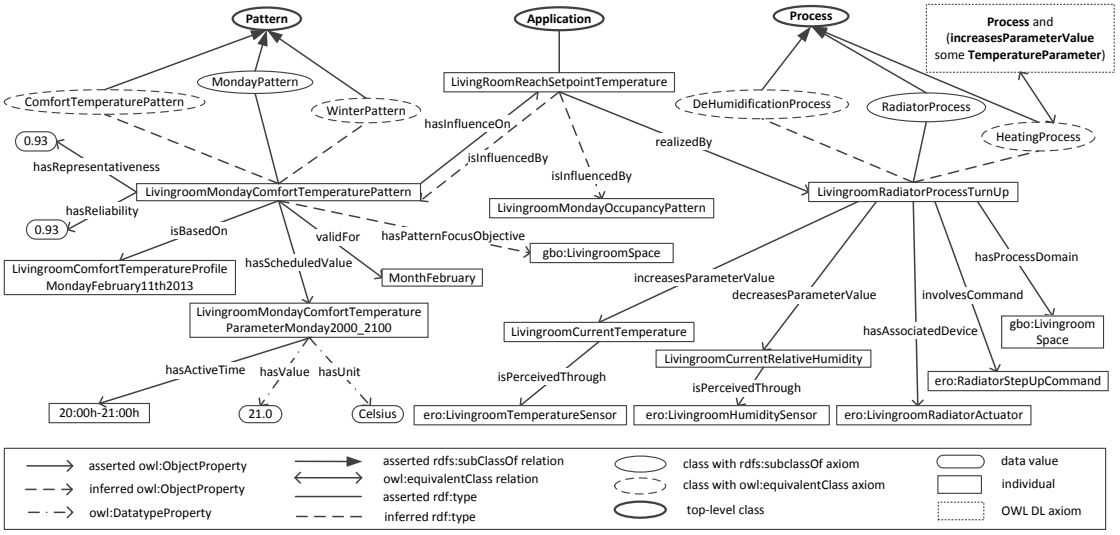

Figure 4.20: Examples for a concrete pattern, application and process and their representation in the process ontology. For dependencies of the shown example profile and its associated parameter values, cf. Figure 4.19

knowledge base in a multi-agent system, only the PTG agent has to store which application is influenced by its generated pattern. The control agent, operating on applications, may retrieve the information directly through the inferred inverse relation. The application is further influenced by predictions about future presences and absences of occupants in the supervised room, which may be found in a dedicated `OccupancyPattern` not described in detail at this point.

Process Processes involved to reach thermal comfort are for example turning the radiator actuator up and down, thus adjusting the temperature environmental value. As an example, Figure 4.20 shows the concrete `LivingroomRadiatorProcessTurnUp` instance that describes the influence of turning up the radiator actuator. The ontology depicts the object of influence for this process via the `hasProcessDomain` property and provides relations for representing which command as well as which device are involved through the two object properties `involvesCommand` and `hasAssociatedDevice`. Thorough information about devices and possible commands are again described in another dedicated ontology (cf. Section 4.3). Parameters which are influenced by the process are in this case connected via the already mentioned `increasesParameterValue` and `decreasesParameterValue` relations: as subproperties of the `influencesParameterValue` property, they describe how environmental parameters are influenced by the pictured process. In this case, increasing the radiator heat output by turning up its radiator actuator is modeled to decrease relative humidity in the room while increasing the current temperature value. While the process is manually modeled as being a radiator process its influences on current values (i.e., current temperature, current relative humidity) allows the reasoning mechanism of the ontology to additionally infer it to be of type

`HeatingProcess` and `DeHumidificationProcess`. The axiom describing the heating process class is exemplarily shown in Figure 4.20, while hereby any process that increases the temperature is defined to be classified as heating process.

4.4.6 Ontology Metrics

As a summary, the metrics of the User Behavior & Building Process ontology are shown in Table 4.10. The ontology currently defines 157 classes, 32 object properties and 10 datatype properties. The main focus of the created classes lies on applications and processes to represent possible actions in the building and patterns as well as profiles for assessing user habits. A set of well-defined object properties describes the relationships between these main hierarchies and also a small amount of datatype properties are defined for further representation of facts.

Table 4.10: Metrics of the User Behavior & Building Processes ontology module

		User Behavior & Building Processes ontology
TBox	Named Classes	157
	Object Properties	32
	Datatype Properties	10
ABox	Instances	15

The ontology includes 15 individuals which are on the one hand used to picture the profile scope. On the other hand, certain individuals and concepts from the W3C time ontology are necessary to model time-based profile classifications. Therefore, the presented metrics include two object properties, one datatype properties and nine individuals from the W3C time ontology.

4.5 Users & Preferences

While user preferences may be automatically inferred through a pattern-based mechanism as just described, it is also important to give the user the possibility to define his or her own preferences. The last thing an occupant wants is to feel patronized by a system that takes away the burden of controlling the environment. As already mentioned in the introduction, it is not desirable to completely take away control from the resident as he/she wants to stay in control and define preferences, adjust parameters and create schedules when necessary and desired. A comprehensive smart home system therefore also needs to address user preferences and demographic information about its users. In this respect, the *Users & Preferences* module may be seen as additional part of the user representation in the envisioned system: while the user context module presented before represents information about the users that can be automatically retrieved and inferred through sensor readings (i.e., behavior and habits), the preferences module contains information that is manually added by users or system engineers e.g., through a graphical user interface, and corresponds to customized or standard comfort definitions.
The Users & Preferences ontology module represents a smaller domain compared to the already discussed parts of the smart home ontology which nevertheless may be regarded important: it

stores various aggregations of preferred values (e.g., temperature, humidity, luminosity) and this way provides a notion of *comfort* to the system. Apart from user-centric comfort definitions also a default description following certain standards (e.g., the ASHRAE standard for thermal comfort [5]) may be defined and can be used in case a customized user preference is not available.
The following sections describe an ontology for user information and manually-defined preferences representation in a multi-agent-controlled smart home system. The reviewed ontology was created as part of a master's thesis supervised by the author. In the present work, the main design decisions, defined concepts as well as their interrelationships and possible connections to other ontology modules are shortly outlined. A detailed discussion of this ontology module can be found in [230]. To what extent preferences represented in this ontology module influence previously introduced patterns and in further consequence the system behavior lies in the responsibility of the multi-agent system operating on the ontology and therefore is not discussed at this point.

4.5.1 Related Work

Apart from preliminary profile-based approaches, there also exist some works that to some extent consider ontologies as a possible solution for the representation of context and user preferences in smart homes. In [48] a standard ontology for ubiquitous and pervasive applications is described. The authors explain how to represent agents, their beliefs desire and intentions as well as space, events or security policies. A user profile ontology is mentioned, while in this case a profile describes the users themselves, their social relations and personal preferences. In [175], the authors describe the challenges for creating a comprehensive ontology capturing user context. The proposed context-aware system also deals with specific users and their preferences as well as activities in the represented environment. The article further describes an algorithm on the recognition and exclusion of duplicate data in order to limit the amount of data to be stored in this ontology. The authors of [246] present a user modeling service for a smart home and introduce a service architecture that among other services facilitates the learning of context-dependent user preferences from an interaction history through an ontology-based knowledge representation. The approach includes the use of stereotypes when creating user profiles to reduce system configuration needs. A user profile is again seen as a collection of user preferences and personal data which has to a great extent been entered through a graphical user interface. The project focuses on representing user activities while also automatic deduction of preferences through interaction is envisaged. In [154], the author proposes user profiles including environmental preferences like lighting, temperature or humidity settings per user alongside activity profiles to model different preferences for single users reflecting time and type of activity they perform. Further, the article introduces object profiles storing behavior descriptions for agents representing environment objects (e.g., doors, windows, lighting) in a multi-agent system. The authors of [74] propose an ontology-based generic context management model that is capable of collecting, representing and storing user context. The model is based on a representation of context data, context instances and rules which may be defined by the users or automatically derived. In further consequence, the proposed model can be applied in a pervasive environment to provide context-aware services that can be used to take decisions on behalf of a user. While they present a case study on reasoning no deeper insights on the structure and extent of the proposed

context ontology are given. The article [106] describes a context-aware middleware architecture for an intelligent building relying on a formal context model based on an ontology that depicts semantic representations. The authors introduce several concepts and relationships necessary to comprehend context and environment in a smart home. An overview of the proposed system architecture is given and it is also shown how ontology reasoning can be exploited. The authors further briefly outline the application of the created ontologies in a pervasive computing environment. While the article focuses on context representation in smart homes, user preferences are not explicitly treated.
While a direct reuse of one of these previously defined ontologies would be desirable, this is often not feasible due to a different focus. Mostly, the ontologies in the context of smart homes focus on user context, while user preferences are often neglected or only treated superficially. Further, many times the presented ontologies are not publicly available and often the only information about the created conceptualizations can be found in the cited publications. Therefore, the decision was made to create a new user preferences ontology from scratch. The benefit that arises from a newly created ontology is that such a conceptualization particularly built for representing user preferences and actions in the context of an autonomous smart home system can be highly customized and tailored to that specific field of application. Nevertheless, some constructs and relationships found in ontologies just presented can be seen as inspiration for the Users & Preference module in case they deemed to be useful and applicable.

4.5.2 Actors, Preferences and Activities

Using a pattern-based approach for autonomous habit recognition in the described system, the ontology part created as basis of this mechanism (cf. Section 4.4) is capable of representing automatically detected user habits. To additionally give the user the possibility to state his own desires regarding behavioral schedules, a user preferences ontology is considered, representing a subset of context-related concepts and properties. Manually-defined preferences described in this part may in further consequence be used by the autonomous system as additional information to complete the knowledge about expected future behavior and allow occupants to influence system operation.
Figure 4.21 gives an overview of the top-level concepts of the created ontology. The following paragraphs present a short summary of the main concepts defined to capture user comfort preferences and activity schedules.

Actor The purpose of the actor concept is mainly to represent the residents of the building in the smart home system. Because the ontology is modeled as foundation of a multi-agent-based system in which collaborating agents can take autonomous actions on behalf of the residents, actors in this case can either be human users or system agents. These different types are differentiated in the actor subhierarchy. For humans, the concept `HumanActor` is provided: in this respect, the knowledge base allows to describe different types of human actors (e.g., male, female, young, aged) through the auxiliary top-level concepts `Age` and `Gender`. A basic form of user satisfaction can be created through the `LevelOfSatisfaction` concept. The ontology further provides the possibility to associate manually added user preferences with single indi-

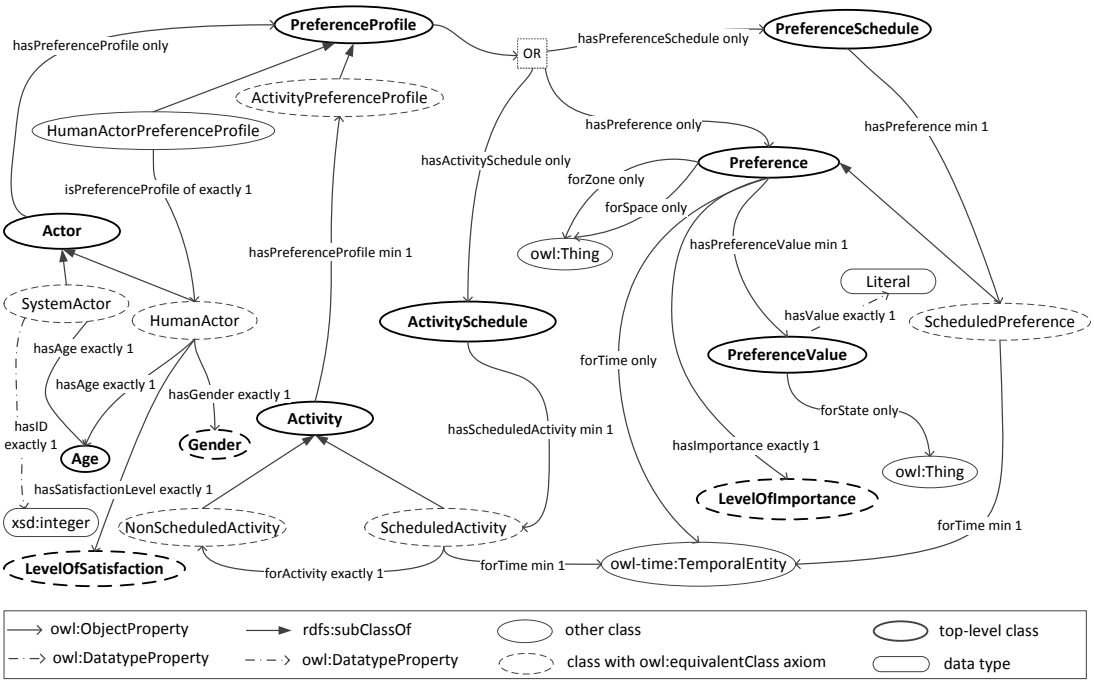

Figure 4.21: High-level view of the Users & Preference ontology presenting top-level concepts and properties. Complete dependencies of pictured subclasses are not shown to keep the diagram concise and highlight relationships between top-level classes.

viduals in the smart home through the `hasPreferenceProfile` object property. Moreover, for agents of the multi-agent system, the class `SystemActor` is available, which may be used to group agents according to their type, and, for example, describe representatives for users in the system (i.e., user system actors, cf. Figure 4.22).

PreferenceProfile Preferences for human actors are represented in a subclass of the preference profile hierarchy (`HumanActorPreferenceProfile`) and may be individually created by each user through a user interface. Preference profiles can further be divided into two main types which are scheduled or non-scheduled profiles. The `ScheduledPreferenceProfile` class is created to allow the definition of manually created preference schedules that may be used additionally to the automatically detected habit profiles described in Section 4.4. One example for a scheduled preference would be the customized setting of various lighting scenes for different times of the day. In this case, a preference profile is related with an element of type `PreferenceSchedule` through the `hasPreferenceSchedule` object property. A scheduled preference profile may include several preference schedules (e.g., temperature, presence, activity). Otherwise, the concept `NonScheduledPreferenceProfile` includes profiles defining general preferences that are not related with a particular time. Such a profile could

for example represent a basic notion of comfort for a particular occupant including his or her personal comfort temperature, lighting and humidity settings.
Another necessary distinction can be made between manually-defined human comfort preferences and standard preferences. The `HumanActorPreferenceProfile` only includes settings that are manually defined by an occupant, while the `StandardPreferenceProfile` is used to represent comfort values according to standards. The second type may then be used in case no manually-defined preferences are available for a particular resident or the user is unknown to the system (e.g., guest user).
A special case of preference profile is the `ActivityPreferenceProfile`: this class groups user-defined profiles that are related to a particular activity. This way it becomes possible for the occupant to define multiple preferences related to the present or scheduled situation and context.

Activity The activity concept hierarchy is provided in the ontology to allow a basic description of activities and associated settings. Firstly, a classification into passive and non-passive activities is created, which splits activities into actions in which the occupant is physically active (e.g., sport, cooking, cleaning) and others in which the resident mainly remains passive (e.g., reading, sleeping). This basic distinction facilitates the association of an appropriate standard preference with an activity in case no manually-entered preferences are provided. For example, it could be inferred that for passive activities the comfort temperature is usually higher than for activities that involve a lot of movement. Further, as already for preference profiles, non-scheduled and scheduled activities are distinguished: for `NonScheduledActivities` the user may define general comfort settings stored in activity preference profiles and associated with the particular activity through the `hasPreferenceProfile` property. Moreover, activities can be scheduled for a particular time frame and in this case, can be inferred to be of type `ScheduledActivity`. A connection between scheduled and basic activity is realized through the property `forActivity` and allows the association of the scheduled activity with preferences generally defined for this particular activity (cf. Figure 4.22).

Activity-/PreferenceSchedule In order to group scheduled activities and preferences, classes to represent these schedules are provided. A member of the class `PreferenceSchedule` can involve personalized settings of parameters like customized lighting of the building environment for particular times of the day but may also include the scheduling of tasks to be executed by the system autonomously at pre-defined times. The concept is designed to store a schedule for several time-sensitive preferences which are of type `ScheduledPreference`, a subclass of the `Preference` top-level concept.
Further, to describe a chronological sequence of activities, the `ActivitySchedule` class is provided. Similar to the `PreferenceSchedule` concept it has the purpose to keep a user-defined timetable of activities that are scheduled for a particular time of the day. The different `ScheduledActivity` individuals connected through the `hasScheduledActivity` object property then include the particular preferences that should be effective at the time the scheduled activity is created for (`forTime`).

Preference This top-level class can be seen as a central class in the ontology and is concerned with the representation of different types of comfort preferences. Firstly, they are separated according to the type of setting that is represented. The concepts `AppliancePreference` and `ApplicationPreference` are created to differentiate between preferences that involve an appliance and hence have a connection to a facility represented in the Energy & Resource ontology (e.g., `LampPreference`) and other preferences that are related with applications (e.g., reaching comfort temperature, reaching visual comfort) which are found in the User Behavior & Building Processes ontology module. A preference that is neither appliance- nor application-centric describes occupancy which is represented by the concept `PresencePreference`. Additionally, further subclasses are used to classify preferences according to their usage. For example, the `ScheduledPreference` subclass may be used to identify a preference that is only valid for a particular point in time (`forTime`). The subclasses `StandardPreference` and `UserDefinedPreference` can again be used to distinguish between manually-defined and standard preferences. Each preference has a relation to one or more elements of type `PreferenceValue` describing the values that are connected to this preference. In case an appliance for which a preference is defined has more than one state, certainly, also one preference value for each state may be defined and needs to be connected to the preference element through the `hasPreferenceValue` property (cf. Figure 4.22). The two object properties `hasSpace` and `hasZone` are provided to make the association of a preference with a particular part of the building possible. The object property `hasImportance` can further be used to realize a simple form of priority scheme for preferences. The associated top-level class `LevelOfImportance` includes a closed set of individuals representing different priorities and allowing the multi-agent system to choose one preference over another in case more than one possibly conflicting preferences are available. Which preference is chosen depends on the system policy and goal-plan of the multi-agent system and is therefore out of scope for the ontology representation.

PreferenceValue Finally, a concept that allows the definition of preference values is provided. An instantiation of this class contains the actual value of the preference connected through the `hasValue` datatype property and may also define a state for which it is defined (`forState`). The reference to a state is particularly needed in case a preference value is associated with an appliance that has more than one state which may be controlled. For the class of preference values a differentiation into binary and continuous values is realized through subclassing: the `BinaryPreferenceValue` only includes preferences that can be represented using two states (e.g., occupied/not-occupied, on/off) while the `ContinuousPreferenceValue` may be applied in a more flexible manner for multiple discrete or continuous state preferences.

4.5.3 Instantiation and Reasoning

To further illustrate the usage of the conceptualization and some of its inferencing capabilities, Figure 4.22 shows a sample instantiation of the main concepts. Firstly, in this example, an adult and male human actor (`HumanActor1`) is represented. From the associated age and gender, the inference mechanism can deduce this human actor to be of subclass `MaleHumanActor` and

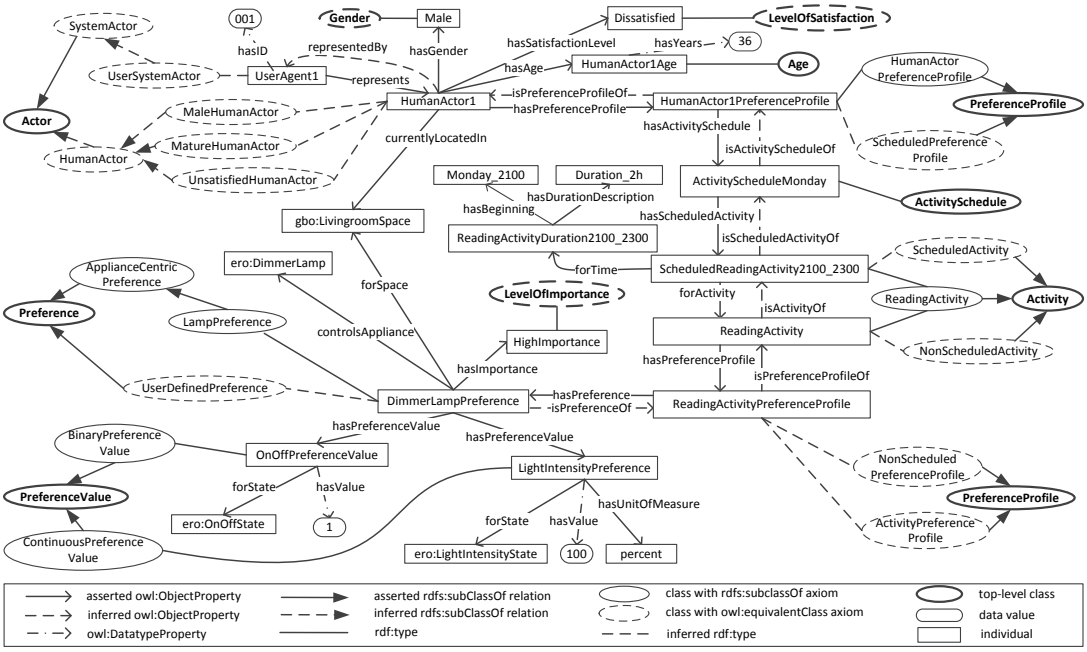

Figure 4.22: Example instantiation for the Users & Preferences Ontology. Individuals defined in other ontology modules are shown with the respective ontology module prefix. `rdf:type` relations are only shown for individuals of the Users & Preferences ontology.

`MatureHumanActor`. A satisfaction level (`LevelOfSatisfaction`) can further be used to describe the mood of the resident towards the system control and can be manually entered by the user through a user interface or may also be associated to a user by the system through a mechanism detecting readjustments on the user level. In the present example, the connection to the `Dissatisfied` individual is an indication for the ontology reasoning mechanism that this user is an `UnsatisfiedHumanActor`. The illustrated example further shows a user agent instance (`UserAgent1`) which `represents` the shown user in the multi-agent system and an associated inverse property allows to infer that the shown human actor is `representedBy` the particular user agent. Because of this relation to a human actor, the user agent can be inferred to be of type `UserSystemActor`. The actual position of a specific human actor in the building can further be expressed through the `currentlyLocatedIn` object property which models a connection to a space or a zone as defined in the Architecture & Building Physics ontology.
The `hasPreferenceProfile` property and its inverse are connecting a human actor individual to a preference representation. The shown profile (`HumanActor1PreferenceProfile`) actually describes a schedule of activities using the `hasActivitySchedule` property and therefore can be inferred to be of type `ScheduledPreferenceProfile`. To exemplify the relationships in the ontology, an associated activity schedule and one scheduled activity are shown exemplarily. While the `ActivityScheduleMonday` instance groups activities defined for a particular period (e.g., day), the time for which an activity is planned is directly

associated with the individual representing the scheduled activity, i.e., ScheduledReading-Activity2100_2300. Additionally to its base class ReadingActivity, the association of the shown activity with a given time may be used to classify it as ScheduledActivity. To create a connection with user-defined preferences for this activity, the forActivity property firstly relates the scheduled activity to a NonScheduledActivity with the same base class. This differentiation between scheduled and non-scheduled activities is designed in order to store user-defined preferences for a single activity only once in the knowledge store while still allowing multiple activity schedules relying on the same set of activity preferences. The activity can then be related to user-defined preferences through the hasPreferenceProfile property. This connection of the profile representing preferences to a non-scheduled activity makes it possible to additionally classify it as NonScheduledPreferenceProfile and ActivityPreferenceProfile.
The bridge between preference profiles and user preference definitions is realized through the hasPreference property and its inverse counterpart. The figure shows one example preference of the defined reading activity preference profile, a DimmerLampPreference. This particular instance involves an appliance which is connected through the controlsAppliance property and may point to a facility stored in the Energy & Resource ontology module. In this case, the associated appliance is a dimmer lamp. Further, the forSpace object property is used to associate the preference with a certain space defined in the Architecture & Building Physics ontology. This connection can safely be omitted if the space is already identified for the appliance in the Energy & Resource ontology model. The hasImportance relation indicates a user-defined importance level: in the pictured example, the user marked this preference as highly important, telling the smart home system that whenever a reading activity is scheduled, it should preferably use this setting over other possibly defined preferences with lower importance. As this preference is associated with a preference profile created by a human actor, it can be inferred to be of type UserDefinedPreference contrary to standard preferences which are associated with standard preference profiles.
Finally, an appliance connected to the preference ontology may have more than one states that can be controlled by the system. In the case of the dimmer lamp, two different states are assumed to be controllable, one, which allows to switch the appliance on and off and another which allows to specify the lighting intensity of the dimmer lamp in percent. The single preference for the dimmer lamp now involves two preference values, one for the on/off and one for the dimming functionality of the associated facility. The individual OnOffPreferenceValue can be seen as BinaryPreferenceValue which represents a preference with two possible states. The other illustrated preference value LightIntensityPreference is modeled as ContinuousPreferenceValue and as such may include more than two states. In this case, the preferred dimming state is expressed in percent which is defined by the element connected through the hasUnitOfMeasure property. Further, through the forState property, a particular state of the facility is associated with each preference value. The state is in the present case again being described in more detail in the Energy & Resource ontology module.

4.5.4 Ontology Metrics

The metrics of the Users & Preference ontology are shown in Table 4.11: this ontology module consists of 72 classes, 29 object properties and 6 datatype properties. These metrics include one class of the OWL-Time ontology to represent scheduled preferences and activities. Further, a class of the Ontology of units of Measure (OM)[17] is included to represent the unit for preferences. The classes are divided among the identified main top-level components with the majority of classes being used for describing actors, activities, preference profiles, preference schedules and preferences.

Table 4.11: Metrics of the Users & Preference ontology module

		Users & Preference ontology
TBox	Named Classes	72
	Object Properties	29
	Datatype Properties	6
ABox	Instances	9

The module also includes 9 individuals which are used to represent the gender of human actors and also their level of satisfaction, as well as the level of importance of a particular preference. This chapter only gave an overview of the Users & Preference ontology that fulfills the requirements needed for taking into account comfort preferences of individual users in an automatically controlled environment. The introduced ontology is intentionally only described from a high-level viewpoint and a more detailed discussion can be found in [230].

4.6 Weather & Exterior Influences

In the operating phase of a building life cycle, the importance of real-time and forecast weather information is undeniable [56], as the inclusion of accurate weather data in automated building control leads to considerable energy savings [183]. Therefore, exterior influences and changing weather situations cannot be neglected and need to be considered when it comes to the optimized control of a future automated home. When integrating short-term weather forecasts into a smart home system, these data can be used to infer proper actions according to circumstances that are imposed on the building by its environment and solve tasks like reaching comfortable thermal conditions most energy-efficiently. Additionally, to guarantee user comfort, this information can be exploited for example to utilize natural lighting through sunlight or solar heat traversing windows.
According to [190], there exist distinct patterns for different regions where renewable energy can be utilized and their energy gain depends on local weather patterns, changing on a daily and seasonal basis. Several projects illustrating advanced building control strategies already consider weather parameters as valuable addition for an energy-efficient operation as discussed in Section 4.6.1. However, one important yet still lacking part is a comprehensive representation of

[17] http://www.wurvoc.org/vocabularies/om-1.8/

the exterior state with all its influence parameters. In order to exploit the whole savings potential of a residential home in a smart grid, weather influences like solar radiation and temperature need to be considered. Knowledge about this domain for example facilitates the exploitation of "free" cooling/heating using favorable meteorological conditions [67].
Weather prediction and forecasting are highly complex fields with many uncertainties. However, as weather conditions may have a distinct impact on monthly electricity demands [124], an integration of weather information in smart home control is a valuable addition to design advanced optimization strategies. Currently, smart home systems seldom assess external influences and thus energy is wasted in the daily operation of residential homes. In this respect, as preliminary work shows (cf. Section 4.6.1), a system incorporating meteorological influences allows buildings to be operated more energy efficiently, sparing residents of high energy costs and energy suppliers of peak demands.
The following sections describe a weather ontology for smart home operation in smart cities. To put the Weather & Exterior Influences ontology in context with the foundational use cases presented in Chapter 3, the subsequent paragraph shortly describes possible scenarios in which knowledge of the weather situation can be used to influence the energy behavior of a building. Two attractive use cases in which the inclusion of the exterior situation can make a significant difference are use case 2 "to automatically achieve thermal comfort on behalf of the resident" and use case 7 the "predictive operation of HVAC services". In different situations for these use cases the inclusion of weather information may lead to a better energy-related behavior of the building. When, for example, during the winter season, the forecast reports sunny weather, the system can await sunrise and as the sun reaches certain parts of the building throughout the day open blinds and shutters to use solar radiation for natural heating of specific rooms. When again fog, rain or generally moist weather states are expected, a room can be naturally humidified. The need for humidification especially emerges when, for example, the air moisture in living spaces drops below the comfort range through space heating. In the summer season again, with a predicted calm and cool night weather state, the system can schedule a natural cooling process this way reducing the need for energy-intensive artificial cooling of the building. If, however, a thunderstorm or similar severe weather state is expected, windows can be shut and blinds can be put in a safe state this way preventing damage to the building or its interior. While such a damage-avoiding control could also be achieved through local sensors only, with a detailed weather forecast, processes that are scheduled to start in the near future (e.g., cooling processes) can be automatically chosen with respect to the predicted exterior state. For example, a severe weather state can prohibit the opening of windows in order to perform natural cooling. It is then in the responsibility of the control system to decide if the task of cooling is delayed to a later point in time, when for example the weather forecast again predicts a calm weather state, or another process that needs more energy like air conditioning or ventilation has to be scheduled instead.
The content of the following sections elaborating concepts and relationships in the Weather & Exterior Influences ontology can to some extent be found in [151], while the information was updated and extended reflecting the current version of the created conceptualization.

4.6.1 Related Work

In some studies about building control, weather data is already considered as predictable influence factor on operation efficiency. The authors of [40] state weather as one of the main influences on cost and energy savings in buildings. The study, which focuses on optimal control of building thermal storage, uses generated weather conditions rather than real weather data for its simulations. In [56], the authors motivate the use of weather information in predictive control. They point out the importance of real-time and forecast weather data for an energy-efficient operation of buildings. The article [270] analyzes the economic impact of introducing advanced weather forecast in energy system operations. The authors study the operational costs of buildings when changing the forecast horizon for weather prediction parameters. They compare two different numerical weather forecasting methods that are weather prediction based on historical weather data as well as a "Weather Research and Forecast" model. Concluding, they recognize that the Weather Research and Forecast model outperforms the empirical model and that costs and power losses can be reduced by accurate weather forecasts. The authors shortly sketch the application of weather forecasting in buildings for temperature set-point calculations. In [183], the use of weather predictions in model predictive control strategies is discussed. The authors point out that with the help of weather data it is possible to considerably increase energy efficiency in office buildings. Using model predictive control enhanced with weather parameters (outside air temperature, wet-bulb temperature, incoming solar radiation), an improved energy-behavior of the building compared to rule-based control is achieved. The integration into a home-automation system as well as the realization of more complex use cases are not pictured. The authors of [67] describe a weather-predicted free cooling system combined with heat storage in order to improve the energy behavior of a building. They advance an HVAC unit with forecasted weather temperature and present a case study analysis showing how an energy usage reduction in the building operation can be achieved.
While these approaches consider a few weather parameters, they all lack a comprehensive weather representation. However, with such a model of weather phenomena and forecasting it is possible to describe more sophisticated control strategies implementing more complex use cases.
Also in the field of ontologies, there exist some works related to weather forecasting, for example, [16]. The described ontology is suggested to augment the weather forecasting process and formalizes some of the information that currently only exists memorized in the minds of weather forecasters. The article only discusses the two initial steps in creating an ontology for weather forecasts, therefore not describing important parts like formalizing the knowledge. Further, the scope of the ontology is on weather forecast design and construction and not on forecast delivery and representation on the consumer side as it is needed in the smart home and smart grid domain. In [187], the authors explain their proposed mapping workflow with the help of rudimentary weather ontologies. A project that to some extent incorporates weather influences in the smart home domain is presented in [210]. In this case, a KNX weather station is accessed with a software agent and its parameters are represented in an OWL ontology. The ontology which is presented in the article however just provides a small amount of parameters, while weather forecasting and online weather services are not considered at all. Other projects like [196] that describe the representation of meteorological information as ontologies can only be found in a

much broader context which, from the treated focus, is not deemed to be suitable for the smart home domain.

4.6.2 Weather Information Retrieval

Weather information for processing in a smart home system can mainly be collected through two different sources: a local weather station or a web-based information service. While a building automation weather station like the one described in [2] retrieves its information about the current weather state mainly through sensors (e.g., temperature sensor, wind speed sensor, rain sensor), an online weather service often additionally provides a short- and long-term forecast. On the contrary, with respect to the current weather situation, data retrieved from an online service is not as precise as values retrieved from sensors. In combination, both sources are capable of giving a comprehensive view on the weather domain and can achieve an accurate picture of the current environmental state and short-term weather prediction.
Internet weather forecast services to be considered as source of weather information in an automation system need to offer some sort of application programming interface (API) to make the represented information machine-accessible. An example of such a weather service with an API is the Norwegian weather service *yr.no*[18] which provides valuable information for smart home operation in the context of a smart grid. This service and its values act as base for the development of the Weather & Exterior Influences ontology module described in the following sections.
Several reasons led to the decision to choose this particular service. Firstly, the weather service allows forecasts not only for Norway but for the whole world making flexible application in a smart home possible independent of the building location. Secondly, it provides an API which is free of charge, based on the REST[19] paradigm and responds with XML documents following an online available XSD schema description[20] which allows the simple transformation and integration of presented information into an OWL ontology. Further, the yr.no weather service is sufficiently documented (cf. [178]) with a clear version management simplifying its application and reuse as it is clear which data can be expected in certain defined elements. Naturally, any other service would be sufficient as well, while yr.no however provides most of the important exterior influence parameters needed for a weather-optimized operation of a smart home or a smart grid (cf. Table 4.12).
As replies of the online weather service are modeled as XML documents, answer set elements could again be transformed into OWL individuals through an XSLT transformation as it has already been described for building information in Section 4.2.3. However, in this case, a programmatic approach through an intermediate Java representation is preferred. There exist many benefits that arise from such a Java-based transformation (cf. [116]): one of the most important in the present case is the flexible manipulation of retrieved data. For example, the transformation of cloud cover from the percentage value retrieved from the service into a more meaningful representation as *okta*[21] values can be realized. A transformation mechanism for data retrieved

[18] http://api.yr.no/weatherapi/locationforecast/
[19] Representational State Transfer
[20] http://api.yr.no/weatherapi/locationforecast/1.8/schema
[21] Okta is a unit of measurement to describe cloud cover.

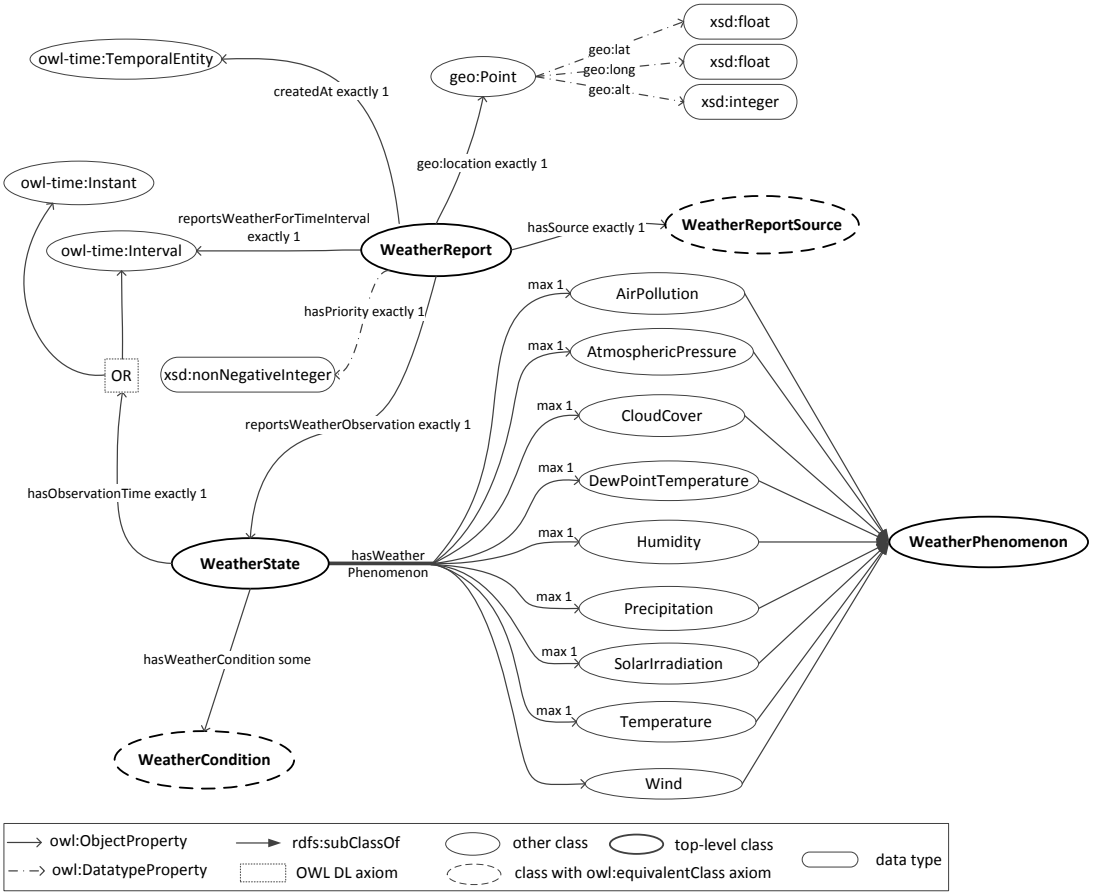

Figure 4.23: Weather ontology top-level concepts and relationships

from yr.no is presented in [228]: a "weather importer" is introduced that can retrieve weather data from yr.no and transform it into an ontology. While the resulting weather ontology in this particular thesis slightly differs from the weather ontology presented in the present work, the main concepts are similar and an adaptation of the Java import mechanism may easily be accomplished.
To discuss the value types that can be retrieved from yr.no, firstly the yr.no API is introduced. A call to the API to retrieve the weather forecast for Vienna, Austria situated at latitude $48°12'$N, longitude $16°22'$E and 177 m above see level, may look like this:

```
http://api.yr.no/weatherapi/locationforecast/1.8/?lat=48.12;lon=16.22;
msl=177
```

For a call, the weather service API as arguments needs the parameters latitude and longitude as well as an optional sea level to identify a location. It then returns a nine days weather forecast for the selected site while for each day four weather forecasts are provided. For the presented smart

home application, only the first 24 hours forecast are of interest: weather states predicted for intervals further in the future tend to become very imprecise, not suitable for optimized operation and predictive control in a smart home. Moreover, as pointed out in [190] and [222], for the application of renewable energy in a smart grid, the most relevant variations in energy supply that in any case need to be recognized and considered are those in the hourly and daily timeframe. Especially for the integration of highly unpredictable sources like wind energy, forecasts are definitely needed in order to optimize and assure the reliability of the grid. Likewise, similar conclusions can be drawn for the use of renewables in a smart home system through micro power plants (eg., solar panels, wind turbines).
An excerpt of the weather information returned from the service can be seen in Listing 4.7.

Listing 4.7: Weather forecast response from yr.no

```
<time datatype="forecast" from="2013-02-11T15:00:00Z"
                          to="2013-02-11T15:00:00Z">
  <location altitude="177" latitude="48.1200"
                         longitude="16.2200">
    <temperature id="TTT" unit="celcius" value="-2.1"/>
    <windDirection id="dd" deg="342.1" name="N"/>
    <windSpeed id="ff" mps="3.4" beaufort="2"
                                  name="Svak vind"/>
    <humidity value="84.8" unit="percent"/>
    <pressure id="pr" unit="hPa" value="1009.3"/>
    <cloudiness id="NN" percent="100.0"/>
    <fog id="FOG" percent="0.0"/>
    <lowClouds id="LOW" percent="96.1"/>
    <mediumClouds id="MEDIUM" percent="97.7"/>
    <highClouds id="HIGH" percent="97.7"/>
  </location>
</time>
<time datatype="forecast" from="2013-02-11T12:00:00Z"
                          to="2013-02-11T15:00:00Z">
  <location altitude="177" latitude="48.1200"
                         longitude="16.2200">
    <precipitation unit="mm" value="1.1"/>
    <symbol id="SNOW" number="13"/>
  </location>
</time>
```

This example shows the two main types of weather forecast available from yr.no: The first type is an *instantaneous* forecast for 15:00 with a variety of weather parameters valuable for smart home control. Sub-elements as seen in Listing 4.7, like temperature, humidity, pressure or wind speed and direction can be taken as basis for a hierarchy about weather phenomena in the proposed OWL ontology (cf. Figure 4.24). However, a special value of interest for the smart home use case not included in this forecast is precipitation for which an instantaneous observation would not be meaningful. Therefore, the second forecast element represents values that are calculated for a *time interval*, in the example for the interval from 12:00 to 15:00. Besides precipitation, for this type of forecast a symbol identifier is calculated by the yr.no service that can be seen as additional description of the weather state prevalent in this timeframe. In the designed general weather and exterior influences ontology for smart homes the two forecast types reporting weather of the same time span are treated as single weather report, incorporating information from both elements.

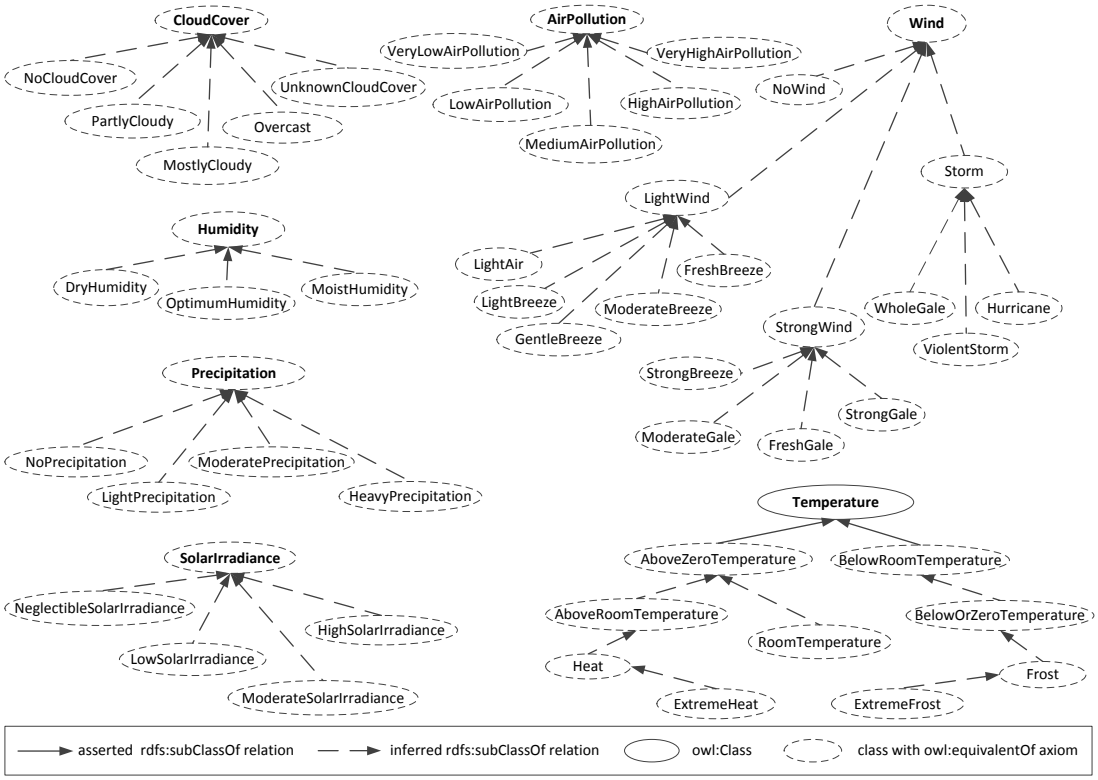

Figure 4.24: Subhierarchies of the top-level element `WeatherPhenomenon`. Only subhierarchies are shown, single subelements are not included in this overview

4.6.3 Weather Phenomena, Weather States and Weather Reports

The use cases of Chapter 3 to some extent reflect the relevance of a weather ontology to increase energy efficiency of smart home operation in general. The ontology described below can to a large part be seen as universally applicable with respect to the presented use cases and with minor changes, indicated at respective sections in the following paragraphs, may be used for application in a smart grid context as well. The modeled ontology is split into several main concepts that can be seen in Figure 4.23 and universally cover the domain of weather forecasts and reports for the application in the smart home domain. The Basic Geo (WGS84 lat/long) Vocabulary [41] is in this case used to represent the location for which a weather report is made by basically describing points by latitude, longitude and altitude. When observing weather states, further an ontological time representation is crucial. Therefore, again, the W3C time ontology recommendation[22] is reused by the proposed weather ontology in order to rely on a standard representation of time instances, intervals and series.

[22] http://www.w3.org/TR/owl-time/

WeatherPhenomenon The elements of the weather service response (cf. Listing 4.7) that are most important for the present field of application are modeled in the weather ontology as "weather phenomena" and the resulting classes can be viewed in Figure 4.23. Mostly, these classes represent values available in a typical yr.no forecast (cf. Table 4.12), sometimes however, a slightly different representation is chosen: in the case of cloudiness for example, the globally recognized meteorological unit okta is used to represent cloud cover instead of the cloudiness percentage value provided by yr.no. Further, in the case of wind, wind direction and speed as retrieved from yr.no are represented as datatype properties `hasDirection` and `hasSpeed` of a general `Wind` concept. In addition, air quality and solar irradiation are added to the weather phenomenon class: they do not have a representation in the yr.no API, however are deemed to be useful for an application in the smart home domain.
For some of the modeled phenomena, value partitions are created. The created partitions are shown in Figure 4.24 while in most of the pictured cases, the *value partition* ODP described in [197] is regarded to be suitable. The article describes two different ways to implement this pattern once with OWL individuals forming partitions and then again with OWL subclasses as value partitions. As seen in Figure 4.24 and illustrated with the "wind" concept, a further subclassification can be useful in the present case and therefore, for the weather ontology, the second variant is more appealing. This possibility of defining a more fine-grained partition hierarchy would not be available when modeling subdivisions as individuals. When creating value partitions following the ODP described in [197] several points need to be considered:

- *exhaustive partition:* The partition has to be made exhaustive, hence, all partitioning classes that may be available for a value partition in this context are mentioned explicitly in the ontology. This is realized by adding a fitting `owl:equivalentClass` axiom to the respective weather phenomenon class. For example, the following listing shows the axiom for the `Humidity` weather phenomenon:

```
owl:equivalentClass [ rdf:type owl:Class ;
                      owl:unionOf ( weo:DryHumidity
                                    weo:MoistHumidity
                                    weo:OptimumHumidity
                                  )
                    ] ;
```

  This type of axiom may also be referred to as *covering axiom* [125] and states that all possible members of a class need to be members of one or more of its subclasses.

- *disjoint subclasses:* The partitioning subclasses explicitly have to be made disjoint so that they are partitioning the value space and overlapping of classes is ruled out. In the case of the weather ontology, `owl:equivalentClass` axioms are defined for the subclasses to facilitate the automatic inference of values into one partition (cf. Section 4.6.4). Usually classes with equivalence axioms are not explicitly made disjoint as disjointness of two classes can be automatically inferred by the reasoner. In the present case, however, disjoint axioms are added to satisfy the requirements of the ODP even though the equivalence axioms are already defining a disjoint classification themselves.

- *functional object property:* Finally, according to the ODP, object properties for each of the value partitions need to be provided. Additionally, the properties are defined to be of type

owl:FunctionalProperty and have the value partition class as rdfs:range. In the weather ontology, these object properties are modeled as subproperties of the hasWeatherPhenomenon property and their rdfs:domain is WeatherState. The following listing exemplarily shows the definition of the hasHumidity object property:

```
:hasHumidity rdf:type owl:FunctionalProperty ,
                      owl:ObjectProperty ;
             rdfs:range :Humidity ;
             rdfs:domain :WeatherState ;
             rdfs:subPropertyOf :hasWeatherPhenomenon .
```

The value partition ODP is only used where it seems to be useful, hence for the weather phenomena air pollution, cloud cover, humidity, precipitation solar irradiation and wind. For the temperature concept however, no classification according to the ODP is realized, as in this particular case subclasses merely reflect weather states that are useful for the smart home domain and as such are not necessarily disjoint. Further, for the remaining weather phenomena atmospheric pressure and dew point temperature a division into subclasses would not be meaningful. As far as possible, standard units and scales are used to define limits for partitions. Air pollution is for example classified using the hourly and daily indices of the European Common Air Quality Index [243]. The wind speed categorization is accomplished by projecting the Beaufort wind force scale [218], while for a cloud cover classification the meteorological unit okta is used. For phenomena where no standard schema exists, reasonable values reflecting a location in Central Europe are used.
As can be seen in Figure 4.24 the partitioning of phenomenon values is accomplished with classes described through owl:equivalentOf axioms. This allows individual weather data acquired from a specific weather forecast being classified into the corresponding value class by the logical reasoning mechanism of OWL DL. Classification of weather phenomena plays a central role for reasoning in the created ontology and is shortly described in Section 4.6.4.

Table 4.12: WeatherPhenomenon classification in the Weather & Exterior Influences Ontology and the corresponding exterior influence parameters available in the yr.no XSD description (ver. 1.8)

weather ontology **owl:Class**	yr.no element
AirQuality	-
AtmosphericPressure	pressure
CloudCover	cloudiness
DewPointTemperature	dewpointTemperature
Humidity	humidity
Precipitation	precipitation
SolarIrradiation	-
ExteriorTemperature	temperature
Wind	windDirection & windSpeed

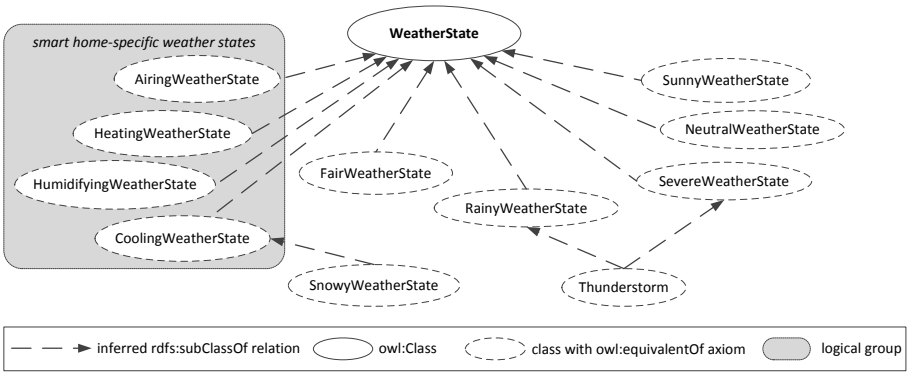

Figure 4.25: Weather states identified for the smart home system use case

WeatherCondition This auxiliary element represents a short description of a specific weather condition and in the present case contains values retrieved from yr.no. The concept involves an enumerated set of individuals in form of an `owl:oneOf` statement. In this case, a classification scheme based on individuals is used and these weather conditions may be seen as additional, descriptive information for a recognized weather state, which are not used for reasoning purposes. If another weather service is used as source of information it may provide a similar string-based mechanism to describe its weather states and in this case, a change of the individual-based classification can easily be accomplished. Possible values of the `symbol` element of the yr.no response (cf. Listing 4.7) like, for example, Snow, Sun, Fog, Rain or Thunder can therefore act as rough description of an expected near future weather state and may additionally be used to identify different weather states represented in the weather ontology.

WeatherState This hierarchy describes the actual weather type that is reported through all the information available from the source reporting it. Different weather states are considered to be either beneficial or unfavorable with respect to performing certain operations in the building and are classified in a subsumption hierarchy. Defined weather states as pictured in Figure 4.25 describe meteorological states that can affect the operation of a smart home system and may be used by a multi-agent system to optimize its operation: the classification is therefore tailored to the needs of a smart home and to enable its operating system to schedule processes with respect to the prevalent weather state that is reported for the current time or for the near future. For example, to save energy with domotic airing systems or air conditioning, the system needs to recognize a state in which the ventilation of the room is possible with outside air, which is a weather state that is not severe and has light to moderate winds, hence an `AiringWeatherState`. The modeled weather states in the ontology are therefore dependent on the domain that is to be covered (i.e., smart homes), and if the weather ontology is considered in another field of application, like for example the smart grid, it would have to be adapted to the domain and system.

WeatherReportSource This concept hierarchy holds information about the origins of weather reports. In the ontology it is mainly differentiated between two types of report sources which

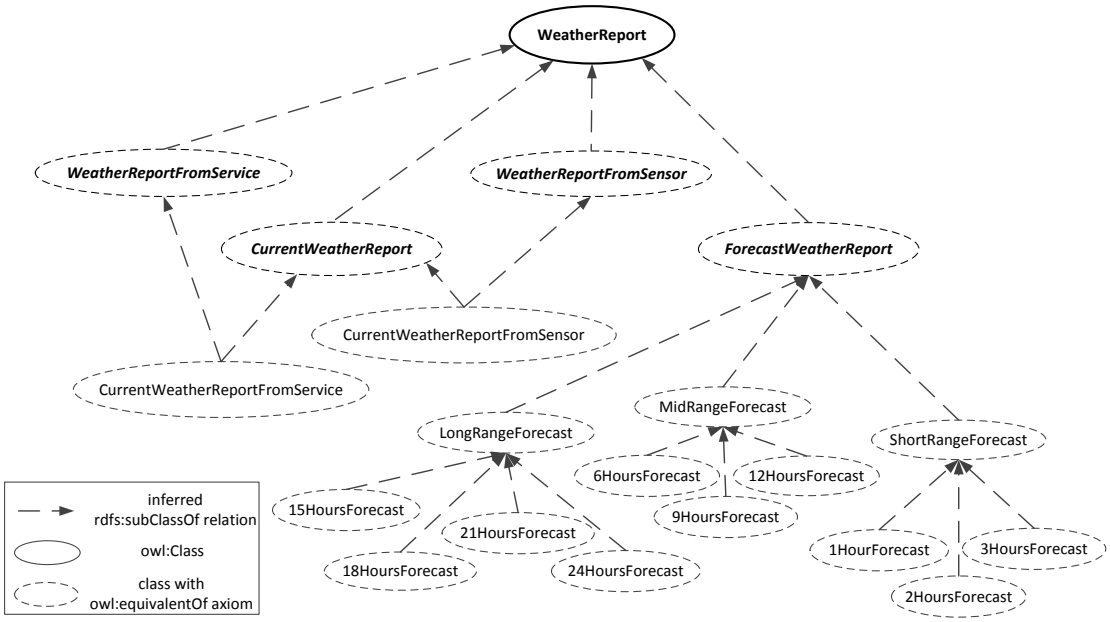

Figure 4.26: Weather report classification in the weather ontology. For brevity of the diagram only a selection of the subclasses of short-, mid- and long-range forecasts are shown exemplarily.

are `SensorReportSource` and `ServiceReportSource` denoting whether the report is coming from a local weather station and its associated sensors or an online forecasting service. In a smart home, a sensor source could be any kind of weather sensor measuring a specific outdoor parameter being integrated into a building automation system via a controller component (e.g., EIB/KNX weather station [2]). While the individual sensors and their details are described in the Energy & Resources ontology (cf. Section 4.3), the weather controller itself is represented as a `SensorReportSource` with associated sensors connecting the weather ontology to the resource ontology module (cf. Section 3.2). Envisioning the application of the ontology in a smart city use case, sensor data could be directly transferred to a grid management system from different private and public weather stations in the area.
If reports from one of these sources are not available at a certain moment in time, the system may still rely on the information from the remaining source. Further, when only a selection of parameters can be retrieved from one information source, like for example humidity and temperature from a local weather controller, this report may be complemented with values from other sources in form of a basic *sensor fusion* to generate a weather domain representation that is as complete and as correct as possible with respect to the monitored location. The `WeatherReportSource` concept therefore includes the set of different online weather services as well as local weather sensors that may be relevant to perceive exterior conditions for a specific building. This classification differentiates between reports originating either from local sensors or from web-based services, and in this respect refers to the weather report source from which it comes. In order to separate weather forecasts and weather reports, four main types

are identified (cf. Figure 4.26). First, the two concepts `WeatherReportFromService` and `WeatherReportFromSensor` separate weather reports that are published by a service interface through the Internet from reports retrieved from a local weather station observing the current weather state through sensors. The classification of these two types of weather reports is accomplished through their associated `WeatherReportSource`.
Further, while a sensor-based weather report mostly only covers current weather states, an online-service is capable of delivering forecast information. Therefore, additionally, a main differentiation between `CurrentWeatherReport` and `ForecastWeatherReport` is realized. Forecasts are classified by a property (`reportsWeatherForTimeInterval`) that is used to model a time division by specifying the offset of the time interval for which a particular weather situation is reported. This way, a hierarchy of concepts can be automatically inferred by the OWL DL reasoning mechanism (cf. Figure 4.26). Due to the accuracy of weather forecasts and their usefulness in the smart home domain, a division into different time-categories of forecasted weather reports is modeled. For the considered field of application forecasts in the range of one to five hours are deemed to be *short-range*, from six to twelve hours are categorized as *mid-range* and from 13 to 24 hours are classified as *long-range* reports. All reports for times greater than 24 hours can safely be ignored, as predictions further into the future tend to become relatively unreliable and as such are not beneficial for the projected field of application. For example, current weather reports describe the predicted weather state at the time of storing the report and as such, the reported time interval is starting with an offset of 0 hours. A 12-hours forecast then again predicts the weather for an interval starting 12 hours in the future and as such is related to an appropriate offset. Weather sensors of weather stations at the building site most likely only report current weather, while a weather forecast from a service consists of one report element that includes the current weather state while all other elements address states in the near future.
The main reason to include such a classification of weather reports in the ontology is to give the system the possibility to assess to which extent a certain weather forecast is reliable, leading to a basic notion of Quality of Service (QoS). For example, a multi-agent system may choose to mainly rely on short-range forecasts, and just taking long-range reports into consideration if a more up to date weather report is not available or if a calculation for peak electricity demand reduction is in need for long-range forecast information.

4.6.4 Reasoning in the Weather & Exterior Influences ontology

The identification of different weather states and reports is a task that can be undertaken by the reasoning mechanism of the ontology. As already seen in Figure 4.24, weather phenomena are realized as subsumption hierarchy, with the leaves of the hierarchy tree often describing a categorization of a weather phenomenon into value partitions. According to defined axioms and subclass relations, an OWL DL reasoner may perform categorizations and subsumption reasoning on this hierarchy. For example, if a relative humidity up to 25% is reported by a weather sensor or forecast, a human observer can conclude that dry weather conditions are expected. In order to also make this knowledge available to a software agent in a multi-agent-controlled smart home system the subconcept `DryHumidity` is defined accordingly with the help of an OWL DL axiom describing the phenomenon.

Listing 4.8: Weather phenomenon class `DryHumidity`

```
weo:DryHumidity rdf:type owl:Class ;
       owl:equivalentClass [ rdf:type owl:Class ;
                             owl:intersectionOf
                                 ( weo:Humidity
                                   [ rdf:type owl:Restriction ;
                                     owl:onProperty weo:hasValue ;
                                     owl:allValuesFrom [ rdf:type rdfs:Datatype ;
                                                         owl:onDatatype xsd:float ;
                                                         owl:withRestrictions
                                                             ( [ xsd:maxInclusive
                                                                 "25.0"^^xsd:float
                                                               ]
                                                             )
                                                       ]
                                   ]
                                 )
                           ] .
```

Listing 4.8 shows the definition of the concept in the designed ontology: the associated axiom classifies all humidity entries with values up to 25% as “dry” humidity.
In further consequence, value partitions identified and defined in this step act as base modules to describe weather states. For example, for a smart home operating system it can be beneficial to know in advance that if a weather state with dry humidity is imminent, humidification by outside air will not be available. This in turn leads to the need of utilizing air humidifiers and thus a higher energy expenditure. In this case, if the room occupancy schedule permits, an intelligent system could schedule the humidifying task out of peak times, simultaneously reducing the energy expenditure of the building as well as the strain on the energy distribution grid and peak electricity demand.

Listing 4.9: Weather state class `Thunderstorm`

```
weo:Thunderstorm rdf:type owl:Class ;
        owl:equivalentClass [ rdf:type owl:Class ;
                              owl:intersectionOf
                                  ( weo:WeatherState
                                    [ rdf:type owl:Restriction ;
                                      owl:onProperty weo:hasWeatherPhenomenon ;
                                      owl:someValuesFrom weo:AboveZeroTemperature
                                    ]
                                    [ rdf:type owl:Restriction ;
                                      owl:onProperty weo:hasWeatherPhenomenon ;
                                      owl:someValuesFrom weo:HeavyPrecipitation
                                    ]
                                    [ rdf:type owl:Restriction ;
                                      owl:onProperty weo:hasWeatherPhenomenon ;
                                      owl:someValuesFrom [ rdf:type owl:Class ;
                                                           owl:unionOf ( weo:LightWind
                                                                         weo:Storm
                                                                         weo:StrongWind
                                                                       )
                                                         ]
                                    ]
                                  )
                            ] ;
        rdfs:subClassOf [ rdf:type owl:Restriction ;
                          owl:onProperty weo:hasWeatherCondition ;
                          owl:hasValue weo:Thunder
                        ] .
```

Another example for a weather phenomenon partition is air temperature. In this case it is abstained from a complete value partition following the conforming ODP in order to represent partitions that are particularly interesting for the optimization of smart home control (cf. Figure 4.24). Accordingly, a temperature higher than 25 degree Celsius can be reasoned to be a member of the `AboveRoomTemperature` concept. A weather state with a temperature in this range is therefore not suitable for natural ventilation as temperature in the room would rise to a level above the comfort zone. Those and similar classifications of weather phenomena can subsequently be used for describing weather states important for smart home control.
As an example for an axiom characterizing a specific weather state, Listing 4.9 shows the logical definition of the `Thunderstorm` element. In the present case, the previously described `WeatherCondition` element fetched from the yr.no online weather service can be seen as additional fact, a one-word description of the identified weather state, that may be inferred for weather states retrieved from sensor systems which originally do not have such an associated weather condition description. This way it becomes possible to make the weather condition short-description from yr.no also available for weather states recognized by sensors. The listing further reveals the situation that classifies as thunderstorm which is solely dependent on the recognized weather phenomena. OWL definitions as shown for the example are realized for all defined weather states, leading to an inferred subsumption hierarchy (cf. Figure 4.25). While in this case some of the modeled weather states are general in nature, others are specifically targeted at an application: the logical group shown in Figure 4.25, for example, describes weather states that may influence the scheduling of HVAC appliances and as such may be used to optimize the establishment of thermal comfort (cf. Chapter 6).

Listing 4.10: Weather report class `ShortRangeForecast`

```
weo:ShortRangeForecast rdf:type owl:Class ;
   owl:equivalentClass
      [ rdf:type owl:Class ;
        owl:intersectionOf
           ( weo:WeatherReport
             [ rdf:type owl:Restriction ;
               owl:onProperty :reportsWeatherForTimeInterval ;
               owl:someValuesFrom
                  [ rdf:type owl:Restriction ;
                    owl:onProperty time:hasBeginning ;
                    owl:someValuesFrom
                       [ rdf:type owl:Restriction ;
                         owl:onProperty time:inDateTime ;
                         owl:someValuesFrom
                            [ rdf:type owl:Restriction ;
                              owl:onProperty time:hour ;
                              owl:someValuesFrom
                                 [ rdf:type rdfs:Datatype ;
                                   owl:intersectionOf
                                   ( [ rdf:type rdfs:Datatype ;
                                       owl:onDatatype xsd:nonNegativeInteger ;
                                       owl:withRestrictions ( [ xsd:minExclusive 0 ] )
                                     ]
                                     [ rdf:type rdfs:Datatype ;
                                       owl:onDatatype xsd:nonNegativeInteger ;
                                       owl:withRestrictions ( [ xsd:maxExclusive 6 ] )
                                     ] )
                                 ]
                            ]
                       ]
                  ]
```

```
                    ]
                ]
            ]
            [ rdf:type owl:Restriction ;
              owl:onProperty :reportsWeatherForTimeInterval ;
              owl:someValuesFrom
                 [ rdf:type owl:Restriction ;
                   owl:onProperty time:hasBeginning ;
                   owl:someValuesFrom
                      [ rdf:type owl:Restriction ;
                        owl:onProperty time:inDateTime ;
                        owl:someValuesFrom
                           [ rdf:type owl:Restriction ;
                             owl:onProperty time:hour ;
                             owl:allValuesFrom
                                [ rdf:type rdfs:Datatype ;
                                  owl:intersectionOf
                                  ( [ rdf:type rdfs:Datatype ;
                                      owl:onDatatype xsd:nonNegativeInteger ;
                                      owl:withRestrictions ( [ xsd:minExclusive 0 ] )
                                    ]
                                    [ rdf:type rdfs:Datatype ;
                                      owl:onDatatype xsd:nonNegativeInteger ;
                                      owl:withRestrictions ( [ xsd:maxExclusive 6 ] )
                                  ] )
                                ]
                           ]
                      ]
                 ]
            ]
        )
    ] .
```

As a final example, Listing 4.10 displays the definition of short-range weather reports in the designed ontology. As shown, the classification into different time classes is realized via the property `reportsWeatherForTimeInterval` representing the interval for which the weather report is valid expressed as offset in hours from the time the report was created. In the example case, only reports that are prognosed between one and five hours into the future are classified as short-range forecasts.
As already mentioned, modeled weather states in the proposed ontology are tailored to the need of optimizing HVAC appliances during smart home operation. Certainly, other weather states may influence the energy-efficient operation of a smart home (e.g., a lighting weather state for consideration of daylighting), and can be added to the knowledge base as desired at a later stage. Further, also for the application of the weather ontology in a smart city context different weather states than for a smart home need to be considered. In this respect, one could think of concepts that model grid-specific use cases like weather states for values that exceed some certain wind speed or solar radiation to estimate the energy that will be produced by power plants or buildings in a specific area at a particular future point in time. At the same time however, all other parts of the weather ontology can remain unchanged, which highlights the versatile applicability of the conceptualization.

4.6.5 Ontology Metrics

Table 4.13 presents metrics of the Weather & Exterior ontology module. The weather ontology designed for the envisioned smart home system currently contains 112 classes, 24 object proper-

ties and 17 datatype properties. Most of the identified classes are representing different weather phenomena and weather states that can be seen as immediate influence parameters for smart home operation. At the same time, also current and forecasted weather reports are distinguished and basic classes for sources of weather information are defined.

Table 4.13: Metrics of the Weather & Exterior Influences ontology module

		Weather & Exterior Influences ontology
TBox	Named Classes	112
	Object Properties	24
	Datatype Properties	17
ABox	Instances	11

Additionally, 11 instances are included, representing an instance-based classification of weather conditions. The value partition ontology design pattern is used throughout the ontology, and reasoning is mainly exploited to automatically categorize phenomena and weather states according to reported values. The presented ontology metrics include definitions from the Basic Geo (WGS84 lat/long) Vocabulary (two classes, one object property and three datatype properties) as well as references to three concepts, two object and one datatype properties of the OWL-Time ontology which are needed to represent the time-based division of weather reports.

CHAPTER 5

Smart Home Ontology Evaluation & Discussion

Ontology evaluation and validation has been of general interest for a long time. As ontology creation is to some part a creative process, no real possibility exists to state that an ontology is correct or incorrect [179]. Each ontology reflects reality according to the views and knowledge of the ontologist, scientist or engineer that built the ontology, and also if two ontologies for the same domain exist chances are high that the ontologies will differentiate from each other to a substantial amount. Therefore, the quality of an ontology is mainly assured by using it in the environment or application it is designed for and observe its sufficiency to address commonly experienced situations.

The previously mentioned methodologies that serve as a foundation for the creation of ontologies in this work (cf. Chapter 3) all envision some kind of evaluation: while in [242] this phase is mentioned, its execution is not explicitly shown. In [104], an evaluation of competency questions against a formal representation of the ontology is presented. The authors of [179] otherwise only mention evaluation through usage of the ontology in applications or the discussion of its contents with experts in the field. Further, the methodology described in [83] views evaluation as associative task that should be carried out throughout the complete design and implementation process. Other works describe the formal evaluation and validation of ontologies in more detail. In [90], three main measure types are proposed to evaluate and validate an ontology, which are structural, functional and usability-profiling measures. *Structural measures* are needed to assure that a created ontology is valid with respect to the used formalism or ontology language and involves topological measurements such as depth, breadth and tangledness. At this stage, the syntax and semantics of the ontology are assessed. *Functional measures* evaluate if an ontology and its defined constructs functionally cover the domain of interest. In this respect, the specific conceptualization that the ontology represents needs to be put into context with the covered application domain. The third category *usability-profiling measures* finally evaluates annotations of the ontology and if they are usable for the intended field of application. A recent article on ontology evaluation identifies five types of evaluation that are generally needed in ontology design

which are intelligibility, fidelity, craftmanship, fitness and deployability [184]. *Intelligibility* means that all intended users can understand the created ontology which can be reached on the one hand through a self-explanatory model but which is also achieved through a rich annotation of the ontology. *Fidelity* as evaluation criteria is concerned with the correctness of the ontology with respect to the particular domain it represents. *Craftmanship* comprises the consistent use of design patterns and design choices throughout the ontology as well as the syntactical correct representation of the model in the chosen ontology formalism. *Fitness* and *deployability* evaluate requirements for the expected field of application and are used to put the ontology into context with a particular system it is designed for. Further, also in [98] a summary on how an evaluation of ontologies should be conducted is presented. The article identifies several questions that need to be answered and also states that the evaluation of ontologies should be performed against a *frame of reference* which in the case of ontologies *"can be a set of competency questions, the requirements or the real world"* [98]. In this context, the article [104] explains the importance of formalization of competency questions. According to the authors, only with formal competency questions it becomes possible to assess if a postulated requirement is adequately represented in an ontology. The authors propose the formalization of informal competency questions with the help of first-order logic. Following the methodology, the formalized questions can in further consequence be used to identify formal axioms that define constraints for concepts in the specified ontology [241].

5.1 Competency Question Formalization

Starting with the informal competency questions identified in Chapter 3, firstly a suitable formal representation needs to be identified. Focusing on OWL and Description Logic-based ontologies, several works explain and propose different types of competency question formalization. In [211], for example, informal competency questions are translated to DL queries. These queries are then submitted to the Protege DL query interface for evaluation. Recently, also an increasing number of projects arises that to some extent use the SPARQL query language to formalize competency questions and thus evaluate the designed ontologies (e.g., [82], [21], [138], [234]). SPARQL as complex and feature-rich query language for RDF can be successfully used to represent a wide range of natural language questions in a question-answering perspective [21]. The retrieval of formal SPARQL queries from natural language competency questions can be split into several steps while in [21] an automatic deduction of appropriate SPARQL queries from natural language questions is pursued. However, the described methodology is tailored to the medical domain and to specific types of competency questions. Taking the findings of this article as basis, [271] describes a more general approach to translate natural language competency questions into formal SPARQL questions roughly following the initial methodology. In [82], different levels are introduced for the formal speification of competency questions: for questions involving reasoning tasks, the Pellet reasoner is used while questions involving OWL instances are formalized as SPARQL queries. For complex questions, the proposed formalization scheme further defines algorithms with the help of the Protege OWL API. Also [174] proposes a methodology for ontology development that involves different forms of competency question formalization. In their approach, the authors propose the answering of competency questions

either by using DL classification, SWRL rules or SPARQL. As a conclusion, it can be noted that there exist many different possibilities to create a formal representation of competency questions. As the main reason for formal competency questions is to ensure that a developed ontology is expressive enough to provide answers to the posed questions, the formalization of competency questions to a certain degree depends on the appropriate query language [168]. In this work, a formalization with SPARQL queries is pursued to reflect the query language used to retrieve data from the ontology in the envisioned smart home system (cf. Section 1.2). Further, especially when querying for individuals in the ontology, SPARQL provides advanced possibilities. Because of the powerful nature of SPARQL it has to be kept in mind that the language can be used to compensate for knowledge not explicitly modeled in the ontology. In such a case, the created ontology could certainly be designed simpler, however, explicitness would be lost [174]. Therefore, when formalizing competency questions with the help of SPARQL it is necessary to apply restrictions on the use of functions in order to assure that all necessary dependencies are sufficiently covered in the ontology itself rather than in the query used to retrieve information from it.

5.1.1 Classification of Competency Questions

For the evaluation procedure, the informal competency questions as identified in Chapter 3 are again consulted and the proposed classification (cf. Section 3.2) is used as a reference. The division into low-level and high-level questions serves as a framework and different approaches are taken for these two question types.
For low-level questions, generally a formalization into SPARQL queries is conducted. Where applicable, equivalent DL queries are provided. In case DL cannot be used to exactly answer the query, an explanation is given and retrieved results are discussed. In the course of the formalization also a simple categorization of questions is performed. Competency questions are first associated with a particular ontology dimension as proposed in Section 3.2 and then further classified with respect to their focus. The *focus* is used to present the core themes of queries for single domains. Further, in case a group of identified questions have similar queries, a generalized question (GROUP) is created. This generalized question is not necessarily an already defined competency question but may be constructed to act as a framework for a set of questions. For the general query, the associated DL and SPARQL representations are discussed in detail, while for the single competency questions belonging to this group shorter elaborations for necessary adaptations are outlined. Further, the return types of the queries are presented in the discussion of the low-level questions providing a simple means to show the correctness of the answer set.
For high-level queries, it is shown how the queries can be answered using the previously defined low-level questions. A hierarchical representation is conducted and a discussion of how the answer of high-level competency questions can be supported by low-level questions is provided.
The procedure that is followed to evaluate the created ontologies against associated low-level competency questions includes the use of the following software and tools: firstly, for performing DL queries on the single ontology modules, the Protege toolkit is used. The Pellet plugin of Protege is further utilized to retrieve OWL DL inferences as well as inferences resulting from DL-safe SWRL rules. The inferred model can be directly exported to a file which is subsequently

manually imported into a triple store. As triple store, the Fuseki server [8] is used which is built on the Jena framework and uses TDB as persistence backend. Fuseki is a simple RDF store that supports SPARQL and SPARQL/Update queries. This way, the SPARQL queries representing the competency questions can be executed on this triple store to extract the answer. The benefit of using Fuseki instead of Protege for SPARQL querying is that also federated queries spanning more than one ontology are possible as the inferred models of all ontology modules can be loaded into the triple store. In Protege, this would just be possible by importing all the modules into one single model and starting the Pellet reasoner on this combined ontology resulting in an extremely high reasoning overhead.
The just defined procedure allows the evaluation of the ontology with respect to its requirements without being dependent on the actual implementation of other system parts. Further, as the competency questions are directly retrieved from the main use cases for the envisioned system, by executing an evaluation of this kind, a sufficient coverage of requirements is assured.

5.2 Low-level Competency Questions

5.2.1 Building Information

Focus: building

TH02.06: How is building X divided into spaces/rooms?

DL query:

```
Space and isBuildingElementOf value BUILDING_ID_BLDG001
```

SPARQL query:

```
PREFIX gbo: <https://www.auto.tuwien.ac.at/downloads/thinkhome/ontology/
             BuildingOntology.owl#>

SELECT ?space
WHERE
{
    ?building gbo:containsSpace ?space .
    FILTER (?building = gbo:BUILDING_ID_BLDG001)
}
```

Discussion:

The building is related to its spaces via the `containsSpace` and a space can be related to a building via the `isBuildingElementOf` object property. The presented DL as well as the SPARQL query return all spaces of the building. To gather more information about the exact building layout, the control system can issue additional queries (e.g., TH02.09). The range of the `containsSpace` property ensures that returned elements are of type `Space`.

TH02.13: What are the materials of wall X?

DL query:

```
isDefinedMaterialOf some (isDefinedLayerOf some
                          (isDefinedConstructionTypeOf value SURFACE_ID_OBJ000))
```

SPARQL query:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX gbo: <https://www.auto.tuwien.ac.at/downloads/thinkhome/ontology/
             BuildingOntology.owl#>

SELECT ?material
WHERE
{
    ?surface gbo:hasDefinedConstructionType ?construction .
    ?construction gbo:hasDefinedLayer ?layer .
    ?layer gbo:hasDefinedMaterial ?material  .
    FILTER (?surface = gbo:SURFACE_ID_OBJ000)
}
```

Discussion:

The material of a wall is defined for each layer (`hasDefinedMaterial`). Further, multiple layers can be defined for a construction type (`hasDefinedLayer`) and a construction type is associated to a particular space via `hasDefinedConstructionType`. For all of these object properties, appropriate inverse properties are defined. DL as well as SPARQL queries may be used to return all materials associated with a particular wall (i.e., surface). The returned value is assured to be of type `Material` because of the range of the `hasDefinedMaterial` object property which is inverse to `isDefinedMaterialOf` and allows the DL reasoner to make the appropriate inferences.

TH02.14: Does wall X face the exterior environment?

DL query:

```
ExtSurface
```

SPARQL query:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX gbo: <https://www.auto.tuwien.ac.at/downloads/thinkhome/ontology/
             BuildingOntology.owl#>

ASK WHERE
{
    ?surface rdf:type gbo:ExtSurface
    FILTER (?surface = gbo:SURFACE_ID_OBJ000)
}
```

Discussion:

The relationship if a wall faces the exterior environment is directly represented in the KB as `ExtSurface` class, a subclass of `Surface`. Therefore, with a DL query it may be queried for all individuals that are of type `ExtSurface`. It is then necessary to determine if the wall of interest is in this set to answer query TH02.14. A SPARQL query further can add a filter for a specific wall, thus directly answering the competency question.

TH02.24: Does wall X have window openings?

DL query:

```
Surface and (containsOpening some Window)
```

SPARQL query:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX gbo: <https://www.auto.tuwien.ac.at/downloads/thinkhome/ontology/
             BuildingOntology.owl#>

ASK
WHERE
{
    ?surface gbo:containsOpening ?opening .
    ?opening rdf:type gbo:Window .
    FILTER (?surface = gbo:SURFACE_ID_OBJ000)
}
```

Discussion:

In the Architecture & Building Physics ontology a surface is connected to a particular opening via the `containsOpening` object property. Further, subclasses are defined for the `Opening` class that make it possible to differentiate between window and door openings. Using these relationships, the DL and the SPARQL query can therefore be used to identify surfaces that contain window openings, answering TH02.24.

TH02.27: What are the dimensions of window X?

DL query:

```
Window and
(containsRectangularGeometry some ((containsWidth some
                                    (hasNativeValue some Literal)) and
(containsHeight some (hasNativeValue some Literal))))
```

SPARQL query:

```
PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX gbo: <https://www.auto.tuwien.ac.at/downloads/thinkhome/ontology/
             BuildingOntology.owl#>

SELECT ?widthVal ?heightVal
WHERE
{
    ?opening gbo:containsRectangularGeometry ?rectangle .
    ?rectangle gbo:containsWidth ?width ;
               gbo:containsHeight ?height .
    ?width gbo:hasNativeValue ?widthVal .
    ?height gbo:hasNativeValue ?heightVal .
    FILTER (?opening = gbo:OPENING_ID_OBJ001)
}
```

Discussion:

The `Window` class determines if a particular opening is a window and the `containsWidth` and `containsHeight` object properties represent its dimensions. Through a DL query it is possible to query for instances of type `Window` that contain dimension relationships (i.e., height, width). With SPARQL, through the associated object properties, the height and width of the window can be obtained from the KB, thus answering TH02.27.

Focus: room

TH02.07: What is the floor area of room X?

DL query:

```
Space and (containsArea some (hasNativeValue some Literal))
```

SPARQL query:

```
PREFIX gbo: <https://www.auto.tuwien.ac.at/downloads/thinkhome/ontology/
             BuildingOntology.owl#>

SELECT ?areaVal
WHERE
{
    ?space gbo:containsArea ?area .
    ?area gbo:hasNativeValue ?areaVal .
    FILTER (?space = gbo:SPACE_ID_ZON001)
}
```

Discussion:

A room (i.e., space) and its area are connected through the `containsArea` object property. With a DL query it is possible to query for all elements that are of type `Space` and are related to an element of type `Area` representing the floor area through the `containsArea` object property. The SPARQL query further can be used to return the area of a specified room.

TH02.08: What is the volume of room X?

DL query:

```
Space and (containsVolume some (hasNativeValue some Literal))
```

SPARQL query:

```
PREFIX gbo: <https://www.auto.tuwien.ac.at/downloads/thinkhome/ontology/
             BuildingOntology.owl#>

SELECT ?volVal
WHERE
{
    ?space gbo:containsVolume ?volume .
    ?volume gbo:hasNativeValue ?volVal .
    FILTER (?space = gbo:SPACE_ID_ZON001)
}
```

Discussion:

A room (i.e., space) and its volume are connected through the `containsVolume` object property. As before in TH02.07, with a DL query it is possible to query for all elements that are of type `Space`, this time related with an element of type `Volume` through the `containsVolume` object property. The SPARQL query can be used to return the volume of a specified room (i.e., space).

TH02.09: Which rooms are adjacent to room X?

DL query:

```
Space and (hasDefinedAdjacentSpace value SPACE_ID_ZON001)
```

SPARQL query:

```
PREFIX gbo: <https://www.auto.tuwien.ac.at/downloads/thinkhome/ontology/
             BuildingOntology.owl#>

SELECT ?adjacentSpace
WHERE
{
    ?space gbo:hasDefinedAdjacentSpace ?adjacentSpace .
    FILTER (?space = gbo:SPACE_ID_ZON001)
}
```

Discussion:

Adjacent rooms are put into relation in the KB via the `hasDefinedAdjacentSpace` object property (cf. Listing 4.3, Section 4.2.4.3). The shown DL and SPARQL queries use this object property to return the adjacent space to a particular room. The return value of the SPARQL query can be inferred to be of type `Space` through the defined range of the object property.

TH02.10: Which walls are adjacent to room X?

DL query:

```
Surface and (isDefinedAdjacentSurfaceOf value SPACE_ID_ZON001)
```

SPARQL query:

```
PREFIX gbo: <https://www.auto.tuwien.ac.at/downloads/thinkhome/ontology/
             BuildingOntology.owl#>

SELECT ?surface
WHERE
{
    ?space gbo:hasDefinedAdjacentSurface ?surface .
    FILTER (?space = gbo:SPACE_ID_ZON001)
}
```

Discussion:

Walls that are adjacent with a particular room are related its representation in the KB via the object property `hasDefinedAdjacentSurface` (cf. Listing 4.2, Section 4.2.4.3) and its inverse counterpart `isDefinedAdjacentSurfaceOf`. The DL query uses the second mentioned property to determine all walls that are adjacent to a chosen room. In the case of the SPARQL query, the range of the `hasDefinedAdjacentSurface` property facilitates the inference of the return value to be of type `Surface`.

TH02.11: What is the U-value of wall X adjacent to room Y?

DL query:

```
containsU-value some (hasNativeValue some Literal) and
(isDefinedConstructionTypeOf value SURFACE_ID_OBJ003)
```

SPARQL query:

```
PREFIX gbo: <https://www.auto.tuwien.ac.at/downloads/thinkhome/ontology/
             BuildingOntology.owl#>

SELECT ?surface ?uval
WHERE
{
    ?surface gbo:hasDefinedConstructionType ?construction.
    ?construction gbo:containsU-value ?uvalue .
    ?uvalue gbo:hasNativeValue ?uval .
    FILTER (?surface = gbo:SURFACE_ID_OBJ003)
}
```

Discussion:

It is described in competency question TH02.10 how it can be found out if wall is adjacent to a chosen room. To determine the U-value of a wall it is then necessary to query for the concept `U-value` which is related to the construction of the wall through the `containsU-value` object property. The construction type is further associated with a wall (i.e., surface) via the `isDefinedConstructionTypeOf` property. The DL query may be used to return the construction of the wall which contains the relationship to the desired U-value element. The SPARQL query can answer TH02.11 by returning the value associated with the `U-value` element through the `hasNativeValue` datatype property.

TH02.12: How thick is wall X adjacent to room Y?

DL query:

```
containsThickness some (hasNativeValue some Literal) and
(isDefinedMaterialOf some (isDefinedLayerOf some
                           (isDefinedConstructionTypeOf value SURFACE_ID_OBJ003)))
```

SPARQL query:

```
PREFIX gbo: <https://www.auto.tuwien.ac.at/downloads/thinkhome/ontology/
             BuildingOntology.owl#>

SELECT ?surface ?material ?thicknessVal
WHERE
{
    ?surface gbo:hasDefinedConstructionType ?construction.
    ?construction gbo:hasDefinedLayer ?layer .
    ?layer gbo:hasDefinedMaterial ?material .
    ?material gbo:containsThickness ?thickness .
    ?thickness gbo:hasNativeValue ?thicknessVal .
    FILTER (?surface = gbo:SURFACE_ID_OBJ003)
}
```

Discussion:

How it can be determined if a wall is adjacent to a particular room is shown in TH02.10. The thickness of a wall is represented at the material level: therefore, the knowledge base provides the object property `containsThickness` relating a specific material with its thickness. To determine the materials of a chosen wall (i.e., surface), the same query can be used as presented for competency question TH02.13. The shown DL query returns the different material elements of a wall for which a `Thickness` value is defined. To query for the actual thickness of the material a SPARQL query can be used. With the SPARQL query, the thickness of the material

in the wall is returned per material, this means that the control system needs to calculate the total thickness of the wall. Alternatively, a SPARQL query grouping the thickness values of the individual material instances may be used as shown below. Both variants answer competency question TH02.12.

SPARQL query:

```
PREFIX gbo: <https://www.auto.tuwien.ac.at/downloads/thinkhome/ontology/
             BuildingOntology.owl#>

SELECT ?surface (SUM(?thicknessVal) AS ?thicknessTotal)
WHERE
{
    ?surface gbo:hasDefinedConstructionType ?construction.
    ?construction gbo:hasDefinedLayer ?layer .
    ?layer gbo:hasDefinedMaterial ?material .
    ?material gbo:containsThickness ?thickness .
    ?thickness gbo:hasNativeValue ?thicknessVal .
    FILTER (?surface = gbo:SURFACE_ID_OBJ003)
}
GROUP BY ?surface
```

TH02.28: What is the orientation of wall X adjacent to room Y?

DL query:

```
Surface and (containsRectangularGeometry some (hasAzimuthValue some Literal))
```

SPARQL query:

```
PREFIX gbo: <https://www.auto.tuwien.ac.at/downloads/thinkhome/ontology/
             BuildingOntology.owl#>

SELECT ?azimuth
WHERE
{
    ?surface gbo:containsRectangularGeometry ?rectangle .
    ?rectangle gbo:hasAzimuthValue ?azimuth.
    FILTER (?surface = gbo:SURFACE_ID_OBJ003)
}
```

Discussion:

How it can be determined if a wall (i.e., surface) is adjacent to a particular room is shown in TH02.10. The orientation of a wall is contained in the *azimuth* value available in the Architecture & Building Physics ontology. The azimuth is defined for each rectangular geometry through the `hasAzimuthValue` datatype property, while a surface and a rectangular geometry are related via the `containsRectangularGeometry` object property. The DL query may determine surfaces that are related with an element of type `RectangularGeometry` for which an azimuth value is defined and therefore have a defined orientation. A software system can then extract the desired surface and its associated azimuth value from this list of surfaces. The SPARQL query can actually be used to return the azimuth value of the rectangular geometry as orientation of a chosen wall/surface which answers TH02.28.

5.2.2 Resource Information

Focus: room, building

TH03.03: What is the current value of illuminance sensor X?

DL query:

```
stateValueOf some (stateOf value LivingRoom1LightSensor1)
```

SPARQL query:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX ero: <https://www.auto.tuwien.ac.at/downloads/thinkhome/ontology/
             EnergyResourceOntology.owl#>

SELECT ?stateVal
 WHERE
{
    ?stateValue ero:realStateValue ?stateVal .
    ?stateValue ero:stateValueOf ?state .
    ?state ero:stateOf ?sensor .
    ?sensor rdf:type ero:LightSensor.
    FILTER (?sensor = ero:LivingRoom1LightSensor1)
}
```

Discussion:

To gather the current state of an illuminance sensor from the Energy & Resource ontology, firstly the object property `stateOf` can be used which returns the `LightIntensityState` of the sensor. Through the `stateValueOf` property further a `LightIntensityStateValue` element can be retrieved by the DL query. This state value can then be used by the software system operating on the ontology to retrieve the actual value which is connected to this element via the `realStateValue` datatype property. The given SPARQL query can directly return the value associated with the selected sensor and thus answers TH03.03.

TH03.10: Which lighting facilities are installed in room X?

DL query:

```
LightingSystemResource and (isIn some (hasAssociatedSpace value Space_ID_zon001))
```

SPARQL query:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX ero: <https://www.auto.tuwien.ac.at/downloads/thinkhome/ontology/
             EnergyResourceOntology.owl#>
PREFIX gbo: <https://www.auto.tuwien.ac.at/downloads/thinkhome/ontology/
             BuildingOntology.owl#>

SELECT ?resource
 WHERE
{
    ?resource rdf:type ero:LightingSystemResource .
    ?resource ero:isIn ?room .
    ?room ero:hasAssociatedSpace ?space .
    FILTER (?space = gbo:Space_ID_zon001)
}
```

Discussion:

Lighting facilities are represented in the Energy & Resource part of the KB in the dedicated class `LightingSystemResource` and can be associated with a particular room through the `isIn` object property. The DL and SPARQL queries are examples of how to request lighting facilities (i.e., lighting system resources) that are located in a specific room with a particular associated space thus answering TH03.10.

TH02.25: Which windows of room X can be controlled by actuators?

DL query:

```
 Room and
 (hasWall some (hasWallOpening some (Window and hasActuator some WindowActuator))) and
 (hasAssociatedSpace value SPACE_ID_ZON001)
```

SPARQL query:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX ero: <https://www.auto.tuwien.ac.at/downloads/thinkhome/ontology/
             EnergyResourceOntology.owl#>
PREFIX gbo: <https://www.auto.tuwien.ac.at/downloads/thinkhome/ontology/
             BuildingOntology.owl#>

SELECT ?window
 WHERE
{
    ?window rdf:type ero:Window ;
            ero:hasActuator ?actuator .
    ?wall ero:hasWallOpening ?window .
    ?room ero:hasWall ?wall ;
          ero:hasAssociatedSpace ?space .
    FILTER (?space = gbo:SPACE_ID_ZON001)
}
```

Discussion:

In the Energy & Resource ontology, a room is associated with a wall through the `hasWall` property. Each wall may have openings connected to a wall through the `hasWallOpening` object property. A room as defined in the Energy & Resource ontology further may have a link to its space representation in the Architecture & Building Physics ontology through the `hasAssociatedSpace` object property (cf. Section 4.1.2). The DL query returns the selected room only if it has a related wall opening that is a window which can be controlled by a window actuator. In this case, it would be possible for a control system to iterate over all windows of a room and select the ones for which the DL query produces an answer. A SPARQL interface however can use a simple SPARQL query to directly answer TH02.25 by returning the controllable windows of a room.

TH02.26: Do controllable shutters exist for room X?

DL query:

```
Shutter and (isIn some (hasAssociatedSpace value SPACE_ID_ZON001)
        and (hasActuator some Actuator))
```

SPARQL query:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX ero: <https://www.auto.tuwien.ac.at/downloads/thinkhome/ontology/
             EnergyResourceOntology.owl#>
PREFIX gbo: <https://www.auto.tuwien.ac.at/downloads/thinkhome/ontology/
             BuildingOntology.owl#>

ASK WHERE
{
    ?shutter rdf:type ero:Shutter ;
             ero:isIn ?room ;
             ero:hasActuator ?actuator .
    ?room ero:hasAssociatedSpace ?space .
    FILTER (?space = gbo:SPACE_ID_ZON001)
}
```

Discussion:

Shutters are identified as own class in the KB and can be directly associated with a particular room through the `isIn` object property. To associate a room with its space representation of the Architecture & Building Physics ontology as before, the `hasAssociatedSpace` object property is used. The DL query only returns elements of type `Shutter` that also have a related actuator (i.e., can be controlled by the system) and are located in a specific room associated with a particular space element. The SPARQL query only returns true or false, regarding if such a shutter exists in this room or not. In this respect, DL and SPARQL can be used to answer competency question TH02.26.

TH05.01: Which energy producing facilities exist in the building?

DL query:

```
EnergyProducerFacility
```

SPARQL query :

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX ero: <https://www.auto.tuwien.ac.at/downloads/thinkhome/ontology/
             EnergyResourceOntology.owl#>

SELECT ?energyProducer
WHERE
{
   ?energyProducer rdf:type ero:EnergyProducerFacility .
}
```

Discussion:

Energy producing facilities are directly represented in the Energy & Resource ontology through the `EnergyProducerFacility` class. The DL as well as the SPARQL query may use this class to answer TH05.01.

Focus: facility

TH04.10: What is the current state of facility X?

DL query:

```
hasCurrentStateValue some StateValue
```

SPARQL query:

```
PREFIX ero: <https://www.auto.tuwien.ac.at/downloads/thinkhome/ontology/
             EnergyResourceOntology.owl#>

SELECT ?curStateVal
WHERE
{
   ?facility ero:hasCurrentStateValue ?curStateVal .
   FILTER(?facility IN (ero:LivingRoom1AirCondition))
}
```

Discussion:

In the KB, an object property (`hasCurrentStateValue`) and the associated `StateValue` represent the current state of a facility. This object property relates the facility with one of its possible state values representing the current state of the facility. A DL query may return all facilities that have an associated current state value. A software system may then choose the desired facility and related state value from this list. The presented SPARQL query otherwise directly returns the current state value for a particular facility answering the competency question. The type of the returned value is defined by the domain of the `hasCurrentStateValue` object property and therefore can be inferred to be `StateValue`.

GROUP: Current perception of the environment

Questions TH01.01, TH02.16, TH02.29, TH03.07 and TH03.12 are related with perceiving the current state of a particular environmental variable in a part of the building. Therefore, the question to be answered in all of the following cases can be generalized to "**What is the current state of a sensor/actuator/facility in a room of the building?**". While generally the current state of a device may be directly retrieved through the building automation subsystem, this information is also represented in the knowledge base. The following DL and SPARQL queries can be used to answer this general question.

DL query:

```
(hasCurrentStateValue some StateValue ) and
(isIn some (hasAssociatedSpace value Space_ID_zon001))
```

SPARQL query:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX gbo: <https://www.auto.tuwien.ac.at/downloads/thinkhome/ontology/
             BuildingOntology.owl#>
PREFIX ero: <https://www.auto.tuwien.ac.at/downloads/thinkhome/ontology/
             EnergyResourceOntology.owl#>

SELECT ?curVal
WHERE
{
```

```
   ?controllable ero:hasCurrentStateValue ?curVal ;
                 rdf:type ?type ;
                 ero:isIn ?room .
   ?room ero:hasAssociatedSpace ?space .
   FILTER ((?space = gbo:SPACE_ID_ZON001) &&
           (?type IN (ero:Controllable)))
}
```

Discussion:

To perceive the environment usually the current state of all sensors and facilities in a particular part of the building (i.e., space) have to be retrieved. In the Energy & Resource ontology, the object property `hasCurrentStateValue` represents the current state value of a controllable device that is of type `StateValue`. With a DL query it is possible to return all devices that have a related current state value and are related with a particular room through the `isIn` object property that itself has an associated space value (`hasAssociatedSpace`). The return type is ensured by the domain of the object property and (`Controllable`). While by returning devices instead of state values the DL query is not exactly answering the question, a SPARQL query on the other hand is capable of directly fetching the current state of the device using the same property. The example SPARQL query in this case selects the current state value of all devices in a particular space. The return value in this case is specified through the range of the `hasCurrentStateValue` object property, which is `StateValue`. Depending if this state value is a value of a continuous or discrete state, an overlying software system can extract more information about the state through associated datatype properties or the type of the state value. In case the facility is working in discrete states, the subclass type of the returned `StateValue` element is a description of the current device state (e.g., on/off, open/close). If like in the present case a continuous state is returned (e.g., temperature sensor state), the `realStateValue` and `hasNativeUnit` datatype properties associated with the current state value provide information about the value of the retrieved state.

TH01.01: What is the current air quality value in room X?

DL query:

```
CO2Sensor and (hasCurrentStateValue some StateValue) and
(isIn some (hasAssociatedSpace value SPACE_ID_ZON001))
```

SPARQL query:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX gbo: <https://www.auto.tuwien.ac.at/downloads/thinkhome/ontology/
             BuildingOntology.owl#>
PREFIX ero: <https://www.auto.tuwien.ac.at/downloads/thinkhome/ontology/
             EnergyResourceOntology.owl#>

SELECT ?curVal
WHERE
{
   ?controllable ero:hasCurrentStateValue ?curVal ;
                 rdf:type ?type ;
                 ero:isIn ?room .
   ?room ero:hasAssociatedSpace ?space .
   FILTER ((?space = gbo:SPACE_ID_ZON001) &&
          (?type IN (ero:CO2Sensor)))
}
```

Discussion:

In a smart home, air quality may be sensed through an air quality sensor (e.g., CO2 Sensor). In the Energy & Resource ontology, a class `CO2Sensor` which is a subclass of `Controllable` is modeled to classify this type of air quality sensors. The DL query in this case is very similar to the general query presented before, while additionally restricting the elements to be of type `CO2Sensor`. This way, only sensors with a current state value that are members of this class are returned. For a SPARQL query that answers TH01.01, it is only necessary to extend the presented general query with a filter statement restricting the type of the returned sensors to `CO2Sensor`.

TH02.16: What is the current temperature value for room X?

DL query:

```
TemperatureSensor and (hasCurrentStateValue some StateValue ) and
(isIn some (hasAssociatedSpace value SPACE_ID_ZON001))
```

SPARQL query:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX gbo: <https://www.auto.tuwien.ac.at/downloads/thinkhome/ontology/
             BuildingOntology.owl#>
PREFIX ero: <https://www.auto.tuwien.ac.at/downloads/thinkhome/ontology/
             EnergyResourceOntology.owl#>

SELECT ?curVal
WHERE
{
   ?controllable ero:hasCurrentStateValue ?curVal ;
                 rdf:type ?type ;
                 ero:isIn ?room .
   ?room ero:hasAssociatedSpace ?space .
   FILTER ((?space = gbo:SPACE_ID_ZON001) &&
          (?type IN (ero:TemperatureSensor)))
}
```

Discussion:

In this case the DL query restricts the result set to elements of type `TemperatureSensor` to only retrieve sensors that are capable of perceiving the temperature. For the SPARQL query, the type filter statement needs to be adjusted to type `TemperatureSensor`.

TH02.29: What is the current occupancy value of room X?

DL query:

```
OccupancySensor and (hasCurrentStateValue some StateValue ) and
(isIn some (hasAssociatedSpace value SPACE_ID_ZON001))
```

SPARQL query:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX gbo: <https://www.auto.tuwien.ac.at/downloads/thinkhome/ontology/
             BuildingOntology.owl#>
PREFIX ero: <https://www.auto.tuwien.ac.at/downloads/thinkhome/ontology/
             EnergyResourceOntology.owl#>

SELECT ?curVal
```

```
WHERE
{
   ?controllable ero:hasCurrentStateValue ?curVal ;
                 rdf:type ?type ;
                 ero:isIn ?room .
   ?room ero:hasAssociatedSpace ?space .
   FILTER ((?space = gbo:SPACE_ID_ZON001) &&
          (?type IN (ero:OccupancySensor)))
}
```

Discussion:

In this case the DL query restricts the result set to elements of type `OccupancySensor` to only retrieve sensors that are capable of perceiving the temperature. For the SPARQL query, the type filter statement needs to be adjusted to type `OccupancySensor`.

TH03.07: What is the current position of blinds/shutters in room X?

DL query:

```
ShutterActuator and (hasCurrentStateValue some StateValue ) and
(isIn some (hasAssociatedSpace value SPACE_ID_ZON001))
```

SPARQL query:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX gbo: <https://www.auto.tuwien.ac.at/downloads/thinkhome/ontology/
             BuildingOntology.owl#>
PREFIX ero: <https://www.auto.tuwien.ac.at/downloads/thinkhome/ontology/
             EnergyResourceOntology.owl#>

SELECT ?curVal
WHERE
{
   ?controllable ero:hasCurrentStateValue ?curVal ;
                 rdf:type ?type ;
                 ero:isIn ?room .
   ?room ero:hasAssociatedSpace ?space .
   FILTER ((?space = gbo:SPACE_ID_ZON001) &&
          (?type IN (ero:ShutterActuator)))
}
```

Discussion:

In the example case, the current position of blinds and shutters are retrieved by a shutter actuator. It is assumed that there only exists one single shutter actuator per room. The DL query again restricts the result set to `ShutterActuator`. For the SPARQL query, the type filter statement needs to be adjusted to type `ShutterActuator`. In this case, more than one current state value is returned (i.e., shutter state value, slat state value, actuator state value) and the control system may choose the needed value according to its associated subtype.

TH03.08: What is the current value of artificial lighting source X?

DL query:

```
DimmerLamp and (hasCurrentStateValue some StateValue ) and
(isIn some (hasAssociatedSpace value Space_ID_zon001))
```

SPARQL query:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX gbo: <https://www.auto.tuwien.ac.at/downloads/thinkhome/ontology/
             BuildingOntology.owl#>
PREFIX ero: <https://www.auto.tuwien.ac.at/downloads/thinkhome/ontology/
             EnergyResourceOntology.owl#>

SELECT ?curVal
WHERE
{
   ?controllable ero:hasCurrentStateValue ?curVal ;
                 ero:isIn ?room .
   ?room ero:hasAssociatedSpace ?space .
   FILTER ((?space = gbo:Space_ID_zon001) &&
           (?controllable = ero:LivingRoom1DimmerLampLight2))
}
```

Discussion:

This question is slightly different to the rest in this question group, as in this case it is assumed that more than one artificial lighting sources may exist for a particular room. In the example query the current value of a particular dimmer lamp (`LivingRoom1DimmerLampLight2`) is to be retrieved. The DL query needs to restrict the result set to `DimmerLamp` to only retrieve facilities of this type. The system operating on the KB is further responsible for extracting the particular lamp and its state from the answer set. In the SPARQL query it is possible to select a concrete instance through an appropriate filter statement that restricts the answer to state values of the `LivingRoom1DimmerLampLight2` entity. Again, multiple state value types could be returned (e.g., on/off state value, dimming state value) and the system operating on the KB may choose a particular state value according to its type.

TH01.03: How can a new value for sensor X in room Y be stored?

SPARQL query:

```
PREFIX ero: <https://www.auto.tuwien.ac.at/downloads/thinkhome/ontology/
             EnergyResourceOntology.owl#>

DELETE
{
   ?curVal ero:realStateValue ?value
}
INSERT
{
   ?curVal ero:realStateValue "21"^^<http://www.w3.org/2001/XMLSchema#double>
}
WHERE
{
   ero:LivingRoom1TemperatureSensor1 ero:hasCurrentStateValue ?curVal .
   ?curVal ero:realStateValue ?value .
}
```

Discussion:

This competency question is not directly related to querying the KB however illustrates how to update information in the KB. Therefore, no DL query can be identified. The SPARQL query shows how it is possible to update information with the SPARQL update functionality [263]. In this case, firstly, the environmental parameter value that needs to be updated has to

be identified. In the example query, the current state value of the living room temperature sensor (LivingRoom1TemperatureSensor1) is updated. This current value is represented as StateValue instance in the KB, which is related with the temperature sensor via the hasCurrentStateValue object property. The actual value for the state value instance is then represented with the help of the datatype property realStateValue. As a next step, the triple relating the selected state value instance with the stale value entry is deleted and a new triple with the actually sensed temperature is inserted.

TH03.09: Which modes of operation exist for facility X?

DL query:

```
stateValueOf some (stateOf value LivingRoom1AirCondition)
```

SPARQL query :

```
PREFIX ero: <https://www.auto.tuwien.ac.at/downloads/thinkhome/ontology/
             EnergyResourceOntology.owl#>

SELECT ?stateValue
WHERE
{
   ?stateValue ero:stateValueOf ?state .
   ?state ero:stateOf ?facility .
   FILTER (?facility IN (ero:LivingRoom1AirCondition))
}
```

Discussion:

In the Energy & Resource ontology, possible operation modes are depicted through hasState as well as hasStateValue object properties and their inverse counterparts stateOf and stateValueOf (cf. Figure 4.12, Section 4.3.4.3). The DL and SPARQL queries gather the possible states of a facility through the stateValueOf property. As the property is the inverse of the hasStateValue object property its domain assures that the type of the returned instance is inferred to be of type StateValue, answering question TH03.09.

TH04.13: What control commands does facility X accept?

DL query:

```
commandOf some (ControlFunctionality and
                (functionalityOf value LivingRoom1AirCondition))
```

SPARQL query:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX ero: <https://www.auto.tuwien.ac.at/downloads/thinkhome/ontology/
             EnergyResourceOntology.owl#>

SELECT ?command
WHERE
{
   ?command ero:commandOf ?functionality .
   ?functionality ero:functionalityOf ?facility ;
                  rdf:type ero:ControlFunctionality .
   FILTER (?facility IN (ero:LivingRoom1AirCondition))
}
```

Discussion:

Facilities are related to commands via their functionalities (`hasFunctionality`) and their associated commands (`hasCommand`). As in this case the queries need to return control commands, only commands that are related to a functionality of type `ControlFunctionality` should be returned. Functionalities are connected to particular facilities through the property `functionalityOf` and the possible control commands of a facility can therefore further be retrieved through the `commandOf` property answering question TH04.13. As this object property is the inverse of the `hasCommand` property its domain assures that the type of the returned instances can be inferred to be of type `Command`.

5.2.3 Energy Information

Focus: energy demand

TH03.06: What is the maximum amount of energy consumed by facility X?

DL query:

```
energyMaxConsumedBy value GeothermalHeatPump1
```

SPARQL query:

```
PREFIX ero: <https://www.auto.tuwien.ac.at/downloads/thinkhome/ontology/
             EnergyResourceOntology.owl#>

SELECT ?maxDemand  ?demandUnt
WHERE
{
    ?facility ero:maxConsumesEnergy ?energyDemand .
    ?energyDemand ero:hasNativeValue ?maxDemand ;
                              ero:hasNativeUnit ?demandUnt .
    FILTER(?facility IN (ero:GeothermalHeatPump1))
}
```

Discussion:

The `maxConsumesEnergy` object property and its inverse `energyMaxConsumedBy` are used in the Energy & Resource ontology to describe the maximum amount of energy that an energy consumer facility consumes. The DL query and the SPARQL query use these object properties to return the maximum energy consumption of a facility and thus answer TH03.06. The return type of the queries can be inferred to be of type `EnergyDemand` through the range of the `maxConsumesEnergy` object property .

TH04.11: What is the current energy demand of facility X?

DL query:

```
energyActuallyConsumedBy value LivingRoom1Tv1
```

SPARQL query :

```
PREFIX ero: <https://www.auto.tuwien.ac.at/downloads/thinkhome/ontology/
             EnergyResourceOntology.owl#>
SELECT ?demandVal  ?demandUnt
WHERE
{
```

```
    ?actualDemand ero:hasNativeValue ?demandVal ;
                  ero:hasNativeUnit ?demandUnt .
    ?facility ero:actuallyConsumesEnergy ?actualDemand .
    FILTER(?facility IN (ero:LivingRoom1Tv1))
}
```

Discussion:

The `actuallyConsumesEnergy` object property and the appropriate inverse counterpart `energyActuallyConsumedBy` are used in the KB for describing the actual amount of energy that an energy consumer facility consumes. The DL query and the SPARQL query use these object properties to return the current energy consumption of a facility and thus answer TH04.11. The return type of the queries can be inferred to be of type `EnergyDemand` through the range of the `actuallyConsumesEnergy` object property.

TH05.14: What is the energy demand of facility X in operation state Y?

DL query:

```
isEnergyDemandOf value LivingRoom1Computer1StandbyMode
```

SPARQL query:

```
PREFIX ero: <https://www.auto.tuwien.ac.at/downloads/thinkhome/ontology/
             EnergyResourceOntology.owl#>
SELECT ?demandVal  ?demandUnt
WHERE
{
    ?facility ero:hasState ?state .
    ?state ero:hasStateValue ?stateValue .
    ?stateValue ero:hasEnergyDemand ?energyDemand .
    ?energyDemand ero:hasNativeValue ?demandVal ;
                                ero:hasNativeUnit ?demandUnt .
    FILTER((?facility IN (ero:LivingRoom1Computer1)) &&
           (?stateValue IN (ero:LivingRoom1Computer1StandbyMode)))
}
```

Discussion:

In the KB, the energy demand in a particular operation state is realized via an object property `hasEnergyDemand` (cf. Figure 4.12, Section 4.3.4.3) that connects `StateValue` entities with `EnergyDemand` entities. The DL query uses the inverse property `isEnergyDemandOf` to retrieve the energy demand for a particular operation state. If a state value always belongs to only one facility in the building, the DL query is sufficient and answers competency question TH05.14. In the case that more facilities can have the same state value assigned, another restriction needs to be applied: the shown SPARQL query additionally filters with respect to facilities to only return the particular facility in question, thus answering TH05.14 for all possible KB instantiations.

TH04.04: Which facilities are currently consuming electric energy?

DL query:

```
EnergyConsumerFacility and (actuallyConsumesEnergy some
                            ((hasNativeValue some double[>0.0])
                        and (ofEnergyType value ElectricEnergy)))
```

SPARQL query:

```
PREFIX ero: <https://www.auto.tuwien.ac.at/downloads/thinkhome/ontology/
             EnergyResourceOntology.owl#>

SELECT ?energyConsumer
WHERE
{
   ?energyConsumer ero:actuallyConsumesEnergy ?demand .
   ?demand ero:ofEnergyType ero:ELECTRICENERGY ;
                   ero:hasNativeValue ?enVal .
   FILTER (?enVal > 0.0)
}
```

Discussion:

Energy consumers are represented in the Energy & Resource ontology as members of the class `EnergyConsumerFacility`. In this case, the property `actuallyConsumesEnergy` describes the current energy consumption of a facility. The domain of this property lets a reasoner infer that the returned instances are of type `EnergyConsumerFacility`. Both queries thus return the energy consumers that show an actual energy consumption greater than zero, thus answering the question TH04.04.

TH04.14: Which facilities need permanent power supply?

DL query:

```
needsPermanentSupply value true
```

SPARQL query:

```
PREFIX ero: <https://www.auto.tuwien.ac.at/downloads/thinkhome/ontology/
             EnergyResourceOntology.owl#>
SELECT ?facility
WHERE
{
    ?facility ero:needsPermanentSupply true
}
```

Discussion:

In the Energy & Resource ontology, the query if a facility needs permanent supply is directly represented through the `needsPermanentSupply` datatype property (cf. Figure 4.12, Section 4.3.4.3). The DL and the SPARQL query make use of this datatype property by querying for facilities for which the related value is `true`, and as such are answering TH04.14. The domain of the datatype property is `EnergyConsumerFacility`, which defines the inferred return type of the queries.

Focus: energy generation/transformation

TH05.02: What type of renewable energy is used as input by energy producing facility X?

DL query:

```
EnergyProducerFacility and (hasEnergySource some Renewable)
```

SPARQL query:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX ero: <https://www.auto.tuwien.ac.at/downloads/thinkhome/ontology/
             EnergyResourceOntology.owl#>
SELECT ?energySource
WHERE
{
    ?facility ero:hasEnergySource ?energySource .
    ?energySource rdf:type ?type.
    ?type rdfs:subClassOf ero:Renewable .
    FILTER(?facility IN (ero:GeothermalHeatPump1))
}
```

Discussion:

The type of energy that is used as input for an energy producing facility is realized as object property `hasEnergySource` in the Energy & Resource ontology. Further, a classification of energy sources into renewable and non-renewable energy sources is realized (cf. Figure 4.10, Section 4.3.4.1). With a DL query it therefore becomes possible to identify and return energy producing facilities that use renewable energy as input, however the competency question cannot be directly answered. SPARQL can be used to extract the needed information from the KB: in this case, all energy sources that are connected to a particular energy producer facility through the `hasEnergySource` object property are returned. A restriction assures to only return energy sources of type `Renewable` to precisely answer TH05.02.

TH05.03: What is the maximum energy output of energy producing facility X?

DL query:

```
energyMaxProducedBy value GeothermalHeatPump1
```

SPARQL query:

```
PREFIX ero: <https://www.auto.tuwien.ac.at/downloads/thinkhome/ontology/
             EnergyResourceOntology.owl#>

SELECT ?maxSupply  ?supplyUnt
WHERE
{
    ?facility ero:maxProducesEnergy ?energySupply .
    ?energySupply ero:hasNativeValue ?maxSupply ;
                  ero:hasNativeUnit ?supplyUnt .
    FILTER(?facility IN (ero:GeothermalHeatPump1))
}
```

Discussion:

The `maxProducesEnergy` object property and its inverse `energyMaxProducedBy` are used in the KB to describe the maximum amount of energy that an energy producer facility produces. The DL query and the SPARQL query use these object properties to return the maximum energy output of a facility and answer TH05.03. The return type of the queries can be inferred to be of type `EnergySupply` through the range of the `maxProducesEnergy` object property.

TH05.04: What is the current energy output of energy producing facility X?

DL query:

```
energyActuallyProducedBy value GeothermalHeatPump1
```

SPARQL query:

```
PREFIX ero: <https://www.auto.tuwien.ac.at/downloads/thinkhome/ontology/
             EnergyResourceOntology.owl#>

SELECT ?curSupply  ?supplyUnt
WHERE
{
    ?facility ero:actuallyProducesEnergy ?energySupply .
    ?energySupply ero:hasNativeValue ?curSupply ;
                  ero:hasNativeUnit ?supplyUnt .
    FILTER(?facility IN (ero:GeothermalHeatPump1))
}
```

Discussion:

The `actuallyProducesEnergy` object property property and the appropriate inverse counterpart `energyActuallyProducedBy` are used in the KB for describing the actual amount of energy that an energy producing facility produces at a certain moment in time. The DL query and the SPARQL query use these object properties to return the current energy production of an energy producer facility and thus answer TH05.04. The return type of the queries can be inferred to be `EnergySupply` through the range of the `actuallyProducesEnergy` object property.

Focus: energy providers

TH05.16: Which external energy providers provide energy type X?

DL query:

```
ElectricEnergyProvider
```

SPARQL query:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX ero: <https://www.auto.tuwien.ac.at/downloads/thinkhome/ontology/
             EnergyResourceOntology.owl#>

SELECT ?energyProvider
WHERE
{
    ?energyProvider rdf:type ero:ElectricEnergyProvider
}
```

Discussion:

For external energy providers a class hierarchy with the root node `EnergyProvider` is modeled in the Energy & Resource ontology (cf. Figure 4.11, Section 4.3.4.2). The DL and the SPARQL query both use the `ElectricEnergyProvider` subclass to identify and return external electricity providers, thus answering the competency question TH05.16 for the energy type *electricity*.

TH06.01: What is the default energy provider for energy type X?

DL query:

```
isDefaultProviderOf value ElectricEnergy
```

SPARQL query :

```
PREFIX ero:<https://www.auto.tuwien.ac.at/downloads/thinkhome/ontology/
              EnergyResourceOntology.owl#>

SELECT ?energyProvider
WHERE
{
    ?energyProvider ero:isDefaultProviderOf ero:ElectricEnergy .
}
```

Discussion:

To identify default energy providers, the Energy & Resource ontology provides the object property `isDefaultProviderOf` and its inverse `hasDefaultProvider`. Taking the energy type `ElectricEnergy` as example, the DL query and the SPARQL query return the default energy provider for the selected energy type, thus answering competency question TH06.01. The return type of the query can be inferred to be `EnergyProvider` through the domain of the `isDefaultProviderOf` object property.

TH06.03: What energy mix is used by electricity provider X?

DL query:

```
ElectricEnergyProvider and hasEnergySource some (percentageOfEnergyMix some Literal)
```

SPARQL query:

```
PREFIX ero: <https://www.auto.tuwien.ac.at/downloads/thinkhome/ontology/
             EnergyResourceOntology.owl#>

SELECT ?energySource ?percent
WHERE
{
    ?provider ero:hasEnergySource ?energySource .
    ?energySource ero:percentageOfEnergyMix ?percent
    FILTER (?provider IN (ero:ElectricityProvider1))
}
```

Discussion:

The energy mix that is associated with an electricity provider is represented through different primary energy sources that are connected to it through the `hasEnergySource` object property and the particular percentage they are involved in the production of the electric energy represented through the `percentageOfEnergyMix` datatype property (cf. Figure 4.11, Section 4.3.4.2). A DL query to the ontology can be used to return all electric energy providers that have an associated energy source representing a particular percentage of the energy mix of this particular provider. The actual energy mix would have to be extracted by the overlying control system or a human observer. The SPARQL query otherwise may be used to directly answer TH06.03: in this case, the energy source and its percentage of the energy mix is returned. The return type of the query can be inferred by a DL reasoner using the range of the `hasEnergySource` object property, which is `EnergySource`.

TH06.04: Is electricity provider X a "green" energy provider, hence does it only provide energy from renewable energy forms?

DL query:

```
ElectricEnergyProvider and GreenEnergyProvider
```

SPARQL query :

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX ero: <https://www.auto.tuwien.ac.at/downloads/thinkhome/ontology/
             EnergyResourceOntology.owl#>

ASK
WHERE
{
    ?provider rdf:type ero:ElectricEnergyProvider ;
              rdf:type ero:GreenEnergyProvider .
    FILTER (?provider IN (ero:ElectricityProvider2))
}
```

Discussion:

Green electricity providers are directly modeled in the KB through the energy provider subclass `GreenEnergyProvider` (cf. Section 4.3.4.2). The DL and the SPARQL query therefore only need to query for entities in the ontology that are of type `ElectricEnergyProvider` and `GreenEnergyProvider` to answer competency question TH06.04.

Focus: energy tariffs

TH06.05: Which energy tariffs are offered by energy provider X?

DL query:

```
hasEnergyTariff some EnergyTariff
```

SPARQL query:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX ero: <https://www.auto.tuwien.ac.at/downloads/thinkhome/ontology/
             EnergyResourceOntology.owl#>

SELECT ?tariff
WHERE
{
    ?provider ero:hasEnergyTariff ?tariff  .
    FILTER (?provider IN (ero:ElectricityProvider2))
}
```

Discussion:

An energy tariff is related with an energy provider through the `hasEnergyTariff` object property. A DL query can be used to query for all energy providers that have at least one associated energy tariff. In a further step, it would be necessary to extract the energy tariffs by an overlying control system. The shown SPARQL query otherwise returns the specific energy tariffs related with a particular energy provider and thus answers TH06.05. The type of the returned value can be determined through the range of the `hasEnergyTariff` object property which is `EnergyTariff`.

TH06.07: What are the active times of available electric energy tariffs?

DL query:

```
EnergyTariff and
(isTariffForEnergySource some (ElectricEnergySource and (hasTimeFrame some Interval)))
```

SPARQL query:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX time: <http://www.w3.org/2006/time#>
PREFIX ero: <https://www.auto.tuwien.ac.at/downloads/thinkhome/ontology/
             EnergyResourceOntology.owl#>

SELECT ?energyTariff ?beginhour ?beginminute ?endhour ?endminute
WHERE
{
    ?energyTariff ero:isTariffForEnergySource ?energySource .
    ?energySource rdf:type ero:ElectricEnergySource ;
                  ero:hasTimeFrame ?timeFrame .
    ?timeFrame time:hasBeginning ?beginning ;
               time:hasEnd ?end.
    ?beginning time:inDateTime ?dateTimeBeginning .
    ?dateTimeBeginning time:hour ?beginhour ;
              time:minute ?beginminute .
    ?end time:inDateTime ?dateTimeEnd .
    ?dateTimeEnd time:hour ?endhour ;
              time:minute ?endminute .
}
```

Discussion:

At which times a particular energy tariff is active is represented by the `hasTimeFrame` object property that associates a tariff with a particular time interval. The DL query may be used to retrieve a list of electric energy tariffs by querying for tariffs that have an element of type `ElectricEnergySource` connected through the `isTariffForEnergySource` object property. Further, the DL query restricts the answer set to elements that also have at least one associated time frame interval (`hasTimeFrame`). As this query is only capable of returning tariffs following the specified DL pattern, the competency question is not sufficiently answered. The SPARQL query otherwise can exactly answer TH06.07 by returning begin hour and minute as well as end hour and minute of each energy tariff stored in the KB. The types of the return values can be determined through the ranges of the involved object and datatype properties.

TH06.08: What are the active months of available electric energy tariffs?

DL query:

```
EnergyTariff and (isTariffForEnergySource some (ElectricEnergySource and
                 (hasTimeFrame some
                  (hasBeginning some (inDateTime some
                                     (month some Literal))))))
```

SPARQL query:

```
PREFIX time: <http://www.w3.org/2006/time#>
PREFIX ero: <https://www.auto.tuwien.ac.at/downloads/thinkhome/ontology/
             EnergyResourceOntology.owl#>

SELECT ?energyTariff ?beginmonth ?beginday ?endmonth ?endday
WHERE
```

```
{
    ?energyTariff ero:isTariffForEnergySource ?energySource .
    ?energySource ero:hasTimeFrame ?timeFrame .
    ?timeFrame time:hasBeginning ?beginning ;
               time:hasEnd ?end.
    ?beginning time:inDateTime ?dateTimeBeginning .
    ?dateTimeBeginning time:month ?beginmonth ;
              time:day ?beginday.
    ?end time:inDateTime ?dateTimeEnd .
    ?dateTimeEnd time:month ?endmonth ;
              time:day ?endday  .
}
```

Discussion:

As the question is similar to TH06.07 again the `hasTimeFrame` object property is used to retrieve the needed information. In this case the DL query returns all energy tariffs that are tariffs for an electric energy source and for which a month is associated through the `hasTimeFrame` object property. While this does not answer question TH06.08, a SPARQL query can be used to retrieve the desired answer from the KB by extracting begin and end month of an energy tariff's active interval.

TH06.06: What is the monetary cost of energy tariff X?

DL query:

```
EnergyTariff and hasMonetaryEnergyCost some EnergyCost
```

SPARQL query:

```
PREFIX ero: <https://www.auto.tuwien.ac.at/downloads/thinkhome/ontology/
             EnergyResourceOntology.owl#>

SELECT ?costVal ?costUnit ?relVal ?relUnit
WHERE
{
    ?tariff ero:hasMonetaryEnergyCost ?cost .
    ?cost ero:hasNativeValue ?costVal ;
          ero:hasMonetaryUnit ?costUnit ;
          ero:hasRelatedValue ?relVal ;
          ero:hasRelatedUnit ?relUnit ;
    FILTER(?tariff IN (ero:ElectricityProvider2DayTimeTariff))
}
```

Discussion:

An energy tariff is associated with its monetary cost through the `hasMonetaryEnergyCost` object property. As such, a DL query can be used to retrieve all energy tariffs that have an associated monetary energy cost, which however is not directly answering the competency question. To actually query for the energy cost of a particular tariff and thus answer competency question TH06.06, SPARQL can be used: in this case, the `hasNativeValue` and `hasCostUnit` datatype properties describe the price and its unit as charged per energy supply unit. The `hasRelatedValue` and `hasRelatedUnit` properties further depict for which amount of energy the price is charged (cf. Figure 4.11, Section 4.3.4.2).

TH05.18: Which feedback tariffs exist in the case that excess energy is produced?

DL query:

```
EnergyRecoveryTariff
```

SPARQL query:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX ero: <https://www.auto.tuwien.ac.at/downloads/thinkhome/ontology/
             EnergyResourceOntology.owl#>

SELECT ?tariff ?gainVal ?gainUnit ?relVal ?relUnit
WHERE
{
    ?tariff rdf:type ero:EnergyRecoveryTariff;
            ero:hasGainValue ?gain .
    ?gain ero:hasNativeValue ?gainVal ;
             ero:hasMonetaryUnit ?gainUnit ;
             ero:hasRelatedValue ?relVal ;
             ero:hasRelatedUnit ?relUnit ;
}
```

Discussion:

Feedback tariffs are identified as `EnergyRecoveryTariffs` in the ontology. Therefore, to answer the competency question, the DL query simply has to return all instances of this class. The shown SPARQL query additionally to the tariff returns gain value and monetary unit as well as related value and unit.

5.2.4 Building Processes Information

TH07.08: Which HVAC processes can be started in room X?

DL query:

```
(HeatingProcess or CoolingProcess or AiringProcess or HumidificationProcess) and
(hasProcessDomain value SPACE_ID_ZON001)
```

SPARQL query:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX gbo:<https://www.auto.tuwien.ac.at/downloads/thinkhome/ontology/
            BuildingOntology.owl#>
PREFIX ppo:<https://www.auto.tuwien.ac.at/downloads/thinkhome/ontology/
            ProcessOntology.owl#>

SELECT DISTINCT ?process
WHERE
{
    ?process rdf:type ?procType ;
             ppo:hasProcessDomain ?procDomain .
    FILTER ((?procType IN (ppo:HeatingProcess,ppo:AiringProcess,
                           ppo:CoolingProcess,ppo:HumidificationProcess)) &&
            (?procDomain IN (gbo:SPACE_ID_ZON001)))
}
```

Discussion:

For different types of HVAC processes a classification hierarchy exists in the User Behavior & Building Processes ontology. This hierarchy can be used to identify the type of processes for the HVAC domain which is one of `HeatingProcess`, `CoolingProcess`, `AiringProcess`

or HumidificationProcess. The DL query uses these classes together with the object property hasProcessDomain that associates a process to a particular room in order to answer the competency question. The SPARQL query uses the same constructs to answer the query. Duplicates, that arise from the fact that one process can have more than one type are eliminated by using the SPARQL "distinct" keyword.

TH02.17: Which environmental parameters are influenced by process X?

DL query:

```
parameterValueInfluencedBy value LR1ProcessOpenWindow
```

SPARQL query :

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX ppo: <https://www.auto.tuwien.ac.at/downloads/thinkhome/ontology/
             ProcessOntology.owl#>

SELECT ?parameterValue
WHERE
{
   ?process ppo:influencesParameterValue ?parameterValue .
   ?parameterValue rdf:type ppo:CurrentSituationParameter .
   FILTER(?process IN (ppo:LR1ProcessOpenWindow))
}
```

Discussion:

The object property influencesParameterValue is used to connect processes with the environmental parameters that are influenced by it. In this case, the inverse to this object property (parameterValueInfluencedBy) can be used by the DL and SPARQL queries to fetch environmental parameters that are influenced by a particular process, answering competency question TH02.17. The range of influencesParameterValue allows the reasoner to infer that the returned values are of type CurrentSituationParameter.

TH07.07: Which processes can influence parameter X in room Y?

DL query:

```
(influencesParameterValue some TemperatureParameter) and
(hasProcessDomain value Space_ID_zon001)
```

SPARQL query :

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX gbo: <https://www.auto.tuwien.ac.at/downloads/thinkhome/ontology/
             BuildingOntology.owl#>
PREFIX ppo: <https://www.auto.tuwien.ac.at/downloads/thinkhome/ontology/
             ProcessOntology.owl#>

SELECT ?process
WHERE
{
   ?process ppo:influencesParameterValue ?parameterValue ;
            ppo:hasProcessDomain ?procDomain .
   ?parameterValue rdf:type ?parameterType.
   FILTER((?parameterType IN (ppo:TemperatureParameter)) &&
          (?procDomain IN (gbo:Space_ID_zon001)))
}
```

Discussion:

The queries formalizing TH07.07 again use the property `influencesParameterValue` connecting processes with the environmental parameters that are influenced by it (cf. TH02.17). The parameter can be used by the DL and SPARQL queries to fetch processes that influence particular parameters. The query for processes that influence room temperature is exemplified: the `hasProcessDomain` object property is used to limit the result set to processes defined for a particular room (`Space_ID_zon001`). Together, the two object properties can be used in the DL as well as the SPARQL query to answer the competency question. The domain of `influencesParameterValue` can be used to infer that the returned values are of type `Process`.

GROUP: environmental influence of a process

Questions TH01.09, TH02.18, TH02.19 and TH03.04 are related with how a process can influence the environment. Therefore, the question to be answered in all of the following cases can be generalized to "**Which processes can increase/decrease parameters in a part of the building?**". The DL and SPARQL queries presented hereinafter can be used to answer this general question and act as a framework for the questions in this group.

DL query:

```
(increasesParameterValue some ProcessParameter) and (hasProcessDomain some Thing)
```

SPARQL query:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX gbo: <https://www.auto.tuwien.ac.at/downloads/thinkhome/ontology/
             BuildingOntology.owl#>
PREFIX ppo: <https://www.auto.tuwien.ac.at/downloads/thinkhome/ontology/
             ProcessOntology.owl#>

SELECT ?process
WHERE
{
   ?process ppo:increasesParameterValue ?parameterValue ;
            ppo:hasProcessDomain ?procDomain .
   FILTER((?procDomain IN (gbo:Space_ID_zon001)))
}
```

Discussion:

How a process influences a particular current parameter can be modeled using the object property `influencesParameterValue` and its subproperties `increasesParameterValue` or `decreasesParameterValue` (cf. Figure 4.20, Section 4.4.5). The subproperties can in this case be used to model the direction in which a process changes a particular parameter. Further, the `hasProcessDomain` object property may be used used to limit the result set to processes that for example can be started for a particular room. In this generalized case, the DL and the SPARQL query are defined in a way to return all processes that exist in the KB and may be used to increase some parameter. A query for decreasing a parameter can be defined accordingly. The type of the returned values may be inferred by a reasoner through the domain of the two involved object properties which in both cases is `Process` or a subtype thereof. The query

is purposely very general in nature to be able to act as a framework query for the following competency questions[1].

TH01.09: Which processes can improve the air quality in room X?

DL query:

```
Process and (increasesParameterValue some AirQualityParameter) and
(hasProcessDomain value Space_ID_zon001)
```

SPARQL query:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX gbo: <https://www.auto.tuwien.ac.at/downloads/thinkhome/ontology/
             BuildingOntology.owl#>
PREFIX ppo: <https://www.auto.tuwien.ac.at/downloads/thinkhome/ontology/
             ProcessOntology.owl#>

SELECT ?process
WHERE
{
   ?process ppo:increasesParameterValue ?parameterValue ;
            ppo:hasProcessDomain ?procDomain .
   ?parameterValue rdf:type ?type.
   FILTER((?procDomain IN (gbo:Space_ID_zon001)) &&
          (?type IN (ppo:AirQualityParameter)))
}
```

Discussion:

The example queries that formalize this competency question request the returned process elements to have at least one `AirQualityParameter` element which is connected via the object property `increasesParameterValue` to fetch the processes that increase the air quality. Further, the `hasProcessDomain` object property is used to limit the result set to processes that are available (i.e., can be started) for a particular room (i.e., `Space_ID_zon001`) which answers TH01.09.

TH02.18: Which processes can be used to increase/decrease the temperature of room X?

DL query:

```
Process and (decreasesParameterValue some TemperatureParameter) and
(hasProcessDomain value Space_ID_zon001)
```

SPARQL query:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX gbo: <https://www.auto.tuwien.ac.at/downloads/thinkhome/ontology/
             BuildingOntology.owl#>
PREFIX ppo: <https://www.auto.tuwien.ac.at/downloads/thinkhome/ontology/
             ProcessOntology.owl#>

SELECT ?process
WHERE
{
   ?process ppo:decreasesParameterValue ?parameterValue ;
```

[1] In the queries that belong to this group always only one example is given, either returning processes that increase a parameter or decrease a parameter. The opposing query may always be conducted equally by using the appropriate object property.

```
            ppo:hasProcessDomain ?procDomain .
   ?parameterValue rdf:type ?type.
   FILTER((?procDomain IN (gbo:Space_ID_zon001)) &&
          (?type IN (ppo:TemperatureParameter)))
}
```

Discussion:

The example queries request the returned processes to have a `TemperatureParameter` element connected via `decreasesParameterValue` to fetch the processes that can be used for cooling. Further, the `hasProcessDomain` object property is used to limit the result set to processes that can be started for a particular room (i.e., `Space_ID_zon001`) which answers TH02.18.

TH02.19: Which processes can be used to increase/decrease the relative humidity of room X?

DL query:

```
Process and (decreasesParameterValue some HumidityParameter) and
(hasProcessDomain value Space_ID_zon001)
```

SPARQL query:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX gbo: <https://www.auto.tuwien.ac.at/downloads/thinkhome/ontology/
             BuildingOntology.owl#>
PREFIX ppo: <https://www.auto.tuwien.ac.at/downloads/thinkhome/ontology/
             ProcessOntology.owl#>

SELECT ?process
WHERE
{
   ?process ppo:decreasesParameterValue ?parameterValue ;
            ppo:hasProcessDomain ?procDomain .
   ?parameterValue rdf:type ?type.
   FILTER((?procDomain IN (gbo:Space_ID_zon001)) && (?type IN (ppo:HumidityParameter)))
}
```

Discussion:

The example queries request the returned processes to have at least one `HumidityParameter` element connected via `decreasesParameterValue` to fetch the processes that can be used to decrease humidity. Further, the `hasProcessDomain` object property is used to limit the result set to processes that can be started for a particular room (i.e., `Space_ID_zon001`) which answers TH02.19.

TH03.04: Which processes can be used to increase/decrease lighting of room X?

DL query:

```
Process and (decreasesParameterValue some LightingParameter) and
(hasProcessDomain value Space_ID_zon001)
```

SPARQL query:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX gbo: <https://www.auto.tuwien.ac.at/downloads/thinkhome/ontology/
             BuildingOntology.owl#>
```

```
PREFIX ppo: <https://www.auto.tuwien.ac.at/downloads/thinkhome/ontology/
             ProcessOntology.owl#>

SELECT ?process
WHERE
{
   ?process ppo:decreasesParameterValue ?parameterValue ;
            ppo:hasProcessDomain ?procDomain .
   ?parameterValue rdf:type ?type.
   FILTER((?procDomain IN (gbo:Space_ID_zon001)) && (?type IN (ppo:LightingParameter)))
}
```

Discussion:

The example queries return processes that have at least one `LightingParameter` element connected via `decreasesParameterValue` to fetch the specific processes that can be used to decrease light. Further, the `hasProcessDomain` object property is used to limit the result set to processes that can be started for a particular room (i.e., `Space_ID_zon001`) which answers TH03.04.

TH01.10: Which facilities are involved with process X?

DL query:

```
hasAssociatedDevice some Thing
```

SPARQL query:

```
PREFIX ppo: <https://www.auto.tuwien.ac.at/downloads/thinkhome/ontology/
             ProcessOntology.owl#>

SELECT ?facility
WHERE
{
   ?process ppo:hasAssociatedDevice ?facility.
   FILTER(?process IN (ppo:LR1ProcessCloseWindow))
}
```

Discussion:

In the KB, facilities are related to processes through the `hasAssociatedDevice` object property. A DL query in this case may return all processes that have an associated device by querying for this object property which however does not answer the posed question. The SPARQL query otherwise can be used to directly answer TH01.10 by returning all facilities that are associated with a particular process through the `hasAssociatedDevice` object property. For the sake of simplicity, the return type remains unrestricted as classification of facilities is not realized in this particular ontology.

TH01.11: Is process X dependent on the weather situation?

DL query:

```
dependsOnExteriorState value true
```

SPARQL query:

```
PREFIX ppo: <https://www.auto.tuwien.ac.at/downloads/thinkhome/ontology/
             ProcessOntology.owl#>
```

```
ASK
WHERE
{
   ?process  ppo:dependsOnExteriorState true .
   FILTER(?process in (ppo:LR1ProcessOpenWindow))
}
```

Discussion:

If a process is dependent on the weather situation it is modeled in the User Behavior & Building Processes ontology through the `dependsOnExteriorState` datatype property. If the value of this property is *true*, the process is dependent on the weather situation, otherwise it is not. This property is in further consequence used by the DL and the SPARQL queries to answer the competency question. The DL query returns all processes that depend on the weather situation. The overlying control system has to filter the interesting process out of this list of processes. Alternatively a concrete process can be stated in the DL query as well, having it returning the process or no answer if it is not dependent on the weather situation. The SPARQL query returns true or false either if the process is dependent on the weather or not. The return type of the DL query can be inferred through the domain of the property which is `Process`.

5.2.5 User Behavior Information

Focus: room

TH07.03: What is the predicted value for all different parameters in room X in the next few hours?

DL query:

```
(isScheduledValueOf some
 (MondayPattern and (hasPatternFocusObjective value Space_ID_zon001) and
(validFor value TheMonthFebruary))) and
(hasActiveTime some (hasBeginning some
                     (inDateTime some ((hour some int[>13]) and
                                       (hour some int[<=17]) ))))
```

SPARQL query:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX time: <http://www.w3.org/2006/time#>
PREFIX ppo: <https://www.auto.tuwien.ac.at/downloads/thinkhome/ontology/
             ProcessOntology.owl#>
SELECT ?scheduleVal ?scheduledHour ?value
WHERE
{
   ?pattern rdf:type ?type ;
            ppo:validFor ?patValidity .
   ?scheduleVal ppo:isScheduledValueOf ?pattern ;
                ppo:hasValue ?value ;
                ppo:hasActiveTime ?scheduleTime .
   ?scheduleTime time:hasBeginning ?beginning .
   ?beginning time:inDateTime ?dateTime .
   ?dateTime time:hour ?scheduledHour .
   BIND (now() AS ?currentTime)
   BIND (hours(?currentTime) AS ?currentHour)
   FILTER ((?scheduledHour >= ?currentHour &&
            ?scheduledHour <= ?currentHour+4) &&
           (?patValidity IN (ppo:TheMonthFebruary)) &&
           (?type IN (ppo:MondayPattern))) }
```

Discussion:

To describe predicted values for facilities, parameters and spaces the class `Pattern` with its associated object property `hasScheduledValue` and its inverse `isScheduledValueOf` are provided (cf. Section 4.4.4). The object property `hasActiveTime` can be used to describe the time for which a parameter value is scheduled. To model time-based facts, relationships and classifications of the OWL-Time ontology are reused. To give an example, in the query the next 4 hours are used to represent the phrase "next few hours" from the competency question. To formulate a DL query capable of answering this competency question, beginning as well as end time need to be specified as parameters. The beginning hour in this case needs to correspond to the hour of the current time, in the example 13 and the end hour is fixed to 17. With these parameters specified, the DL query is capable of answering the competency question limiting the result to patterns defined for the four following hours. With SPARQL, the built-in function *now()* can be used to determine the current time. It is then possible to use the same relationships as presented in the DL query to limit the answer set to scheduled values in the next four hours. In the DL as well as the SPARQL query returned patterns are limited to be of type `MondayPattern` and having a `validFor` relation to `TheMonthFebruary`, assuming that the query is issued on a Monday in February.

TH07.05: What is the predicted occupancy schedule for room X in the next few hours?

DL query:

```
(isScheduledValueOf some (OccupancyPattern and MondayPattern and
                          (hasPatternFocusObjective value Space_ID_zon001) and
                          (validFor value TheMonthFebruary))) and
(hasActiveTime some (hasBeginning some (inDateTime some ((hour some int[>13]) and
                                                         (hour some int[<=17]) ))))
```

SPARQL query :

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX time: <http://www.w3.org/2006/time#>
PREFIX gbo: <https://www.auto.tuwien.ac.at/downloads/thinkhome/ontology/
             BuildingOntology.owl#>
PREFIX ppo: <https://www.auto.tuwien.ac.at/downloads/thinkhome/ontology/
             ProcessOntology.owl#>

SELECT ?scheduleVal ?scheduledHour ?value
WHERE
{
   ?pattern rdf:type ?type1;
            rdf:type ?type2;
            ppo:validFor ?patValidity ;
            ppo:hasPatternFocusObjective ?space .
   ?scheduleVal ppo:isScheduledValueOf ?pattern ;
                ppo:hasValue ?value ;
                ppo:hasActiveTime ?scheduleTime .
   ?scheduleTime time:hasBeginning ?beginning .
   ?beginning time:inDateTime ?dateTime .
   ?dateTime time:hour ?scheduledHour .
   BIND (now() AS ?currentTime)
   BIND (hours(?currentTime) AS ?currentHour)
   FILTER ((?scheduledHour > ?currentHour && ?scheduledHour <= ?currentHour+4) &&
           (?space IN (gbo:Space_ID_zon001)) &&
           (?patValidity IN (ppo:TheMonthFebruary)) &&
           (?type1 IN (ppo:MondayPattern)) &&
```

```
          (?type2 IN (ppo:OccupancyPattern)))
}

ORDER BY ?scheduledHour
```

Discussion:

This competency question is a specialized version of TH07.03. Therefore, to describe predicted occupancy values for a particular room, a similar approach as in TH07.03 is followed. In the DL query, the `isScheduledValueOf` object property is used to get all values that are scheduled for a particular schedule (i.e., pattern). The returned patterns are limited on the one hand by querying for the type `OccupancyPattern` which is a subclass of the pattern hierarchy and also limiting the focus objective to one particular room, in the example case to `Space_ID_zon001`. The object property `hasActiveTime` is again used to limit the returned instances to a four-hours time window (cf. TH07.03 for details). With a particular start time and end time specified, the DL query is capable of answering the competency question. Also with the SPARQL query, the process of limiting the set to values of the next few hours is similar to TH07.03 and can be reviewed in detail there. Returned patterns are again limited to be of type `MondayPattern` and having a `validFor` relation to `TheMonthFebruary`, assuming that the query is issued on a Monday in February.

TH03.12: Which automatically created visual comfort schedules exist for room X?

DL query:

```
LightLevelPattern and (hasPatternFocusObjective value SPACE_ID_ZON001)
```

SPARQL query:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX gbo: <https://www.auto.tuwien.ac.at/downloads/thinkhome/ontology/
             BuildingOntology.owl#>
PREFIX ppo: <https://www.auto.tuwien.ac.at/downloads/thinkhome/ontology/
             ProcessOntology.owl#>

SELECT ?pattern
WHERE
{
   ?pattern rdf:type ?patternType ;
            ppo:hasPatternFocusObjective ?space .
   FILTER ((?patternType IN (ppo:LightLevelPattern)) &&
           (?space IN (gbo:SPACE_ID_ZON001)))
}
```

Discussion:

As already mentioned, automatically created schedules are stored in an own hierarchy starting at the `Pattern` node (cf. Section 4.4.4). To answer this competency question the DL and the SPARQL queries make use of the hierarchy to identify schedules for visual comfort as entities of type `LightLevelPattern`, a subclass of `Pattern`. Further, the queries use the `hasPatternFocusObjective` to only select schedules that have a relation to a particular part of the building (i.e., room/space). The final queries therefore return visual comfort schedules that exist for a particular room. The control system is in this case responsible to select schedules according to their time and values.

TH04.02: Which automatically generated occupancy schedule exists for room X?

DL query:

```
OccupancyPattern and (hasPatternFocusObjective value SPACE_ID_ZON001)
```

SPARQL query :

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX gbo: <https://www.auto.tuwien.ac.at/downloads/thinkhome/ontology/
             BuildingOntology.owl#>
PREFIX ppo: <https://www.auto.tuwien.ac.at/downloads/thinkhome/ontology/
             ProcessOntology.owl#>

SELECT ?pattern
WHERE
{
   ?pattern rdf:type ppo:OccupancyPattern ;
            ppo:hasPatternFocusObjective ?space .
   FILTER(?space IN (gbo:SPACE_ID_ZON001))
}
```

Discussion:

This competency question is similar to question TH03.12. Again, the hierarchy for automatically created schedules is used to identify occupancy schedules of type `OccupancyPattern` which is a subclass of `Pattern`. The DL and the SPARQL queries to answer TH04.02 thus make use of this subclass to select occupancy schedules and further also use the object property `hasPatternFocusObjective` to reduce the returned schedules to the ones that have a relation to a particular room (i.e., space) in the building.

Focus: facility

TH07.02: Which automatically defined facility schedules exist for room X and day Y?

DL query 1 (User Behavior & Building Processes):

```
MondayPattern and (validFor value THEMONTHFEBRUARY) and
(hasPatternFocusObjective some Thing)
```

DL query 2 (Energy & Resource):

```
Controllable and (isIn some (hasAssociatedSpace value SPACE_ID_ZON001))
```

SPARQL query (federated):

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX gbo: <https://www.auto.tuwien.ac.at/downloads/thinkhome/ontology/
             BuildingOntology.owl#>
PREFIX ero: <https://www.auto.tuwien.ac.at/downloads/thinkhome/ontology/
             EnergyResourceOntology.owl#>
PREFIX ppo: <https://www.auto.tuwien.ac.at/downloads/thinkhome/ontology/
             ProcessOntology.owl#>

SELECT ?pattern
WHERE
{
   ?pattern rdf:type ?patternType ;
            ppo:validFor ?patValidity ;
            ppo:hasPatternFocusObjective ?facility .
   ?facility rdf:type ?type;
             ero:isIn ?room .
```

```
   ?room ero:hasAssociatedSpace ?space .
   FILTER ((?patternType IN (ppo:MondayPattern)) &&
           (?patValidity IN (ppo:TheMonthFebruary)) &&
           (?type IN (ero:Controllable)) &&
           (?space IN (gbo:Space_ID_zon001)))
}
```

Discussion:

To retrieve the automatically defined facility schedules for a particular room two different ontologies need to be queries, the User Behavior & Building Processes and the Energy & Resource ontology. Firstly, to gather all automatically defined schedules for a particular day a subclass of `Pattern` (e.g., `MondayPattern`) in combination with the `validFor` object property can be used as already shown in TH07.03. In order to retrieve *facility* schedules, the focus objective associated with this pattern through the (`hasPatternFocusObjective`) object property has to be a facility that is installed in a particular room. It is not directly possible to model this relation in a single DL query as the requested information is available in another ontology, the Energy & Resource ontology. Therefore, in the first query, the type of the element connected through `hasPatternFocusObjective` stays unrestricted. A DL query to the second ontology then may extract all facilities that exist in a particular room. The class `Controllable` is defined in the Energy & Resource ontology as a superclass for a hierarchy classifying controllable building equipment. To answer competency question TH07.02 only with DL queries, it is necessary to match the element connected to patterns through the `hasPatternFocusObjective` properties in DL query one with the controllable facilities of the particular room retrieved through DL query two. As the required facts are stored in different ontologies, they either have to be joined by importing one ontology into the other, or the matching needs to happen in an overlying control program.
A SPARQL query otherwise is capable of answering competency question TH07.02 by federating the two ontologies: The query is executed on two ontology graphs stored in one triple store and returns only patterns (i.e., schedules) that are defined for facilities located in a particular room. In this case, a SPARQL filter assures that only facilities in `Space_ID_zon001` are returned. Another SPARQL filter defines that the focus objective of the pattern is of type `Controllable` as defined in the Energy & Resource ontology. The return type of this query is a subtype of `Pattern` describing patterns that are valid on Mondays in February.

TH03.13: Which automatically defined facility schedules exist for facility X and day Y?

DL query:

```
MondayPattern and (validFor value TheMonthFebruary) and
(hasPatternFocusObjective value LivingRoom1DimmerLampLight2)
```

SPARQL query:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX ero: <https://www.auto.tuwien.ac.at/downloads/thinkhome/ontology/
             EnergyResourceOntology.owl#>
PREFIX ppo: <https://www.auto.tuwien.ac.at/downloads/thinkhome/ontology/
             ProcessOntology.owl#>

SELECT ?pattern
WHERE
```

```
{
   ?pattern ppo:hasPatternFocusObjective ?facility ;
            rdf:type ?patternType ;
            ppo:validFor ?patValidity .
   FILTER ((?patternType IN (ppo:MondayPattern)) &&
           (?patValidity IN (ppo:TheMonthFebruary)) &&
           (?facility IN (ero:LivingRoom1DimmerLampLight2)))
}
```

Discussion:

In the present case as example again the patterns for Mondays in February are requested. Further, this time available patterns for one specified facility are requested, hence, the range of the `hasPatternFocusObjective` object property is limited to one single entity in the KB, which in the example case is `LivingRoom1DimmerLampLight2`. DL as well as SPARQL query can successfully answer TH03.13 by returning the patterns defined for a particular facility and a defined day.

5.2.6 User Preferences Information

TH04.03: Which manually defined occupancy schedules exist for room X?

DL query:

```
PresencePreferenceSchedule and (hasPreference some (forSpace value Space_ID_zon001))
```

SPARQL query:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX apo: <https://www.auto.tuwien.ac.at/downloads/thinkhome/ontology/
             ActorOntology.owl#>
PREFIX gbo: <https://www.auto.tuwien.ac.at/downloads/thinkhome/ontology/
             BuildingOntology.owl#>

SELECT DISTINCT ?presSched
WHERE
{
    ?presSched rdf:type  ?prefType ;
               apo:hasPreference ?preference .
    ?preference apo:forSpace ?space.
    FILTER ((?prefType = apo:PresencePreferenceSchedule)  &&
            (?space = gbo:Space_ID_zon001))
}
```

Discussion:

The query returns preference schedule individuals that link to single user-defined preferences. To define preference schedules, the ontology provides a `PreferenceSchedule` hierarchy. One subconcept of this hierarchy is `PresenceSchedule`, joining all occupancy schedules manually defined by different users. To retrieve the presence schedule of a particular room (i.e., space) in the building, further, the space to which this preference can be associated to has to be determined. The connection between preference and space is in this case made through the object property `forSpace`. The SPARQL "distinct" keyword is used to return each preference schedule only once. These dependencies are sufficient to create DL and SPARQL queries that answer question TH04.03.

TH01.08: Which user preferences for air quality exist?

DL query:

AirQualityPreference and *UserDefinedPreference*

SPARQL query:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX apo: <https://www.auto.tuwien.ac.at/downloads/thinkhome/ontology/
             ActorOntology.owl#>

SELECT ?preference
WHERE
{
    ?preference rdf:type  ?prefType1 ;
                rdf:type  ?prefType2 .
    FILTER ((?prefType1 = apo:AirQualityPreference) &&
            (?prefType2 = apo:UserDefinedPreference))
}
```

Discussion:

In this ontology, preferences are stored in an own `Preference` hierarchy. This hierarchy represents single preference entries created by the user or defined through standards. The class `UserDefinedPreference` allows to distinguish a customized user preference from a standard preference. Furthermore, to identify preferences for air quality, the subconcept `AirQualityPreference` is defined. Therefore, these two classes may be used to retrieve the desired entries from the knowledge store and answer the competency question.

TH02.03: Which user preferences for comfort temperature exist?

DL query:

ComfortTemperaturePreference and *UserDefinedPreference*

SPARQL query:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX apo: <https://www.auto.tuwien.ac.at/downloads/thinkhome/ontology/
             ActorOntology.owl#>

SELECT ?preference
WHERE
{
    ?preference rdf:type  ?prefType1 ;
                rdf:type  ?prefType2 .
    FILTER ((?prefType1 = apo:ComfortTemperaturePreference) &&
            (?prefType2 = apo:UserDefinedPreference))
}
```

Discussion:

Again, the concept `UserDefinedPreference` of the `Preference` hierarchy is used to return only user-defined preferences. Further, a `TemperaturePreference` subhierarchy is provided: hereby, different temperature types can be distinguished, with one subclass being `ComfortTemperaturPreference`. An answer of the competency question TH02.03 can be accomplished by combining `ComfortTemperaturePreference` and `UserDefined-Preference`, thus, only querying for individuals being members of both classes.

TH02.04: Which user preferences for comfort humidity exist?

DL query:

```
RelativeHumidityPreference and UserDefinedPreference
```

SPARQL query:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX apo: <https://www.auto.tuwien.ac.at/downloads/thinkhome/ontology/
             ActorOntology.owl#>

SELECT ?preference
WHERE
{
    ?preference rdf:type  ?prefType1 ;
                rdf:type  ?prefType2 .
    FILTER ((?prefType1 = apo:RelativeHumidityPreference) &&
            (?prefType2 = apo:UserDefinedPreference))
}
```

Discussion:

Similar to queries TH01.08 and TH02.03, a user preference is firstly identified through the `UserDefinedPreference` subclass of the `Preference` hierarchy. To further narrow the results to humidity preferences, the `RelativeHumidityPreference` class may be used. With the help of these two classes, DL and SPARQL queries can successfully answer competency question TH02.04.

TH03.01: Which manually defined visual comfort schedules for room X exist?

DL query:

```
VisualComfortPreferenceSchedule and
(hasPreference some ( UserDefinedPreference and
                     (forSpace value SPACE_ID_ZON001)))
```

SPARQL query:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX apo: <https://www.auto.tuwien.ac.at/downloads/thinkhome/ontology/
             ActorOntology.owl#>
PREFIX gbo: <https://www.auto.tuwien.ac.at/downloads/thinkhome/ontology/
             BuildingOntology.owl#>

SELECT DISTINCT ?schedule
WHERE
{
    ?schedule rdf:type ?schedType;
              apo:hasPreference ?preference .
    ?preference rdf:type  ?prefType ;
                apo:forSpace ?space.
    FILTER ((?schedType = apo:VisualComfortPreferenceSchedule) &&
            (?prefType = apo:UserDefinedPreference) &&
            (?space = gbo:SPACE_ID_ZON001))
}
```

Discussion:

To answer this query a preference schedule is needed and thus the `PreferenceSchedule` hierarchy is used to retrieve the answer. In this hierarchy, a subclass for preference schedules describing visual comfort is provided (`VisualComfortPreferenceSchedule`). A

schedule manually defined by a human actor can further be identified through associated preferences, which need to be of type `UserDefinedPreference`. To only retrieve schedules for a particular room a `forSpace` object property restriction is added for preferences associated with the preference schedule. The SPARQL query uses the "distinct" keyword to only return the schedule once, also in case that more than one preference is associated with it. DL and SPARQL queries can this way successfully answer question TH03.01.

TH01.06: What is the minimum air quality that has to be assured according to standards?

DL query:

```
AirQualityPreference and StandardPreference
```

SPARQL query:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX apo: <https://www.auto.tuwien.ac.at/downloads/thinkhome/ontology/
             ActorOntology.owl#>

SELECT ?preference
WHERE
{
    ?preference rdf:type  ?prefType1 ;
                rdf:type  ?prefType2 .
    FILTER ((?prefType1 = apo:AirQualityPreference) &&
            (?prefType2 = apo:StandardPreference))
}
```

Discussion:

For air quality preferences, an own subconcept `AirQualityPreference` is defined in the `Preference` hierarchy. The `StandardPreference` subclass can further be used to identify if a preference represents a value retrieved through a particular standard (e.g., ASHRAE standard for indoor air quality [6]). An individual that is member of both classes therefore is an air quality that is defined and recommended by a standard answering question TH01.06. It has to be noted that in this case all standard air quality temperature preferences stored in the knowledge base are returned. If more than one individual fulfills the query, the request needs to be adapted to only return the specific individual of interest.

TH02.02: What is the standard comfort humidity?

DL query:

```
RelativeHumidityPreference and StandardPreference
```

SPARQL query:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX apo: <https://www.auto.tuwien.ac.at/downloads/thinkhome/ontology/
             ActorOntology.owl#>

SELECT ?preference
WHERE
{
    ?preference rdf:type  ?prefType1 ;
                rdf:type  ?prefType2 ;
    FILTER ((?prefType1 = apo:RelativeHumidityPreference) &&
            (?prefType2 = apo:StandardPreference))
}
```

Discussion:

To identify a standard preference defining comfort humidity, an individual is required to be member of the class `RelativeHumidityPreference` representing all stored humidity preferences as well as `StandardPreference` to recognize it as a preference advised through a particular standard. The combination of these two classes denotes the result set of the competency question.

TH02.01: What is the standard comfort temperature?

DL query:

```
ComfortTemperaturePreference and StandardPreference
```

SPARQL query:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX apo: <https://www.auto.tuwien.ac.at/downloads/thinkhome/ontology/
             ActorOntology.owl#>

SELECT ?preference
WHERE
{
    ?preference rdf:type  ?prefType1 ;
                rdf:type  ?prefType2 .
    FILTER ((?prefType1 = apo:ComfortTemperaturePreference) &&
            (?prefType2 = apo:StandardPreference))
}
```

Discussion:

As already before in TH01.06 and TH02.02, a standard preference can be identified by using the `StandardPreference` class. An individual representing a comfort temperature preference is further a member of the class `ComfortTemperaturePreference` and a combination of the two classes answers question TH02.01.

TH03.02: Which recommended preferences for lighting exist?

DL query:

```
LightingLevelPreference and StandardPreference
```

SPARQL query:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX apo: <https://www.auto.tuwien.ac.at/downloads/thinkhome/ontology/
             ActorOntology.owl#>

SELECT ?preference
WHERE
{
    ?preference rdf:type  ?prefType1 ;
                rdf:type  ?prefType2 .
    FILTER ((?prefType1 = apo:LightingLevelPreference) &&
            (?prefType2 = apo:StandardPreference))
}
```

Discussion:

Lighting preferences are identified through the `LightingLevelPreference` class. A combination with the `StandardPreference` concept can be used to answer competency question TH03.02.

TH02.30: What is the setback temperature for the building/a space/a room?

DL query:

SetbackTemperaturePreference

SPARQL query :

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX apo: <https://www.auto.tuwien.ac.at/downloads/thinkhome/ontology/
             ActorOntology.owl#>

SELECT ?preference
WHERE
{
    ?preference rdf:type  ?prefType.
    FILTER ((?prefType = apo:SetbackTemperaturePreference))
}
```

Discussion:

To identify setback temperature preferences, there exists a subclass `SetbackTemperature-Preference` in the `Preference` hierarchy. When querying for members of this class, all setback temperature preferences defined for the building are returned, thus answering question TH02.30. To restrict the results to preferences for a particular room, similar as in answering question TH03.01 the query would include the connection to the room through the `forSpace` object property.

TH07.01: Which manually defined facility schedules exist for room X?

DL query:

AppliancePreferenceSchedule and (**hasPreference** some (**forSpace** value SPACE_ID_ZON001))

SPARQL query:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX apo: <https://www.auto.tuwien.ac.at/downloads/thinkhome/ontology/
             ActorOntology.owl#>
PREFIX gbo: <https://www.auto.tuwien.ac.at/downloads/thinkhome/ontology/
             BuildingOntology.owl#>

SELECT DISTINCT ?schedule
WHERE
{
    ?schedule   rdf:type  ?schedType1 ;
                rdf:type  ?schedType2 ;
                apo:hasPreference ?preference .
    ?preference apo:forSpace ?space .
    FILTER ((?schedType1 = apo:AppliancePreferenceSchedule) &&
            (?space = gbo:SPACE_ID_ZON001))
}
```

Discussion:

For schedules that are related with a facility, the `AppliancePreferenceSchedule` class is provided as subconcept of the `PreferenceSchedule` hierarchy. To only retrieve those preference schedules created for a particular room in the building, further, a restriction through the `forSpace` object property is necessary for related `Preference` individuals. Alternatively, it would be possible to retrieve the affected space by querying for the location of the associated facility which is specified in the Energy & Resource ontology module. As the `forSpace` property is defined for each `Preference` element of the schedule, the "distinct" keyword needs to be used in the SPARQL query to only return preference schedules once, regardless of the amount of connected preferences. The presented DL and SPARQL queries are this way capable of answering competency question TH07.01.

5.2.7 Exterior Influences Information

Focus: current weather situation

GROUP: reported current weather situation

Questions TH05.06, TH01.13, TH07.13 and TH02.23 are all connected with the perception of the current weather situation. In this case, the common query to be answered can be generalized to question TH05.06, "**What is the current weather situation?**". The following DL and SPARQL queries can be used to answer this general question and may be seen as a framework for the questions in this group.

TH05.06: What is the current weather situation?

DL query

```
WeatherState and
(hasWeatherReport some
 (CurrentWeatherReport and
 (createdAt some (inDateTime some
                  ((hour value 19) and
                   (day value "---10"^^gDay) and
                   (month value "--02"^^gMonth) and
                   (year value "2013"^^gYear))))))
```

SPARQL query:

```
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX time: <http://www.w3.org/2006/time#>
PREFIX weo: <https://www.auto.tuwien.ac.at/downloads/thinkhome/ontology/
             WeatherOntology.owl#>

SELECT ?weatherState
WHERE
{
   ?weatherState rdf:type ?weatherType ;
                 weo:hasWeatherReport ?weatherReport.
   ?weatherReport rdf:type weo:CurrentWeatherReport ;
                  weo:createdAt ?wrTime .
   ?wrTime time:inDateTime ?dateTime .
   ?dateTime time:hour ?wsHour ;
             time:day ?wsDay ;
```

```
            time:month ?wsMonth ;
            time:year ?wsYear .
   BIND (now() AS ?currentTime)
   BIND (hours(?currentTime) AS ?currentHour)
   BIND (xsd:gDay(CONCAT("---",xsd:string(day(?currentTime)))) AS ?currentDay)
   BIND (xsd:gMonth(CONCAT("--",xsd:string(month(?currentTime)))) AS ?currentMonth)
   BIND (xsd:gYear(xsd:string(year(?currentTime))) AS ?currentYear)
   FILTER ((?wsHour = ?currentHour) &&
           (?wsDay = ?currentDay) &&
           (?wsMonth = ?currentMonth) &&
           (?wsYear = ?currentYear) &&
           (?weatherType IN (weo:WeatherState)))

}
```

Discussion:

In the Weather & Exterior Influences ontology, the actually reported weather situation is related to a weather report through the `hasWeatherReport` object property. Weather reports are then modeled as class hierarchy with `CurrentWeatherReport` being one subclass of this hierarchy (cf. Figure 4.26, Section 4.6.3). This subclass contains weather reports that cover the instantaneous weather situation at the time they are recorded. The time the report is recorded is captured by the `createdAt` object property which links a report with a particular time instant. The presented DL query only returns the current weather reports that have been created recently, hence, which creation time is equal with the current time to the granularity *hour*: therefore, a restriction on time-related properties modeled in OWL-Time like hour, day, month as well as year is specified to retrieve only the weather report that corresponds to the actual *now*[2]. If no weather state is recorded in the KB for the current time, the query returns an empty set. The DL query in this case therefore only returns the weather states that are reported for the current time and thus answers TH05.06[3].

The SPARQL query uses the same relations already introduced for the presented DL query. A SPARQL filter statement assures that the returned weather type is of `WeatherState`. This filter statement is not necessary for the correct result set however can be used in the following queries to only return weather states of a particular subtype. In SPARQL further the built-in function *now()* can be used to retrieve the current time which is then compared to the time values associated with a particular weather state (hour, day, month, year) to answer the competency question.

TH01.13: Does the current weather situation facilitate natural ventilation?

DL query:

```
AiringWeatherState and
(hasWeatherReport some
 (CurrentWeatherReport and
 (createdAt some (inDateTime some
                  ((hour value 19) and
                   (day value "---10"^^gDay) and
                   (month value "--02"^^gMonth) and
                   (year value "2013"^^gYear))))))
```

[2]The time-related parameters need to be provided to the DL query to limit the result set. In the example DL query, 10-02-2013 19:00 is taken as current time

[3]It has to be noted that in this case weather forecasting is not considered.

SPARQL query:

```
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX time: <http://www.w3.org/2006/time#>
PREFIX weo: <https://www.auto.tuwien.ac.at/downloads/thinkhome/ontology/
             WeatherOntology.owl#>

SELECT ?weatherState
WHERE
{
   ?weatherState rdf:type ?weatherType ;
                 weo:hasWeatherReport ?weatherReport.
   ?weatherReport rdf:type weo:CurrentWeatherReport ;
                  weo:createdAt ?wrTime .
   ?wrTime time:inDateTime ?dateTime .
   ?dateTime time:hour ?wsHour ;
             time:day ?wsDay ;
             time:month ?wsMonth ;
             time:year ?wsYear .
   BIND (now() AS ?currentTime)
   BIND (hours(?currentTime) AS ?currentHour)
   BIND (xsd:gDay(CONCAT("---",xsd:string(day(?currentTime)))) AS ?currentDay)
   BIND (xsd:gMonth(CONCAT("--",xsd:string(month(?currentTime)))) AS ?currentMonth)
   BIND (xsd:gYear(xsd:string(year(?currentTime))) AS ?currentYear)
   FILTER ((?wsHour = ?currentHour) &&
           (?wsDay = ?currentDay) &&
           (?wsMonth = ?currentMonth) &&
           (?wsYear = ?currentYear) &&
           (?weatherType IN (weo:AiringWeatherState)))

}
```

Discussion:

If a current weather situation facilitates natural ventilation can be expressed using the weather state hierarchy of the ontology (cf. Figure 4.25), concretely the `AiringWeatherState` subclass. The provided DL query returns the weather state if it facilitates natural ventilation and does not return a result if it does not. The SPARQL query returns true in case an airing weather state is currently reported, otherwise false. In the SPARQL query, the result set limitation to elements of type `AiringWeatherState` is realized as filter statement. Both queries are capable of successfully answering TH01.13.

TH07.13: Does the current weather situation allow passive humidification (i.e., is "moist" weather expected?)

DL query:

```
HumidifyingWeatherState and
(hasWeatherReport some
 (CurrentWeatherReport and
 (createdAt some (inDateTime some
                  ((hour value 19) and
                   (day value "---10"^^gDay) and
                   (month value "--02"^^gMonth) and
                   (year value "2013"^^gYear))))))
```

SPARQL query:

```
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX time: <http://www.w3.org/2006/time#>
PREFIX weo: <https://www.auto.tuwien.ac.at/downloads/thinkhome/ontology/
             WeatherOntology.owl#>

ASK
WHERE
{
   ?weatherState rdf:type ?weatherType ;
                 weo:hasWeatherReport ?weatherReport.
   ?weatherReport rdf:type weo:CurrentWeatherReport ;
                  weo:createdAt ?wrTime .
   ?wrTime time:inDateTime ?dateTime .
   ?dateTime time:hour ?wsHour ;
             time:day ?wsDay ;
             time:month ?wsMonth ;
             time:year ?wsYear .
   BIND (now() AS ?currentTime)
   BIND (hours(?currentTime) AS ?currentHour)
   BIND (xsd:gDay(CONCAT("---",xsd:string(day(?currentTime)))) AS ?currentDay)
   BIND (xsd:gMonth(CONCAT("--",xsd:string(month(?currentTime)))) AS ?currentMonth)
   BIND (xsd:gYear(xsd:string(year(?currentTime))) AS ?currentYear)
   FILTER ((?wsHour = ?currentHour) &&
           (?wsDay = ?currentDay) &&
           (?wsMonth = ?currentMonth) &&
           (?wsYear = ?currentYear) &&
           (?weatherType IN (weo:HumidifyingWeatherState)))

}
```

Discussion:

The query if a current weather situation facilitates natural humidification can be expressed using the weather state hierarchy of the ontology (cf. Figure 4.25, Section 4.6.3), concretely the defined subclass `HumidifyingWeatherState`. The shown DL query returns the weather state if it allows passive humidification and does not return a result if it does not. The SPARQL query returns true in case a weather state suitable for humidification is currently reported, otherwise false. In the SPARQL query, the result set limitation to `HumidifyingWeatherState` elements is realized as filter statement. Both queries are capable of successfully answering TH07.13.

TH02.23: Does the current weather situation facilitate natural processes?

DL query:

```
(AiringWeatherState or
 CoolingWeatherState or
 HumidifyingWeatherState or
 HeatingWeatherState) and
(hasWeatherReport some
 (CurrentWeatherReport and
 (createdAt some (inDateTime some
                  ((hour value 19) and
                   (day value "---10"^^gDay) and
                   (month value "--02"^^gMonth) and
                   (year value "2013"^^gYear))))))
```

SPARQL query:

```
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX time: <http://www.w3.org/2006/time#>
PREFIX weo: <https://www.auto.tuwien.ac.at/downloads/thinkhome/ontology/
             WeatherOntology.owl#>

SELECT ?weatherState
WHERE
{
   ?weatherState rdf:type ?weatherType ;
                 weo:hasWeatherReport ?weatherReport.
   ?weatherReport rdf:type weo:CurrentWeatherReport ;
                  weo:createdAt ?wrTime .
   ?wrTime time:inDateTime ?dateTime .
   ?dateTime time:hour ?wsHour ;
             time:day ?wsDay ;
             time:month ?wsMonth ;
             time:year ?wsYear .
   BIND (now() AS ?currentTime)
   BIND (hours(?currentTime) AS ?currentHour)
   BIND (xsd:gDay(CONCAT("---",xsd:string(day(?currentTime)))) AS ?currentDay)
   BIND (xsd:gMonth(CONCAT("--",xsd:string(month(?currentTime)))) AS ?currentMonth)
   BIND (xsd:gYear(xsd:string(year(?currentTime))) AS ?currentYear)
   FILTER ((?wsHour = ?currentHour) &&
           (?wsDay = ?currentDay) &&
           (?wsMonth = ?currentMonth) &&
           (?wsYear = ?currentYear) &&
           (?weatherType IN (weo:AiringWeatherState,
                             weo:HeatingWeatherState,
                             weo:CoolingWeatherState,
                             weo:HumidifyingWeatherState)))

}
```

Discussion:

This question is a more general version of the other competency questions in this group (i.e., TH01.13, TH07.13) and therefore the query solutions are again very similar. The DL queries in this case are different to the ones presented for the other two queries in a way that the reported weather situation may in the present case be of four different types: `AiringWeatherState`, `CoolingWeatherState`, `HumidifyingWeatherState` or `HeatingWeatherState`. These four types are created in the ontology to identify weather states that may be used to achieve certain tasks in the smart home: if the type of a reported weather state is one or more of these classes, it facilitates natural processes (cf. Figure 4.25, Section 4.6.3). Thus, the provided DL query answers the competency question TH02.23 for a defined time passed to the query as parameter.

For the SPARQL query also very subtle changes need to be made compared to the other queries in this group. On the one hand, the filter statement involving the weather type is extended to the four weather types as already described for the DL query. On the other hand, the presented SPARQL query returns the actual current weather situation that is reported instead of a boolean value. The weather state is in this case selected with the help of the "distinct" keyword, to only return it once, also if it belongs to more than one `WeatherState` subclass. It can for example be used for further processing tasks in the control system. The SPARQL query also successfully answers TH02.23.

Focus: future weather situation

GROUP: Predicted future weather situation

Questions TH05.07, TH05.08, TH05.09 and TH07.14 are related with perceiving a future weather situation. Therefore, the question to be answered in all of the following cases can be generalized to question TH05.07, "**What will the weather situation be like in X hours?**". The DL and SPARQL queries that are provided to answer this general question therefore can be seen as framework queries for the subsequent, more specialized, queries.

TH05.07: What will the weather situation be like in the next X hours?

DL query 1:

```
hasObservationTime some Interval
```

DL query 2:

```
hasObservationTime some
((hasBeginning some
 (inDateTime some
 ((hour some int[>=9]) and
  (day value "---11"^^xsd:gDay) and
  (month value "--02"^^gMonth) and
  (year value "2013"^^gYear))))
and
(hasBeginning some
 (inDateTime some
  ((hour some int[<=13]) and
   (day value "---11"^^xsd:gDay) and
   (month value "--02"^^gMonth) and
   (year value "2013"^^gYear)))))
```

SPARQL query:

```
PREFIX time: <http://www.w3.org/2006/time#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX weo: <https://www.auto.tuwien.ac.at/downloads/thinkhome/ontology/
             WeatherOntology.owl#>

SELECT ?weatherState
WHERE
{
   ?weatherState weo:hasObservationTime ?wsInterval .
   ?wsInterval time:hasBeginning ?wsBeginning .
   ?wsBeginning time:inDateTime ?dtBeginning .
   ?dtBeginning time:hour ?beginHour ;
                time:day ?beginDay ;
                time:month ?beginMonth ;
                time:year ?beginYear .
   BIND (now() AS ?currentTime)
   BIND (hours(?currentTime) AS ?currentHour)
   BIND (xsd:gDay(CONCAT("---",xsd:string(day(?currentTime)))) AS ?currentDay)
   BIND (xsd:gMonth(CONCAT("--",xsd:string(month(?currentTime)))) AS ?currentMonth)
   BIND (xsd:gYear(xsd:string(year(?currentTime))) AS ?currentYear)
   FILTER (((?beginHour>=?currentHour) && (?beginHour<=?currentHour+4) &&
            (?beginDay = ?currentDay) && (?beginMonth = ?currentMonth)&&
            (?beginYear = ?currentYear)))
}
```

Discussion:

To represent future weather situations, the Weather & Exterior Influences ontology provides the object property `hasObservationTime` for elements of the `WeatherState` hierarchy (cf. Figure 4.23, Section 4.6). The first presented DL query generally returns weather states that are observing the exterior situation for a specified time interval. The overlying software system is in this case responsible for filtering weather states for a particular interval in the future. It is however also possible to perform this filtering through a more complex DL query like shown in the second DL query: this example returns all weather reports that report a weather observation in the next 4 hours. To give an example, the current time is fixed as 9am and the system needs to retrieve forecasts for the weather situation that can be expected 4 hours into the future. The presented DL query therefore only returns weather states with a related observation time interval beginning (`hasBeginning`) between 9 and 13 of a particular defined day.
The same query is presented as SPARQL query using the SPARQL function *now()* returning the weather states stored for the next 4 hours into the future. These example queries show how weather states predicted for a particular time interval in the future can be extracted from the KB, thus answering TH05.07. The domain of the `hasObservationTime` property is `WeatherState` and can be used by a reasoner to infer the correct return type.
In the following queries related to weather forecast the pattern of querying for the begin time of weather states can be used as a frame to get the state at a particular time in the future. For reasons of clarity, only simple DL query variants like the first one for this example are used for illustration.

TH05.08: Can sunny weather be expected in the next X hours?

DL query:

```
SunnyWeatherState and (hasObservationTime some Interval)
```

SPARQL query:

```
PREFIX time: <http://www.w3.org/2006/time#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX weo: <https://www.auto.tuwien.ac.at/downloads/thinkhome/ontology/
             WeatherOntology.owl#>

ASK
WHERE
{
   ?weatherReport weo:reportsWeatherObservation ?weatherState .
   ?weatherState rdf:type ?type ;
                 weo:hasObservationTime ?wsInterval .
   ?wsInterval time:hasBeginning ?wsBeginning .
   ?wsBeginning time:inDateTime ?dtBeginning .
   ?dtBeginning time:hour ?beginHour ;
                time:day ?beginDay ;
                time:month ?beginMonth ;
                time:year ?beginYear .
   BIND (now() AS ?currentTime)
   BIND (hours(?currentTime) AS ?currentHour)
   BIND (xsd:gDay(CONCAT("---",xsd:string(day(?currentTime)))) AS ?currentDay)
   BIND (xsd:gMonth(CONCAT("--",xsd:string(month(?currentTime)))) AS ?currentMonth)
   BIND (xsd:gYear(xsd:string(year(?currentTime))) AS ?currentYear)
   FILTER (((?beginHour>=?currentHour) && (?beginHour<=?currentHour+4) &&
```

```
            (?beginDay = ?currentDay) && (?beginMonth = ?currentMonth) &&
            (?beginYear = ?currentYear)) && (?type IN (weo:SunnyWeatherState)))
}
```

Discussion:

A sunny weather condition is classified in the `SunnyWeatherState` concept, a class of the `WeatherState` hierarchy (cf. Figure 4.25, Section 4.6.3). This concept may therefore also be used to return all sunny weather states that are predicted for the near future. The DL query in this case restricts the type of the returned weather state to `SunnyWeatherState` instances that are observed for a specific interval. The SPARQL query illustrates the selection of weather reports according to a specific time interval in the near future and also corresponds with the solution presented for TH05.07. The only difference in this case is to identify sunny weather states in a dedicated filter statement. The DL as well as the SPARQL queries can therefore be used to answer TH05.08.

TH05.09: Can strong wind be expected in the next X hours?

DL query:

```
WeatherState and
(hasWeatherPhenomenon some (StrongWind or Storm)) and
(hasObservationTime some Interval)
```

SPARQL query:

```
PREFIX time: <http://www.w3.org/2006/time#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX weo: <https://www.auto.tuwien.ac.at/downloads/thinkhome/ontology/
             WeatherOntology.owl#>

ASK
WHERE
{
   ?weatherReport weo:reportsWeatherObservation ?weatherState .
   ?weatherState weo:hasObservationTime ?wsInterval ;
                 weo:hasWeatherPhenomenon ?wsWind .
   ?wsWind rdf:type ?windType .
   ?wsInterval time:hasBeginning ?wsBeginning.
   ?wsBeginning time:inDateTime ?dtBeginning .
   ?dtBeginning time:hour ?beginHour ;
                time:day ?beginDay ;
                time:month ?beginMonth ;
                time:year ?beginYear .
   BIND (now() AS ?currentTime)
   BIND (hours(?currentTime) AS ?currentHour)
   BIND (xsd:gDay(CONCAT("---",xsd:string(day(?currentTime)))) AS ?currentDay)
   BIND (xsd:gMonth(CONCAT("--",xsd:string(month(?currentTime)))) AS ?currentMonth)
   BIND (xsd:gYear(xsd:string(year(?currentTime))) AS ?currentYear)
   FILTER (((?beginHour>=?currentHour) && (?beginHour<=?currentHour+4) &&
            (?beginDay = ?currentDay) && (?beginMonth = ?currentMonth) &&
            (?beginYear = ?currentYear)) &&
            (?windType IN ( weo:StrongWind, weo:Storm)))
}
```

Discussion:

The KB provides classifications of weather phenomena (cf. Figure 4.24, Section 4.6.3), and to answer this competency question, the concepts `StrongWind` and `Storm` which are subclasses

of Wind can be used to identify weather states that imply high wind speeds. The presented DL query therefore, only returns weather states for which strong winds are expected and which are observed during a particular interval. Through the hasObservationTime object property a suitable interval may again be selected as already shown for TH05.07. The SPARQL query shows an example for answering TH05.09. The interval of interest as well as the desired wind types to restrict the returned weather states are selected in the SPARQL filter statement.

TH07.14: Can cool weather for passively cooling the building be expected in the next X hours?

DL query:

```
CoolingWeatherState and (hasObservationTime some Interval)
```

SPARQL query:

```
PREFIX time: <http://www.w3.org/2006/time#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX weo: <https://www.auto.tuwien.ac.at/downloads/thinkhome/ontology/
             WeatherOntology.owl#>

ASK
WHERE
{
   ?weatherState rdf:type ?type;
                 weo:hasObservationTime ?wsInterval .
   ?wsInterval time:hasBeginning ?wsBeginning.
   ?wsBeginning time:inDateTime ?dtBeginning .
   ?dtBeginning time:hour ?beginHour ;
                time:day ?beginDay ;
                time:month ?beginMonth ;
                time:year ?beginYear .
   BIND (now() AS ?currentTime)
   BIND (hours(?currentTime) AS ?currentHour)
   BIND (xsd:gDay(CONCAT("---",xsd:string(day(?currentTime)))) AS ?currentDay)
   BIND (xsd:gMonth(CONCAT("--",xsd:string(month(?currentTime)))) AS ?currentMonth)
   BIND (xsd:gYear(xsd:string(year(?currentTime))) AS ?currentYear)
   FILTER (((?beginHour>=?currentHour) && (?beginHour<=?currentHour+4) &&
            (?beginDay = ?currentDay) && (?beginMonth = ?currentMonth) &&
            (?beginYear = ?currentYear)) && (?type IN (weo:CoolingWeatherState)))
}
```

Discussion:

The class CoolingWeatherState is defined in the ontology as concept describing a weather state with a reported temperature that lies below room temperature. The DL query therefore presents a solution that selects only cool weather states observed during a particular interval. The SPARQL query represents an answer for TH07.14 using the query presented for TH05.07 as framework for time-based relationships. The filter statement again performs the type restriction.

Focus: weather report sources

TH05.11: Which local weather stations exist?

DL query:

```
SensorReportSource
```

SPARQL query:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX weo: <https://www.auto.tuwien.ac.at/downloads/thinkhome/ontology/
             WeatherOntology.owl#>

SELECT ?weatherStation
WHERE
{
   ?weatherStation rdf:type weo:SensorReportSource .
}
```

Discussion:

The ontology provides a class hierarchy for weather report sources with the top-level class `WeatherReportSource`. One subclass of this hierarchy is `SensorReportSource`: this class includes all local weather stations that are stored for the building. To answer competency question TH05.11, this class can be used by the DL and SPARQL queries, returning the local weather stations.

TH05.12: Which Internet weather services exist?

DL query:

```
ServiceReportSource
```

SPARQL query:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX weo: <https://www.auto.tuwien.ac.at/downloads/thinkhome/ontology/
             WeatherOntology.owl#>

SELECT ?weatherService
WHERE
{
   ?weatherService rdf:type weo:ServiceReportSource .
}
```

Discussion:

This competency question can be treated similarly to TH05.11: to answer this query, the ontology provides a subclass of `WeatherReportSource` which is `ServiceReportSource`. This concept includes instances that represent remote weather services which can be used to retrieve exterior situations for the near surroundings of the building. This class can again be used by the DL and SPARQL queries to select the appropriate elements in the KB and thus answer competency question TH05.12.

5.3 High-Level Competency Questions

The low-level queries created in the previous section may finally be used to answer the high-level competency questions identified in Chapter 3 and thus contribute to the achievement of goals in the multi-agent-controlled smart home. The high-level question "**Which user preferences can be obtained?**" (**TH01.07**) may be seen as combination of several low-level questions. Firstly, it is of interest to retrieve all user preferences automatically detected by the smart

home system through past user behavior, i.e., to retrieve the user habits. The pattern detection through a profile-based approach that is presented in [132] and the associated ontology described in Section 4.4 therefore contribute to the answering of this goal. Low-level questions regarding the predicted future situation like TH03.12, TH03.13, TH04.02, TH07.02 and TH07.05 are directly contributing to the answer of this competency question. The focus of these competency questions are mostly the system-detected user preferences (e.g., visual comfort, facility schedules, occupancy schedules, etc.). Otherwise, user preferences can also be manually entered, and to satisfy the general high-level competency question also these preferences need to be fetched from the ontology. In this case, especially questions TH01.08, TH02.03, TH02.04, TH03.01 and TH07.01 can be seen as supporting low-level questions. These questions involve user preferences and schedules manually entered into the system by a user, for example through a graphical user interface. In the case that no automatically inferred or manually defined user preferences can be obtained, as alternative, it should be possible for the system to retrieve standard values from the KB. For this case, the low-level questions TH01.06, TH02.01, TH02.02, TH02.30 and TH03.02 provide examples for queries that can further be used to complete preference information in case user preferences cannot be obtained. The question "**Which occupancy schedules exist for room X in building Y?**" (**TH02.05**) may be seen as another question focusing on manually entered and automatically detected user preferences. The low-level questions TH04.02 and TH07.05 are related with predicted occupancy schedules automatically inferred by the system's habit prediction approach. There also may exist a manually-entered occupancy schedule, which can be retrieved using the low-level competency question TH04.03. Another question related with building context in general is the question "**What is the near future state of the indoor environment?**" (**TH03.14**). The last two high-level questions regarding preferences and automated schedules TH01.07 and TH02.05 are directly contributing to the answering of this question. Further, the low-level question TH07.03 queries for the predicted value of all parameters that can be captured in a particular room in the next few hours and thus represents a basic query that may contribute to answering the very general high-level competency question. A high-level question regarding exterior influences is "**What are the current and near future weather conditions?**" (**TH02.22**) This question can be directly answered by combining low-level questions regarding the perception of the actual and forecasted weather states. The main two questions that in combination answer this high-level query are TH05.06 and TH05.07. There also exist more specific questions about weather states like TH01.13, TH02.23, TH05.08, TH05.09 and TH07.13, TH07.14 that also may be used as contribution to providing an answer for this high-level question. However, these questions are tailored to a particular problem and therefore only provide a fraction of the result set that is expected from TH02.22. It is further necessary for the control system to know the possibilities to retrieve exterior information. The related query "**Which possibilities exist to assess the current and future weather situation?**" (**TH05.10**) in this case comprises the two low-level questions TH05.11 and TH05.12 which can be used to retrieve all sources of weather reports. The control system is then responsible for choosing the most reliable weather source for a particular task and time frame. Another high-level question is "**Do current weather situation and room characteristics (i.e., controllable windows, shutters) facilitate passive processes?**" (**TH02.20**). Again, other high and low-level questions can be used to answer this query. Firstly, to assess if the current weather situation

facilitates passive processes, competency question TH02.23 can be used. With respect to room characteristics, for the two processes exemplified in the second use case (i.e., opening windows, raising shutters), the competency questions TH02.25 and TH02.26 may be used to evaluate if natural processes are an option to change the environmental state. One more question concerning facilities in the building in general is if a particular facility will be used in the near future (TH04.06). In the context of a smart home control it can be seen as equivalent to "**Which operation schedule exists for energy consuming facility X?**" (**TH05.15**) as through the retrieval of a facility schedule also the future state can be retrieved. There are two low-level questions identified from the use cases and are directly contributing to answering this question, TH07.01 and TH07.02. The first of these low-level questions queries the Users & Preferences ontology part for manually defined facility preferences, while the second query is related with automatically detected patterns. If no schedules exist on the facility level, the predicted occupancy can also be used to estimate if a facility will be used in the near future. Therefore, in a wider sense, also the question TH02.05 can help the control system to answer the query. The results together can be used to assess the probability that a facility will be used which in further consequence can be used for optimized energy scheduling. For the perception of the current state, the control system operating on the ontology must be capable to retrieve information about the indoor environment. Therefore, the high-level question "**What is the current state of the indoor environment?**" (**TH02.15**) needs to be answered. In this particular case, the state of facilities as well as different environmental parameters are important. Low-level questions that contribute to the answer of this question involve queries to the Energy & Resource information ontology and have already been presented as belonging to one single low-level group before (cf. Section 5.2, Group: current perception of the environment). The low-level questions TH01.01, TH02.16, TH02.29, TH03.07, TH03.08 and TH04.10 are therefore contributing to answer TH02.15. As the list of competency questions is not exhaustive, however, more queries may be needed to fully perceive the current environmental state of a part of the building. The provided questions can for that reason only be seen as an exemplification of possible queries with regard to the identified use cases, and will have to be evaluated in detail for each particular room with regard to installed equipment and completeness of represented environmental parameters. It may be necessary to define additional queries to retrieve the complete state of the environment. Directly connected with the current state of the environment is the high-level question "**Which facilities can be turned off to save energy?**" (**TH04.05**). The low-level questions TH04.04, TH04.11 and TH04.14 related with the current state of a facility can be seen as contributing to answering the question. The first two low-level queries may be used to assess if a facility is currently consuming energy and the third question directly tells if a facility needs permanent supply. In a broader sense, apart from current facility states, naturally also occupancy is important when answering this high-level question. Therefore, the high-level question regarding predicted occupancy (TH02.05) as well as the query for current occupancy (TH02.29) conducted for the particular room the facility is installed in need to be answered to let the control system decide if a facility can be safely turned off to save energy. A general query regarding the building and its construction itself is "**What are the building values for room X?**" (**TH07.06**). This question can be seen as a combination of the presented low-level queries TH02.13, TH02.07, TH02.08, TH02.09, TH02.11, TH02.12, TH02.14, TH02.24, TH02.27 and TH02.28. These queries again are only a selection of possi-

ble queries and not all of the information retrieved through these queries may be necessary for a particular system task. It is in the responsibility of the overlying control system to fetch all needed information through a combination of low-level queries that represent the situation in the necessary detail. As the list of competency questions is not exhaustive, other queries not mentioned at this point might be needed if trying to solve other situations than the ones outlined as use cases.

5.4 Summary

In this chapter it has been shown that by solely using properties and concepts from the created ontologies it is possible to sufficiently answer the questions identified in the use case analysis of Chapter 3. Therefore, for all low-level competency questions, formal representations using the created vocabulary have been created on the one hand in pure DL and on the other hand in SPARQL. Combinations of low-level questions have then been presented to answer previously identified high-level competency questions. The evaluation showed that by solely using the ontology vocabulary all questions can be successfully formulated and answered in SPARQL, while a large share of the queries also has a direct representation as DL query. For those questions where an expression in DL is not directly possible, auxiliary DL queries are presented which do not precisely answer the posed competency questions, however, demonstrate the representation and availability of related knowledge in the ontology. As the system concept includes a SPARQL interface between the multi-agent system and the knowledge base, the SPARQL representation of the low-level questions is the main focal point of the evaluation. The phrasing of competency questions as SPARQL queries is conducted in a way, that only simple SPARQL functions have been used, mainly to link triples. This deliberate limitation in usage of functionalities is performed to assure that all of the knowledge necessary to answer the queries is also actually represented in one of the created ontology modules. Only in case of time-based queries, also more sophisticated SPARQL functions have been used to, for example, retrieve the current time or conduct XSD datatype comparisons. The example SPARQL queries introduced in this chapter are further expressed in a way so that they may be used as templates for actual production queries in a multi-agent-controlled smart home system. The general form using filter functions to concretely specify needed limitations allows easy adaptation to similar problems that may not have been covered by the identified use cases.

CHAPTER 6

Integration into a multi-agent-based Smart Home System - Case Study

When considering an ontology as base for a multi-agent-controlled smart home system its purpose is mainly the environmental representation: the smart home, its context and surroundings have to be depicted in a way so that the MAS can take all parameters into account that may be influencing the behavior of the building. While this environmental representation includes the provision of a great variety of parameters and influence factors, the way how these parameters are used to operate the building, hence, the control strategies, are realized in the multi-agent system. The *goals* that have to be achieved by such a system may to some part have a representation in the knowledge base (cf. Section 4.4) however, the path to reach these goals needs to be determined by the control system itself. With respect to generality and flexibility, to model all possible paths in the knowledge base itself is inadvisable, as it would lead to an overly complex representation. Therefore, sometimes the reaching for goals involves the answering of questions which are not directly represented in the knowledge base and have to be solved using consecutive queries to the ontology with the multi-agent system determining the path according to the perceived environmental situation. This chapter exemplifies one of the use cases introduced before (cf. Chapter 3) and details the interaction between ontology and multi-agent system to reach a particular desired state. As this dissertation is conducted in the course of a smart home project, the following scenario has already been outlined from a multi-agent perspective in [199]. At this point, the scenario is again discussed from an ontology viewpoint to detail the communication flow and show how the defined ontologies can be used to achieve one of the global goals of a smart home system.

To illustrate the interactions between ontology and multi-agent system, the use case **TH02 - Assure Thermal Comfort** is chosen as case study. For simplification reasons, only the automatic preparation of the environment with respect to temperature is shown. Other parameters involved with thermal comfort (e.g., humidity, air quality) may be adjusted in a similar way. The environment to be prepared in this example case is one particular room of the building. Further, to present one coherent example and stay in line with the deliberations in [199], the cooling case is

selected for discussion. Again, the heating case may be solved in an analogous manner.
When reviewing the use case description provided in Chapter 3, several questions can be identified that need to be answered by the system but do not lie in the responsibility of the knowledge base. Some examples are:

- When does the temperature have to be at comfort level?
- What is the exact preparation time to reach comfort temperature?
- When does a process have to be started at the latest possible time?
- Is it possible to start a specific energy efficient process at a particular point in time?

The following sequence diagrams illustrate how these global questions (i.e., goals) can be answered by the system with the help of information provided in the knowledge base. For the subsequent discussion, a situation between MAS and KB is assumed where no information had been exchanged before. This means, the multi-agent system does not yet have any information about the environment it is operating on. With such an assumed situation, the entire communication between KB and MAS can be illustrated. The scenario further presents the connection to the KB queries identified before (cf. Chapter 5) and shows how they can be used by the MAS to gather the necessary information. Of course, considering usual day-to-day operation of the smart home system, it has to be kept in mind that some of the demonstrated sequences only have to be performed infrequently as the MAS may for example cache information from a previous request. Nevertheless, the full sequence is shown at this point to demonstrate interaction with all different ontology parts identified in this work.

6.1 Introductory Phase: occupancy detection

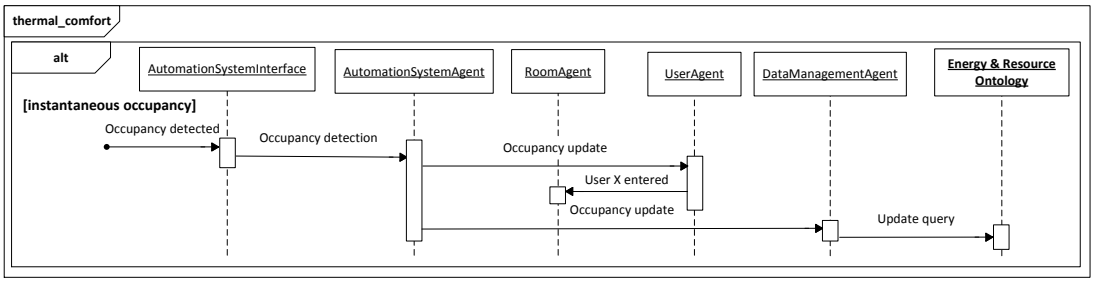

Figure 6.1: Occupancy detection for a particular part of the building: instantaneous occupancy.

The shown scenario begins with detected or predicted occupancy for a particular room in the building. There mainly exist three different scenarios how occupancy can be detected: either instantaneous occupancy is detected by sensors (cf. Figure 6.1), or a user did manually enter his occupancy preference or a reliable pattern exists which predicts occupancy for a particular

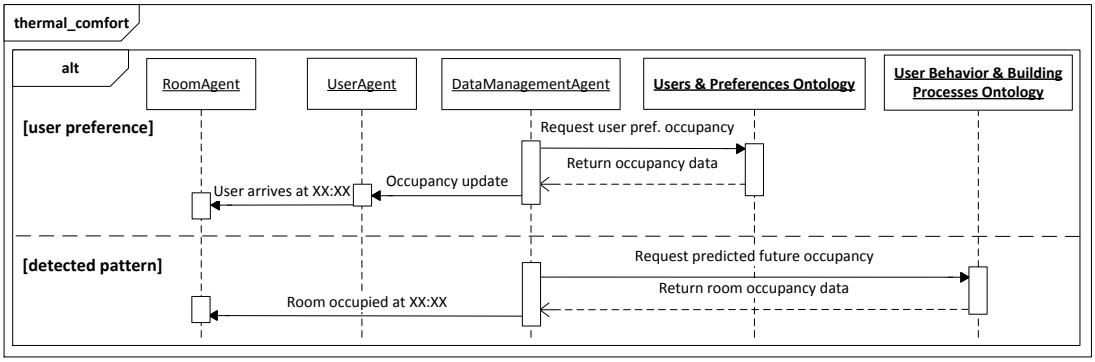

Figure 6.2: Occupancy detection for a particular part of the building: occupancy prediction through user preferences; occupancy prediction through detected occupancy patterns.

time and space (cf. Figure 6.2). In the case instantaneous occupancy of a particular user is detected, for example, through a sensor or RFID tag reader, the information is firstly transferred to the responsible agent (`AutomationSystemAgent`). In case of individual user detection via RFID, this agent informs the responsible user agent (`UserAgent`) that the resident entered the room and updates the occupancy sensor value for the particular room the user is located at the moment. Updating sensor information in the KB can be done analogous to query **TH01.03** by writing the new occupancy sensor value to the state value of the sensor representation in the Energy & Resource ontology. The user agent further informs the room agent associated with the occupied room that the given user has entered and initiates the thermal comfort scenario.
In the other two cases, the `DataManagementAgent` may from time to time[1] query the KB for patterns that are available for a particular room in the building. These patterns may involve user-defined or automatically detected occupancy schedules. If a user presence preference schedule can be retrieved from the Users & Preferences ontology through a query like **TH04.03**, the `UserAgent` associated with the particular user is contacted, which itself informs the `RoomAgent` that the user will arrive at a particular time. In case no user preference is available or only room-based occupancy detection is implemented, the `DataManagementAgent` may retrieve a reliable and valid occupancy pattern for the particular room through the query **TH04.02**. In case a valid pattern is found, the `RoomAgent` is directly informed about the time in the future at which the room will be occupied.

6.2 Auxiliary Information Phase: preferences retrieval, building characteristics collection, weather prediction request

In the next phase, the smart home system needs to gather auxiliary information that may influence the preparation time of a particular room (cf. Figure 6.3). Firstly, again the Users & Prefer-

[1] usually once per day

ences ontology is queried for temperature preferences. If a particular user is detected the system tries to obtain his or her preferred comfort temperature (**TH02.03**). In case no user-defined preferences are available or if pattern recognition is favored, the responsible agent queries for a valid and reliable pattern in the User Behavior & Building Processes ontology. In this case, the query **TH03.12** adapted to returning the detected comfort temperature schedule instead of the visual comfort schedule may be used. If only room-based occupancy detection is supported and no user-defined comfort temperature preferences and also no valid patterns are available, the system may alternatively query the Users & Preferences Ontology for a defined standard temperature preference (**TH02.01**).
To consider the influence of building structure on the preparation time, the room agent further queries for the building characteristics. Certainly, building characteristics can be stored in the agent and in the best case only have to be fetched once. The retrieval of room characteristics can be accomplished by query **TH07.06** while some of its subqueries can be selected to only return values of interest for the specific case. Parameters like U-value or alternatively thickness and material of walls, the orientation and volume of the room or the number and size of window openings can be taken into account to calculate the latest possible time to start a (possibly energy-intensive) cooling process. Finally, exterior influences can be retrieved from the Weather & Exterior Influences ontology to assess a possible exploitation of the current weather situation in favor of the task to be accomplished. In the cooling case, the query **TH07.14** is suitable to retrieve information about the expected weather situation.

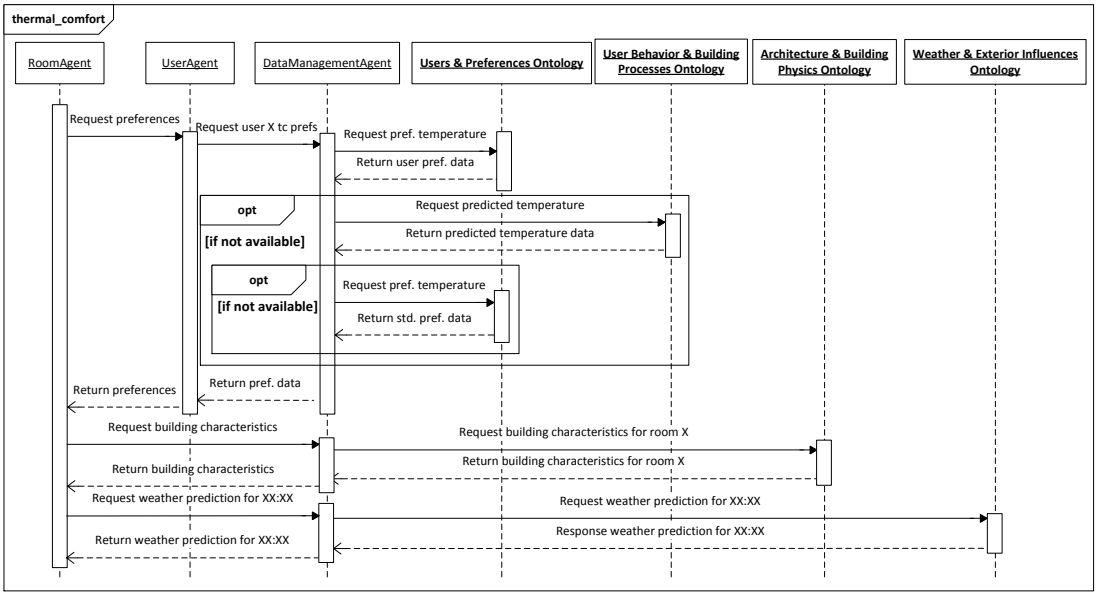

Figure 6.3: Collection of auxiliary data as influence parameters to the comfort temperature preparation time: temperature preferences or temperature patterns; building characteristics; weather prediction.

6.3 Preparation Phase: cooling options retrieval, energy consumption obtainment

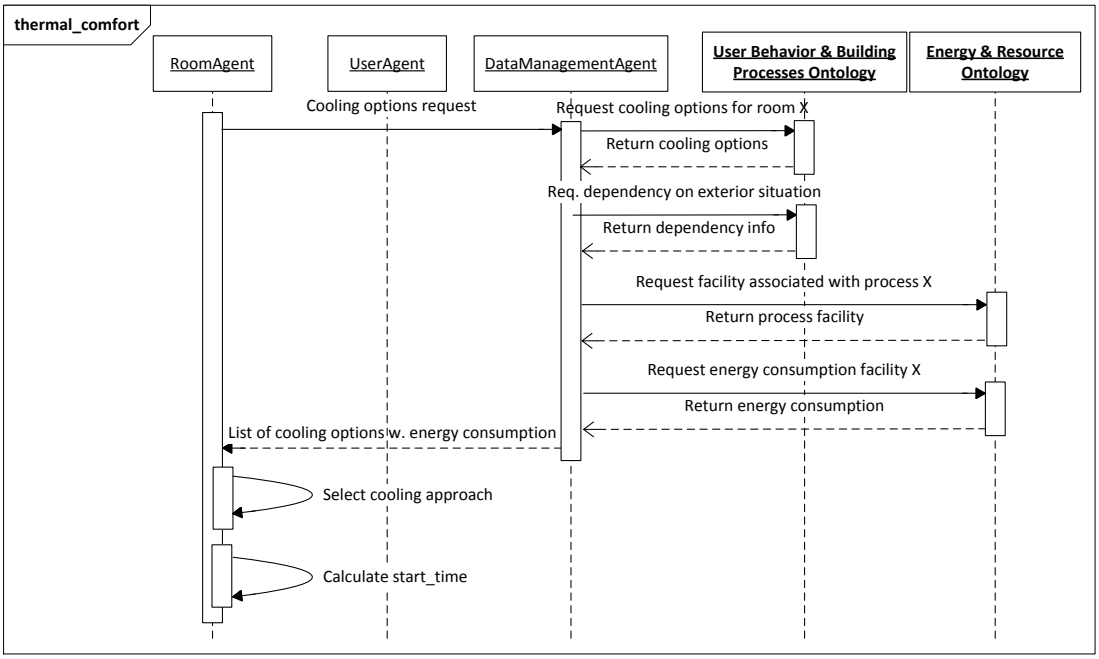

Figure 6.4: Fetching information for process selection: cooling options; associated facilities; facility energy demand.

In the preparation phase, the multi-agent system firstly requests all possible options that may be used to cool a room. In this case, the `RoomAgent` starts by querying for available cooling options in the particular room. The `DataManagementAgent` may issue a query (**TH02.18**) to the User Behavior & Building Processes ontology to retrieve all available cooling processes. Another request to this ontology reveals if a process is dependent on the weather situation (**TH01.11**). This information is relevant when considering favorable exterior conditions to achieve the current goal. The agent may further issue queries to the Energy & Resource ontology to associate a facility and an energy demand to a particular process (**TH01.10**, **TH05.14**)[2].
After retrieving this information from the ontology, it is returned to the `RoomAgent` which is then capable of selecting the most energy efficient process to reach a particular state. In case a process depends on the weather situation, the agent certainly has to consider the exterior influences fetched before from the Weather & Exterior Influences ontology: for example, if a cool weather situation is expected in the selected time interval, it can choose the most energy efficient cooling process from all available processes including the ones that are dependent on the

[2]The different calls to the KB are only shown for illustrative purposes. In an actual implementation using an API with a triple store (e.g., Jena [7]) the queries may of course be consolidated.

exterior situation (e.g., natural cooling by opening the window). Otherwise, if the weather is not favorable to the achievement of the present goal, only processes that are independent of the exterior situation may be used. Also in this case, the most energy efficient process has to be chosen by the MAS according to its energy profile. Further, by taking into account the building characteristics for the respective room the optimal starting time for the specific process can be calculated.

6.4 Execution Phase: control command request, facility querying

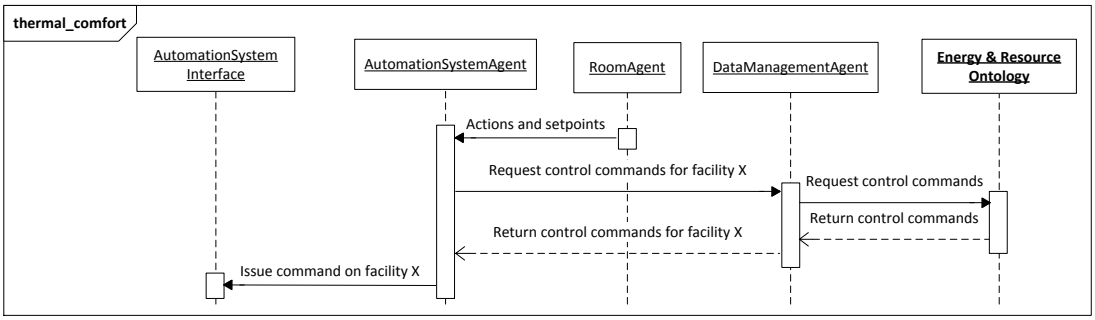

Figure 6.5: Information retrieval for executing the actions in the building automation system: process facility; control commands.

The execution phase starts at the previously calculated time with the transmission of facility actions and setpoints from the `RoomAgent` to the `AutomationSystemAgent`. The `RoomAgent` passes the particular facility to be controlled as well as necessary actions and setpoints to the `AutomationSystemAgent`. In case this agent does not yet have information about the control possibilities of the facility, it needs to fetch the possible control commands from the Energy & Resource ontology. Query **TH04.13** can be used to return information about possible control commands for the selected facility. It is the responsibility of the agent system to match the suitable command to the action requested by the room agent. The `AutomationSystemAgent` may then use the chosen command by directly calling the `AutomationSystemInterface` to issue it on the selected facility. It can be noted that in the best case the request for control commands has to be performed only once, provided the `AutomationSystemAgent` caches this information for the facilities in the building. However, as the present chapter exemplifies the initial execution of the thermal comfort scenario in the system and does not assume any pre-fetched information, the shown interaction is required.

CHAPTER 7

Conclusions & Outlook

7.1 Contribution

With the wide variety of parameters found in current home automation systems, the creation of an unambiguous base of knowledge is seen as a prerequisite of future intelligent buildings. Therefore, the central question that had to be answered in this work was

> *"How can a comprehensive and sophisticated representation of environmental parameters support intelligent behavior in future smart home systems?"*

The prevalent situation that is found in the typical smart home is a highly heterogeneous and complex environment which is in need of a representation that is independent of underlying technologies and associated peculiarities. To generate a common model of this environment for an intelligent building control system, a knowledge base is indispensable as it provides a view of the part of the world relevant in a particular application case while at the same time reflecting reality as close as possible. In case of a multi-agent-controlled smart home, a knowledge base can thus be seen as a mediator between agents that provides a global view of the world around them and enables semantic interoperability. With the agents committing to the shared vocabulary, they are capable of interoperation while still staying autonomous [24]. In this context, an ontology defines terms that are of interest for the created system and thus limits the universe of discourse, abstracting from unnecessary details. The knowledge base is the central repository for data, information and knowledge about the environment. It can be queried by the autonomous building control system operating on it to retrieve facts necessary for the optimization of main system tasks and reach global goals like energy efficiency and user comfort.
This work presents a comprehensive knowledge base designed to perceive the current state of the building and its environment in order to provide a unified view of the world for a multi-agent-controlled smart home. Based on the openness of the multi-agent approach the Web Ontology Language was chosen as representation formalism. The methodology used to construct the ontology can be seen as requirement-centric approach. Therefore, firstly, envisioned benefits of an

intelligent smart home system have been worked out in form of use cases. The described use cases have been focused around the two main goals that an intelligent automation system for the residential sector needs to achieve: user comfort and building energy efficiency. To design the ontology according to the requirements postulated by the use cases, a set of competency questions has been extracted from each use case determining the focus of the knowledge base. To achieve a wide applicability and a certain amount of generality, the created ontology has subsequently been split into several modules:

1. *Architecture & Building Physics:* This part describes the static building structure and its properties and models the general layout of the building. Important parameters that are treated in this module include types of building elements such as walls and windows (i.e., material, construction) but also their size and orientation. Focus was put on the representation of parameters that are considered influential with respect to main building control tasks.

2. *Energy & Resource:* This module provides an energy-centric view on the interior and exterior building environment. It describes energy facilities that consume or produce energy in the building itself and also represents their current state. Because of the energy-centric viewpoint, additionally also energy providers, i.e., companies providing energy to the building and their terms of energy delivery are included.

3. *User Behavior & Building Processes:* As user behavior can be seen as major influence factor for reaching a comfortable environment in an energy efficient way, this component is designed to model predicted future user behavior. A profile-based approach for smart home control (cf. [129]) is used as basis. This method describes user profiles and patterns expressing user habits which have been modeled in the designed ontology. Additionally, building processes that can be associated with detected user habits are represented as well.

4. *Users & Preferences:* Allowing the occupant to influence the behavior of a system and to perform manual adjustments in the environment is of major importance for system acceptance. Alongside the representation of general and scheduled preference settings for each inhabitant, this module also provides elements to characterize different users. Further, a hierarchy of activities is introduced to give the user the possibility to create customized preferences for different scheduled activities.

5. *Weather & Exterior Influences:* To represent the exterior situation as additional influencing factor on the energy consumption of a smart home, the designed knowledge base provides a module that represents weather phenomena and situations. In this ontology, weather states useful for smart home operation are identified and illustrated. Different types of weather reports allow to distinguish between current weather states and several weather forecasts as well as local weather reports perceived through sensors and reports retrieved through an Internet weather service.

For each of the modules the dissertation discussed introduced concepts and their relationships. Focus was put on answering the described competency questions. At the same time, a general view of the modeled domain was provided to create ontologies that may be used for a wider field

of application. With regard to the general research question, reasoning capabilities of the Web Ontology Language have been exploited where useful in order to represent common queries already in the ontology and thus to take away complexity from the system operating on it. These defined queries have necessarily been tailored to the previously identified smart home requirements, while the general structure of the ontologies remains applicable to a variety of different use cases, also in domains apart from smart home control.
Finally, this work concluded with the evaluation of the ontologies against the identified competency questions. This evaluation showed that the ontologies in combination are capable of answering all posed questions. It further led to the implementation of several queries that may be used as reference for interfacing with the created knowledge base.

7.2 Derived Publications

Several publications have previously been published at international conferences and in scientific journals and eventually led to this dissertation. Firstly, the focus was put on the general system architecture and a definition of appropriate use cases. The general system concept is initially introduced in [203]. With respect to the knowledge base, this publication suggests a first partitioning of the universe of discourse into different domains of interest. This classification has gradually been revised and adapted. The article [202] goes into detail on the general system architecture and presents main use cases for a future smart home system.
Further publications that exclusively focus on the created knowledge base can be categorized according to the ontology module they describe. Firstly, regarding the Architecture & Building Physics ontology, in [147], an approach of retrieving information on the building structure and architecture from a Building Information Model is described. The generated model is presented in more detail in [148] where in this case especially the refinement through a manually created vocabulary and the extension through logical rules are the focal points. Other publications focus on the central Energy & Resource ontology module, describing facilities, energy consumption and energy production as well as delivery. The first publication in this context, [150], gives an overview of how to represent energy-related information for energy providing companies and building facilities using the Web Ontology Language. Alongside this general overview, in [149], additional use cases are introduced that describe the application of the ontology module in a smart home context. The article [152] starts from a combination of the previous two publications and extends the provided information regarding resource representation and the presented use cases. In [139], the application of the Energy & Resource ontology for smart homes in the context of a smart grid is illustrated. The focus of the deliberations lies on the energy provider side. The conceptualization realized for the Weather & Exterior Influence module is presented in [151] which describes in detail how current and near-future weather situations are reflected in the ontology and how reasoning can be applied to identify favorable weather conditions. The User Behavior & Building Processes ontology module is finally introduced in [146]. It is firstly shown how concepts used in the profile-based control strategies of the envisioned smart home system are mapped to an ontological representation. Further, the article provides an example instantiation of the ontology for the thermal comfort use case highlighting reasoning capabilities as well as interconnections between created concepts.

7.3 Future Work

While the system concept outlined in this dissertation features a multi-agent approach, the application of the created smart home ontology is not limited to this particular system type. Instead, any type of system may be operating on the ontology using the general queries provided in the present work. Further, by making the designed ontologies available through the Internet and discussing their interrelations and capabilities in detail in this thesis, the widespread usage in other scientific and non-scientific works is promoted and encouraged. One future field of research is the application of the ontology on a larger scale in multiple systems. The distribution of intelligence between buildings may lead to advanced use cases like the autonomous trading of excess energy and associated benefits on a district or even city level. Also with respect to user comfort new fields of application arise like, for example, the transition and reuse of user preferences at different locations and between different building control systems. Challenges and benefits emerging from the coupling of buildings therefore still remain to be explored.
Apart from contributions in the smart home domain the outcome of this thesis is also beneficial for related areas. From the community viewpoint (i.e., villages, cities), the introduction of the created ontology in multiple building control systems brings along additional benefits. Therefore, the focus of further research may be the application of the ontology in the context of emerging *smart cities*. Buildings can in this case be seen as distributed energy prosumers contributing to the general energy supply of a community through in situ energy generation by micro plants such as solar panels or geothermal heat pumps. A *smart grid* assuring coordinated and economical operation of distributed energy sources and loads is considered the next step in energy management [80]. The unified representation of energy-relevant concepts in smart buildings through a common ontology may be used to simplify distributed tasks like the optimization of energy consumption and production in the grid: in this context, the shifting of energy intensive processes can significantly reduce energy demand peaks which currently pose one of the main problems for energy supply operators.
Another field of research can be the adaptation of parts of the ontology for application in a smart energy grid control system. The modularization of the created knowledge base facilitates the partial re-use in the smart grid to, for example, collect knowledge about the predicted weather situation and the associated energy production that can be expected from certain renewable energy power plants like wind parks or photovoltaic fields. In this case, the ontology for Weather & Exterior Influences necessarily has to be adapted and extended to fit such smart city use cases. Also for other modules, a possible application in a grid control system needs to be investigated. Especially with the introduction of the Internet of Things (IoT), which envisions a comprehensive connection of sensors and field devices through the Internet [162] another possible area of application for a smart home ontology opens. If sensors that are integrated in the smart home are described in a machine-interpretable language, innovative business processes arising from the interconnection of sensors and devices through the IoT can be simplified. This has the potential to lead to a more interconnected world where, for example, optimizations in energy consumption can be targeted on a truly global level.

Bibliography

[1] R. L. Ackoff. From Data to Wisdom. *Journal of Applied System Analysis*, 16:3–9, 1989.

[2] ABB AG. ABB i-bus EIB/KNX Weather Station WS/S 4.1. Available Online: http://www05.abb.com/global/scot/scot209.nsf/veritydisplay/6501fbf7c0fc4d78c12570830047964b/$file/2cdc504031d0202.pdf. Product Manual, Last Accessed 18. May 2013.

[3] ABB AG. Energy measurement with ABB i-bus KNX. Available Online: http://www.knx-gebaeudesysteme.de/sto_g/English/SALES_INFORMATION/EMS_3161_FL_EN_V1-1_2CDC505119D0201.PDF, 2012. Product Information. Last Accessed: 14. April 2013.

[4] D. Allemang and J. Hendler. *Semantic Web for the Working Ontologist*. Morgan Kaufmann Publishing/Elsevier, 2008.

[5] American Society of Heating, Refrigerating and Air-Conditioning Engineers. ANSI/ASHRAE Standard 55-2010 – Thermal Environmental Conditions for Human Occupancy, 2010. Published standard.

[6] American Society of Heating, Refrigerating and Air-Conditioning Engineers. ANSI/ASHRAE Standard 62.1-2013 - Ventilation for Acceptable Indoor Air Quality, 2013. Published standard.

[7] Apache Software Foundation. Apache Jena. http://jena.apache.org/. Project homepage, Last Accessed: 12. December 2013.

[8] Apache Software Foundation. Fuseki: serving RDF ddata over HTTP. http://jena.apache.org/documentation/serving_data/. Project homepage, Last Accessed: 12. December 2013.

[9] G. Aranda-Mena and R. Wakefield. Interoperability of building information: myth or reality? In *Proc. of the eWork and eBusiness in Architecture, Engineering and Construction Conference*, pages 127–133, 2006.

[10] J. Augusto and C. Nugent. Smart Homes Can be Smarter. In *Designing Smart Homes: The Role of Artificial Intelligence*, chapter 1. Springer, 2006.

[11] S. Azhar, A. Nadeem, J. Y. N. Mok, and B. H. Y. Leung. Building Information Modeling (BIM): A New Paradigm for Visual Interactive Modeling and Simulation for Construction Projects. In *Proc. of the International Conference on Cosntruction in Developing Countries*, pages 435–446, 2008.

[12] F. Baader. Logic-based knowledge representation. In *Artificial Intelligence Today, Recent Trends and Developments*, number 1600 in Lecture Notes in Computer Science (LNCS), pages 13–41. Springer Verlag, 1996.

[13] F. Baader, I. Horrocks, and U. Sattler. *Handbook of Knowledge Representation*, chapter 3, pages 135–179. Elsevier B.V., 2008.

[14] F. Baader and W. Nutt. *The Description Logic Handbook: Theory, Implementation, and Applications*, chapter 2, pages 43–95. Cambridge University Press, 2003.

[15] C. A. Balaras, K. Droutsa, E. Dascalaki, and S. Kontoyiannidis. Heating energy consumption and resulting environmental impact of European apartment buildings. *Energy and Buildings*, 37(5):429–442, 2005.

[16] J. Bally, T. Boneh, A. E. Nicholson, and K. B. Korb. Developing An Ontology for the Meteorological Forecasting Process. In *International Conference on Decision Support Systems*, 2004.

[17] A. Barbato, L. Borsani, A. Capone, and S. Melzi. Home energy saving through a user profiling system based on wireless sensors. In *Proc. of the ACM Workshop on Embedded Sensing Systems for EnergyEfficiency in Buildings*, pages 49–54, 2009.

[18] L. Bartram, J. Rodgers, and R. Woodbury. Smart homes or Smart Occupants? Supporting Aware Living in the Home. In *Proc.of the IFIP TC 13 International Conference on Human-Computer Interaction - Volume Part II*, pages 52–64, 2011.

[19] J. Beetz, J. van Leeuwen, and B. de Vries. IfcOWL: A case of transforming EXPRESS schemas into ontologies. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, 23(01):89, 2009.

[20] G. Bellinger, D. Castro, and A. Mills. Data, Information, Knowledge, and Wisdom. Available Online: `http://www.systems-thinking.org/dikw/dikw.htm`, 2004. Last Accessed: 29. December 2012.

[21] A. Ben Abacha and P. Zweigenbaum. Medical question answering: Translating medical questions into SPARQL queries. In *Proc. of the International Health Informatics Symposium*, pages 41–50, 2012.

[22] K.-I. Benta, A. Hoszu, L. Văcariu, and O. Creţ. Agent based smart house platform with affective control. In *Proc. of Euro-American Conference on Telematics and Information Systems*, pages 1–7, 2009.

[23] D. Benyoucef, P. Klein, and T. Bier. Smart Meter with non-intrusive load monitoring for use in Smart Homes. In *Proc. of the International Energy Conference (EnergyCon)*, pages 96–101, 2010.

[24] J. Bermejo-Alonso, R. Sanz, and I. Lópes. A Survey on Ontologies for Agents. Technical Report ASLab-ICEA-R-2006-002 v 1.0 Draft, Autonomous Systems Laboratory (ASLab) Universidad Politécnica de Madrid, April 2006. Available Online: `http://tierra.aslab.upm.es/documents/controlled/ASLAB-R-2006-05-v1-Draft-JB.pdf`, Last Accessed: 21. February 2013.

[25] T. Berners-Lee. Semantic Web on XML. Available Online: `http://www.w3.org/2000/Talks/1206-xml2k-tbl/slide10-0.html`, December 2000. Last Accessed: 24. November 2013.

[26] T. Berners-Lee and M. Fischetti. *Weaving the web: The original design and ultimate destiny of the world wide web by its inventor*. Harper, 1999.

[27] T. Berners-Lee, J. Hendler, and O. Lassila. The Semantic Web. *Scientific American*, 284(5):34–43, May 2001.

[28] P. Bertoldi and B. Atanasiu. Electricity Consumption and Efficiency Trends in European Union. Status Report, Institute for Energy, 2009. Joint Research Center - European Commission.

[29] P. Bertoldi, B. Hirl, and N. Labanca. Energy Efficiency Status Report 2012 - Electricity Consumption and Efficiency Trends in the EU-27. Available Online: `http://iet.jrc.ec.europa.eu/energyefficiency/sites/energyefficiency/files/energy-efficiency-status-report-2012.pdf`, 2012. Last Accessed: 9. March 2013.

[30] M. Bhatt, J. Hois, and O. Kutz. Ontological modelling of form and function for architectural design. *Applied Ontology*, 7(3):233–267, 2012.

[31] Y. Bichiou and M. Krarti. Optimization of envelope and HVAC systems selection for residential buildings. *Energy and Buildings*, 43:3373 – 3382, 2011.

[32] H. Bohring and S. Auer. Mapping XML to OWL Ontologies. In *Leipziger Informatik-Tage, volume 72 of LNI*, pages 147–156, 2005.

[33] D. Bonino, E. Castellina, and F. Corno. DOG: An Ontology-Powered OSGi Domotic Gateway. In *Proc. of the International Conference on Tools with Artificial Intelligence*, pages 157–160, 2008.

[34] D. Bonino and F. Corno. DogOnt - ontology modeling for intelligent domotic environments. In *Proc. of the International Conference on the Semantic Web*, pages 790–803, 2008.

[35] D. Bonino and F. Corno. Interoperation modeling for intelligent domotic environments. In *Proc. of the European Conference on Ambient Intelligence (AmI)*, pages 143–152, 2009.

[36] D. Bonino, F. Corno, and F. Razzak. Enabling machine understandable exchange of energy consumption information in intelligent domotic environments. *Energy and Buildings*, 43(6):1392–1402, February 2011.

[37] P. Bonsma and M. Bourdeau. Semantic product modelling and configuration: challenges and opportunities. *Journal of Information Technology*, 14:507–525, August 2009.

[38] R. J. Brachman and H. J. Levesque. *Knowledge Representation and Reasoning*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2004.

[39] R. J. Brachman and J. G. Schmolze. An Overview of the KL-ONE Knowledge Representation System. *Cognitive Science*, 9(2):171–216, 1985.

[40] J. E. Braun. Reducing Energy Costs and Peak Electrical Demand through Optimal Control of Building Thermal Storage. *ASHRAE Transactions*, 96(2):876–888, 1990.

[41] D. Brickley. Basic Geo (WGS84 lat/long) vocabulary. Available Online: `http://www.w3.org/2003/01/geo/`, February 2004. Last Accessed: 25. August 2013.

[42] N. Bricon-Souf and C. R. Newman. Context awareness in health care: A review. *Medical Informatics*, 76(1):2–12, 2007.

[43] G. Brusa, M. L. Caliusco, and O. Chiotti. A Process for Building a Domain Ontology: an Experience in Developing a Government Budgetary Ontology. In *Proc. of the Second Australasian Ontology Workshop*, volume 72, pages 7–15, 2006.

[44] A. Buccella, A. Cechich, and N. Brisaboa. An Ontology Approach to Data Integration. *Journal of Computer Science & Technology*, 3(2):62–68, 2003.

[45] buildingSMART International: Model Support Group, Implementation Support Group. Industry Foundation Classes Specifications. Available Online: `http://www.buildingsmart-tech.org/specifications/ifc-releases/`, March 2013. Current Schema: IFC4, Last Accessed: 19. May 2013.

[46] D. A. Campbell. Modeling rules. Available Online: `http://www.architectureweek.com/2006/1011/tools_1-1.html`, October 2006. Magazine: Architecture Week, Last Accessed: 17. May 2013.

[47] M. Chan, D. Estève, C. Escriba, and E. Campo. A review of smart homes - present state and future challenges. *Computer Methods and Programs in Biomedicine*, 91:55–81, 2008.

[48] H. Chen, T. Finin, and A. Joshi. The SOUPA Ontology for Pervasive Computing. In *Ontologies for Agents: Theory and Experiences*, pages 233–258, 2005.

[49] H. Chen, F. Perich, D. Chakraborty, T. Finin, and A. Joshi. Intelligent Agents Meet Semantic Web in a Smart Meeting Room. In *Proc. of the International Conference on Autonomous Agents and Multiagent Systems*, pages 854–861, 2004.

[50] L. Chen and C. Nugent. Ontology-based activity recognition in intelligent pervasive environments. *International Journal of Web Information Systems*, 5(4):410–430, 2009.

[51] D. Chwieduk. Towards sustainable-energy buildings. *Applied Energy*, 76:211–217, 2003.

[52] A. Cockburn. *Writing Effective Use Cases*. Addison Wesley Longman, Inc., November 2001.

[53] M. D. Cohen and P. Bacdayan. Organizational Routines Are Stored As Procedural Memory: Evidence from a Laboratory Study. *Organization Science*, 5(4):554–568, 1994.

[54] D. Conradie. Building Information Modeling (BIM). *Green building handbook South Africa. (A guide to ecological design)*, Vol.1(Feb-2009):pp 225–231, 2009.

[55] D.J. Cook, M. Youngblood, E.O. Heierman, K. Gopalratnam, S. Rao, A. Litvin, and F. Khawaja. MavHome: an agent-based smart home. In *Proc. of the International Conference on Pervasive Computing and Communications*, pages 521–524, 2003.

[56] A. Cooperman, J. Dieckmann, and J. Brodrick. Using Weather Data For Predictive Control. *ASHRAE Transactions*, (December):130 – 132, 2010.

[57] Cycorp Inc. Opencyc platform. Available Online: `http://www.cyc.com/platform/opencyc`, 2013. Last Accessed: 12. April 2013.

[58] S. Darby. The effectiveness of feedback on energy consumption: a review for DEFRA of the literature on metering, billing and direct displays. Technical report, Environmental Change Institute, University of Oxford, 2006.

[59] S. Davidoff, M. K. Lee, C. Yiu, J. Zimmerman, and A. K. Dey. Principles of Smart Home Control. In *Ubicomp*, volume 4206 of *Lecture Notes in Computer Science (LNCS)*, pages 19–34. Springer, 2006.

[60] R. Davidrajuh. Logic Programming for Machine Tools. In *PROLAMAT*, pages 717–722, 2006.

[61] P. Davidsson and M. Boman. Saving Energy and Providing Value Added Services in Intelligent Buildings: A MAS Approach. In *Proc. of the International Symposium on Agent Systems and Applications and International Symposium on Mobile Agents*, pages 166–177, 2000.

[62] H. Davis, Schrobe R., and P. Szolovits. What is knowledge representation? *AI Magazine*, 14(1):17–33, 1993.

[63] A. Dengel, editor. *Semantische Technologien: Grundlagen - Konzepte - Anwendungen*. Spektrum Akademischer Verlag, Heidelberg, 2012.

[64] Department of Defense. Department of Defense World Geodetic System 1984: Its Definition and Relationships with Local Geodetic Systems. Technical report, National Imagery and Mapping Agency, 2000.

[65] B. Dong, K. P. Lam, Y. C. Huang, and G. M. Dobbs. A comparative study of the IFC and gbXML informational infrastructures for data exchange in computational design support environments. In *Proc. of the International Building Performance Simulation Association Conference*, pages 1530–1537, 2007.

[66] P. Doran. Ontology reuse via ontology modularisation. In *Proc. of KnowledgeWeb PhD Symposium*, June 2006.

[67] K. Dovrtel and S. Medved. Weather-predicted control of building free cooling system. *Applied Energy*, 88(9):3088–3096, September 2011.

[68] B. DuCharme. *Learning SPARQL*. O' Reilly Media, Inc., 1. edition, 2011.

[69] C. Eastman, P. Teicholz, R. Sacks, and K. Liston. *BIM Handbook*. John Wiley & Sons Inc., 2008.

[70] W. K. Edwards and R. E. Grinter. At home with ubiquitous computing: Seven challenges. In *Ubicomp*, Lecture Notes in Computer Science (LNCS), pages 256–272. Springer-Verlag, 2001.

[71] M. Egaña Aranguren. Automatic maintenance of multiple inheritance ontologies. The Ontogenesis Knowledge Blog, Available Online: `http://ontogenesis.knowledgeblog.org/49`, January 2010. Last Accessed: 18. August 2013.

[72] M. Egaña Aranguren, E. Antezana, M. Kuiper, and R. Stevens. Ontology Design Patterns for bio-ontologies: a case study on the Cell Cycle Ontology. *BMC Bioinformatics*, 9:S1, 2008.

[73] M. Egaña Aranguren, A. Rector, E. Antezana, M. Kuiper, and R. Stevens. Upper Level Ontology ODP. Available Online: `http://odps.sourceforge.net/odp/html/Upper_Level_Ontology.html`, 2009. Last Accessed: 12. December 2013.

[74] D. Ejigu, M. Scuturici, and L. Brunie. An Ontology-Based Approach to Context Modeling and Reasoning in Pervasive Computing. In *Proc. of the International Conference on Pervasive Computing and Communications*, pages 14–19, 2007.

[75] EN 13321-2. Open Data communication in building automation, conrols and building management - Home and building electronic systems - Part 2: KNXnet/IP Communication, 2005.

[76] EN 13757-3. Communication Systems for Meters and Remote Reading of Meters - part 3: Dedicated application layer, 2011.

[77] EN 14908. Open Data Communication in Building Automation, Control and Building Management - Control Network Protocol, 2005.

[78] S. Engstrom. Kant's distinction between theoretical and practical knowledge. *Harvard review of philosophy*, 10:49–63, 2002.

[79] European Commission. Commissioner Piebalgs welcomes political agreement on energy performance of buildings. Online Available: http://europa.eu/rapid/press-release_IP-09-1733_en.htm, November 2009. Last Accessed: 17. May 2013 REF-ID: IP/09/1733.

[80] H. Farhangi. The path of the smart grid. *Power and Energy Magazine*, 8(1):18–28, 2010.

[81] A. Fensel, S. Tomic, V. Kumar, M. Stefanovic, S. V. Aleshin, and D. O. Novikov. SESAME-S Semantic Smart Home System for Energy Efficiency. *Informatik-Spektrum*, 36(1):46–57, February 2013.

[82] S. Fenz and A. Ekelhart. Formalizing information security knowledge. In *Proc. of the 4th ACM Symposium on Information, Computer, and Communications Security*, pages 183–194, 2009.

[83] M. Fernández, A. Goméz-Pérez, and N. Juristo. Methontology: From ontological art towards ontological engineering. In *Proc. of the AAAI '97 Symposium on Ontological Engineering*, pages 33–40, 1997.

[84] M. Férnandez López. Overview of methodologies for building ontologies. In *Proc. of the IJCAI-99 workshop on Ontologies and Problem-Solving Methods*, 1999.

[85] C. Fischer. Feedback on household electricity consumption: a tool for saving energy? *Energy Efficiency*, 1(1):79–104, 2008.

[86] S. Franklin and A. Graesser. Is it an agent, or just a program?: A taxonomy for autonomous agents. In *Intelligent Agents III Agent Theories, Architectures, and Languages*, volume 1193 of *Lecture Notes in Computer Science (LNCS)*, pages 21–35, 1997.

[87] J. Fürnkranz. The Role of Qualitative Knowledge in Machine Learning. Report, Austrian Research Institute for Artificial Intelligence, Vienna, 1993. number: TR-93-09.

[88] A. Gangemi. Ontology Design Patterns for Semantic Web Content. In *Proc. of the International Semantic Web Conference*, pages 262–276, 2005.

[89] A. Gangemi. DOLCE+DnS Ultralite. Available Online: `http://ontologydesignpatterns.org/wiki/Ontology:DOLCE%2BDnS_Ultralite`, 2010. Last Accessed: 12. April 2013.

[90] A. Gangemi, C. Catenacci, M. Ciaramita, and J. Lehmann. Modelling Ontology Evaluation and Validation. In *ESWC*, volume 4011, pages 140–154, 2006.

[91] A. Gangemi, N. Guarino, C. Masolo, A. Oltramari, and L. Schneider. Sweetening Ontologies with DOLCE. In *Proc. of the 13th EKAW International Conference*, pages 166–181, 2002.

[92] A. Gangemi and P. Mika. Understanding the semantic web through descriptions and situations. In *Proc. of ODBASE03 Conference*, pages 689–706, 2003.

[93] R. García Gonzáles. *A Semantic Web Approach to Digital Rights Management*. PhD thesis, Universitat Pompeu Fabra, 2005.

[94] D. Gašević, V. Devedžić, and D. Djurić. *Model Driven Engineering and Ontology Development*. Springer-Verlag Berlin Heidelberg, 2. edition, 2009.

[95] gbXML.org. Open Green Building XML Schema: a Building Information Modeling Solution for Our Green World. Available Online: `http://www.gbxml.org/`, January 2013. Current Schema Version: 5.10, Last Accessed: 3 April 2013.

[96] B. Glimm, M. Horridge, B. Parsia, and P. F. Patel-Schneider. A Syntax for Rules in OWL 2. In *Proc. of OWL: Experiences and Directions 2009 (OWLED 2009)*, volume 529, 2009.

[97] L. Gomez, A. Laube, and C. Purr. Ontological Middleware for Dynamic Wireless Sensor Data Processing. In *Proc. of the International Conference on Sensor Technologies and Applications*, pages 145–151, 2009.

[98] A. Goméz-Pérez. Some ideas and examples to evaluate ontologies. In *Proc. of the Conference on Artificial Intelligence for Applications*, pages 299–305, 1995.

[99] M. Grassi, M. Nucci, and F. Piazza. Towards an ontology framework for intelligent smart home management and energy saving. In *Proc. of the International Symposium on Industrial Electronics*, pages 1753–1758, 2011.

[100] M. Grassi, M. Nucci, and F. Piazza. Ontologies for smart homes and energy management: An implementation-driven survey. In *Proc. of the Workshop on Modeling and Simulation of Cyber-Physical Energy Systems*, pages 1–3, 2013.

[101] J. Groppe and W. Mueller. Profile Management Technology for Smart Customizations in Private Home Applications. In *Proc. of the International Workshop on Database and Expert Systems Applications*, pages 226–230, 2005.

[102] T. R. Gruber. Toward principles for the design of ontologies used for knowledge sharing. *Human-Computer Studies*, 43:907–928, 1993.

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

[104] M. Grüninger and M. S. Fox. Towards a Methodology for the Design and Evaluation of Ontologies. In *Workshop on Basic Ontological Issues in Knowledge Sharing*, 1995.

[105] M. Gruninger and J. Lee. Ontology - Applications and Design. *Communications of the ACM*, 45(2):39–41, 2002.

[106] T. Gu, H. K. Pung, and D. Q. Zhang. A Service-oriented Middleware for Building Context-aware Services. *Network and Computer Applications*, 28(1):1–18, 2005.

[107] N. Guarino. Semantic Matching: Formal Ontological Distinctions for Information Organization, Extraction, and Integration. In *Information Technology, International Summer School, SCIE-97*, pages 139–170, 1997.

[108] N. Guarino. Formal Ontology and Information Systems. In *Proc. of the Formal Ontology in Information Systems (FOIS) Conference*, pages 3–15, 1998.

[109] G. Guzman. What is practical knowledge? *Knowledge Management*, 13(4):86–98, 2009.

[110] S. Ha, H. Jung, and Y. Oh. Method to analyze user behavior in home environment. *Personal and Ubiquitous Computing*, 10(2-3):110–121, 2006.

[111] V. Haarslev and R. Möller. RACER System Description. In *IJCAR*, pages 701–706, 2001.

[112] R. Haas. Energy efficiency indicators in the residential sector. *Energy Policy*, 25(7-9):789–802, 1997.

[113] R. Harper, editor. *Inside The Smart Home*. Springer-Verlag London Limited, 2003.

[114] G. Hart and C. Dolbear. *Linked Data: a Geographic Perspective*. CRC Press, Taylor & Francis Group, 2013. pages 241-243.

[115] D. Harvey. Reducing energy use in the buildings sector: measures, costs, and examples. *Energy Efficiency*, 2:139–163, 2009.

[116] J. Hebeler, M. Fisher, R. Blace, A. Perez-Lopez, and M. Dean. *Semantic Web Programming*. John Wiley & Sons Inc., 2009.

[117] S. Helal, W. Mann, H. El-Zabadani, J. King, Y. Kaddoura, and E. Jansen. The Gator Tech Smart House: a programmable pervasive space. *Computer*, 38(3):50–60, March 2005.

[118] J. Hendler. Agents and the Semantic Web. *Intelligent Systems*, 16(2):30–37, 2001.

[119] R. Hierzinger, M. Albu, H. van Elburg, A. J. Scott, A. Łazicki, L. Penttinen, F. Puente, and H. Sæle. European Smart Metering Landscape Report 2012. Available Online: `http://www.energyagency.at/fileadmin/dam/pdf/projekte/klimapolitik/SmartRegionsLandscapeReport2012.pdf`, October 2012. Smart Regions - Deliverable 2.1 Last Accessed: 14.April 2013.

[120] J. Himoff, P. Skobelev, and M. Wooldridge. MAGENTA technology: multi-agent systems for industrial logistics. In *Proc. of the International Conference on autonomous agents and multiagent systems*, pages 60–66, 2005.

[121] P. Hitzler, J. Angele, B. Motik, and R. Studer. Bridging the Paradigm Gap with Rules for OWL. In *W3C Workshop on Rule Languages for Interoperability, 27-28 April 2005, Washington, DC, USA*. W3C, 2005.

[122] P. Hitzler, M. Krötsch, and S. Rudolph. *Foundations of Semantic Web Technologies*. CRC Press, 2010.

[123] R. Hoehndorf. What is an upper level ontology? The Ontogenesis Knowledge Blog, Available Online: `http://ontogenesis.knowledgeblog.org/740`, April 2010. Last Accessed: 19. August 2013.

[124] C.-L. Hor, S. J. Watson, and S. Majithia. Analyzing the Impact of Weather Variables on Monthly Electricity Demand. *IEEE Transactions on Power Systems*, 20(4):2078–2085, November 2005.

[125] M. Horridge et. al. *A Practical Guide To Building OWL Ontologies Using Protégé 4 and CO-ODE Tools*. University of Manchester, 1.3 edition, March 2011.

[126] I. Horrocks, P. F. Patel-Schneider, S. Bechhofer, and D. Tsarkov. OWL Rules: A Proposal and Prototype Implementation. *Journal of Web Semantics*, 3:23–40, 2005.

[127] M. Ibrahim and R. Krawczyk. The Level of Knowledge of CAD Objects within the Building Information Model. In *Proc. of the Conference of the Association for Computer Aided Design In Architecture 2003*, pages 173–177, 2003.

[128] Illuminating Engineering Society (IES). *The Lighting Handbook*. Illuminating Engineering Society, 10 edition, 2011.

[129] F. Iglesias-Vázquez. *Smart Home Control Based on Behavioural Profiles*. PhD thesis, Vienna University of Technology, 2012.

[130] F. Iglesias-Vázquez and W. Kastner. Clustering methods for occupancy prediction in smart home control. In *Proc. of the International Symposium on Industrial Electronics*, pages 1321–1328, 2011.

[131] F. Iglesias-Vázquez and W. Kastner. Detecting user dissatisfaction in ambient intelligence environments. In *Proc. of the International Conference on Emerging Technologies Factory Automation*, page 4 S., 2012.

[132] F. Iglesias-Vázquez, W. Kastner, and M. J. Kofler. Holistic smart home model for air quality and thermal comfort management. *Journal of Intelligent Decision Technologies*, 7(1):23–43, 2013.

[133] S. S. Intille. Designing a Home of the Future. *IEEE Pervasive Computing*, 1(2):76–82, 2002.

[134] ISO 16484-5. Building automation and control systems - Part 5: Data communication protocol, 2007.

[135] ISO 7730. Ergonomics of the thermal environment - Analytical determination and interpretation of thermal comfort using calculation of the PMV and PPD indices and local thermal comfort criteria, 2005.

[136] ISO/IEC 14543-3. Information technology - Home Electronic Systems (HES) Architecture, 2006.

[137] M. Jahn, M. Jentsch, C. R. Prause, F. Pramudianto, A. Al-Akkad, and R. Reiners. The Energy Aware Smart Home. In *Proc. of the 5th International Conference on Future Information Technology (FutureTech)*, pages 1–8, 2010.

[138] S. Jupp, H. E. Parkinson, and J. Malone. Semantic Web Atlas: Putting Gene Expression Data into Biological Context. In *Proc. of the 5th International Workshop on Semantic Web Applications and Tools for Life Sciences (SWAT4LS2012)*, volume 952, 2012.

[139] W. Kastner, M. J. Kofler, and C. Reinisch. Wissensrepräsentation für das adaptive Eigenheim im Kontext von Smart Cities. *Elektrotechnik und Informationstechnik*, 129(4):286–292, 2012.

[140] W. Kastner, G. Neugschwandtner, S. Soucek, and H. M. Newman. Communication systems for building automation and control. *Proc. of the IEEE*, 93(6):1178–1203, 2005.

[141] W. Kastner, S. Soucek, C. Reinisch, and A. Klapproth. *Building and Home Automation*, volume 2, chapter 26, pages 26–1 – 26–15. CRC Press, 2 edition, 2011. Industrial Electronics Handbook, volume 2: Industrial Communication Systems.

[142] L. Khemlani. The IFC Building Model: A Look Under the Hood. Available Online: `http://www.aecbytes.com/feature/2004/IFCmodel.html`, March 2004. AECbytes, Last Accessed: 19. May 2013.

[143] H. S. Kim and J. H. Son. User-pattern analysis framework to predict future service in intelligent home network. In *Proc. of the IEEE International Conference on Intelligent Computing and Intelligent Systems*, volume 2, pages 818–822, 2009.

[144] J.-J. Kim and J. W. Moon. Impact of Insulation on Building Energy Consumption. In *Proc. of the International Building Simulation Conference*, pages 674–680, 2009.

[145] M. Klein, A. Schmidt, and R. Lauer. Ontology-Centred Design of an Ambient Middleware for Assisted Living: The Case of SOPRANO. In *Towards Ambient Intelligence: Methods for Cooperating Ensembles in Ubiquitous Environments, 30th Annual German Conference on Artificial Intelligence*, 2007.

[146] M. J. Kofler, F. Iglesias-Vazquez, and W. Kastner. An ontology for representation of user habits and building context in future smart homes. In *Proc. of the International Workshop: Intelligent Computing in Engineering 2013*, 2013. paper 05.

[147] M. J. Kofler and W. Kastner. A Knowledge Base for Energy-Efficient Smart Homes. In *Proc. of the International Energy Conference*, pages 85–90, 2010.

[148] M. J. Kofler and W. Kastner. Towards an ontology representing building physics for increased energy efficiency in smart home operation. In *Proc. of the Central European Symposium on Building Physics*, pages 51 –58, 2013.

[149] M. J. Kofler, C. Reinisch, and W. Kastner. A Knowledge Representation for Energy Parameters in Sustainable Smart Homes. In *Proceedings of the International Conference on Applied Energy* , pages 1065–1078, 2011.

[150] M. J. Kofler, C. Reinisch, and W. Kastner. An Intelligent Knowledge Representation of Smart Home Energy Parameters. In *Proceedings of the World Renewable Energy Congress (WREC)*, Linköping, Sweden, May 2011. pages 921–928.

[151] M. J. Kofler, C. Reinisch, and W. Kastner. An Ontological Weather Representation for Improving Energy-Efficiency in Interconnected Smart Home Systems. In *Proc. of Applied Simulation and Modelling/Artificial Intelligence and Soft Computing*, Napoli, Italy, 2012. pages 256–263.

[152] M. J. Kofler, C. Reinisch, and W. Kastner. A semantic representation of energy-related information in future smart homes. *Energy and Buildings*, 47(0):169–179, 2012.

[153] W. Kymmell. *Building Information Modeling: Planning and Managing Construction Projects with 4D CAD and Simulations*. McGraw Hill Professional, 2008.

[154] J. Lee. Conflict resolution in multi-agent based Intelligent Environments. *Building and Envrionment*, 45(3):574–585, 2010.

[155] S.-H. Leitner and W. Mahnke. OPC UA - Service-oriented Architecture for Industrial Applications. *Softwaretechnik-Trends*, 26(4), 2006.

[156] D. B. Lenat. CYC: A large-scale investment in knowledge infrastructure. *Communications of the ACM*, 38(11):33–38, November 1995.

[157] N.-S. Liang, L.-C. Fu, and C.-L. Wu. An integrated, flexible, and Internet-based control architecture for home automation system in the Internet era. In *Proc. of the International Conference on Robotics and Automation*, volume 2, pages 1101–1106, 2002.

[158] M. Lutz, J. Sprado, E. Klien, C. Schubert, and I. Christ. Overcoming semantic heterogeneity in spatial data infrastructures. *Computers & Geosciences*, 35(4):739–752, 2009.

[159] J. M. Maestre and E. F. Camacho. Smart home interoperability: the DomoEsi project approach. *International Journal of Smart Home*, 3(3):31–44, 2009.

[160] B. Malinowsky. Building Information Models in Building Automation Systems. Master's thesis, Vienna University of Technology, Institute of Computer Aided Automation, Automation Systems Group, 2009.

[161] J. Malone and H. Parkinson. Reference and Application Ontologies. The Ontogenesis Knowledge Blog, Available Online: `http://ontogenesis.knowledgeblog.org/295`, January 2010. Last Accessed: 24. August 2013.

[162] F. Mattern and C. Floerkemeier. From the Internet of Computers to the Internet of Things. In *From Active Data Management to Event-based Systems and More*, volume 6462 of *Lecture Notes in Computer Science (LNCS)*, pages 242–259. Springer-Verlag, 2010.

[163] A. K. Meier. A Worldwide Review of Standby Power Use in Homes. In *Lawrence Berkeley National Laboratory*, number LBNL-49377, 2001. Available Online: `repositories.cdlib.org/lbnl/LBNL-49377`, Last Accessed: 27. November 2013.

[164] M. Minsky. A Framework for Representing Knowledge. In J. Haugeland, editor, *Mind Design: Philosophy, Psychology, Artificial Intelligence*, pages 95–128. MIT Press, 1981.

[165] Z. Mo and A. Mahdavi. An agent-based simulation-assisted approach to bilateral building systems control. In *Proc. of the 8th International Conference on Building Simulation*, pages 888–894, 2003.

[166] T. Mori, A. Fujii, M. Shimosaka, H. Noguchi, and T. Sato. Typical Behavior Patterns Extraction and Anomaly Detection Algorithm Based on Accumulated Home Sensor Data. In *Proc. of the Future Generation Communication and Networking*, volume 2, pages 12–18, 2007.

[167] T. Mori, A. Takada, Y. Iwamura, and T. Sato. Automatic human life summarization system in sensory living space. In *Proc. of the International Conference on Systems, Man and Cybernetics*, volume 2, pages 1583–1588, 2004.

[168] R. A. Morris, L. Dou, J. Hanken, M. Kelly, D. B. Lowery, B. Ludäscher, J. A. Macklin, and P. J. Morris. Semantic Annotation of Mutable Data. *PLoS ONE*, 2013. to appear, preprint available at: http://sourceforge.net/projects/filteredpush/files/SemAnnMutableDataPreprint/ Last Accessed: 02. October 2013.

[169] B. Motik, U. Sattler, and R. Studer. Query Answering for OWL-DL with Rules. In *Journal of Web Semantics*, pages 549–563. Springer, 2004.

[170] M. Mozer, R. Dodier, D. Miller, M. Anderson, J. Anderson, D. Bertini, M. Bronder, M. Colagrosso, R. Cruickshank, B. Daugherty, et al. The Adaptive House. In *IEE Seminar Digests*, volume 1, page v1:39, 2005.

[171] M. C. Mozer. The Neural Network House: An Environment that Adapts to its Inhabitants. In *Proc. of the American Association for Artificial Intelligence Spring Symposium on Intelligent Environments*, pages 110–114, 1998.

[172] D. Nardi and R. J. Brachman. *The Description Logic Handbook: Theory, Implementation, and Applications*, chapter 1, pages 1–40. Cambridge University Press, 2003.

[173] NBS. National BIM Report 2013. Available Online: `https://www.thenbs.com/pdfs/NBS-NationlBIMReport2013-single.pdf`, 2013. Last Accessed: 19. May 2013.

[174] B. Nemuraite, L. Paradauskas. A Methodology for Engineering OWL 2 Ontologies in Practise Considering their Semantic Normalisation and Completeness. *Electronics and Electrical Engineering*, 120(4):89 – 94, 2012.

[175] T. V. Nguyen, Lim W., H. A Nguyen, Choi D., and C. Lee. Context Ontology Implementation for Smart Home. *Computing in Research Repository (CoRR)*, abs/1007.1273, 2010.

[176] I. Niles and A. Pease. Towards a Standard Upper Ontology. In *Proc. of the International Conference on Formal Ontology in Information Systems*, 2001.

[177] I. Niskanen and J. Kantorovitch. Ontology driven data mining and information visualization for the networked home. In *Proc. of the International Conference on Research Challenges in Information Science*, pages 147–156, 2010.

[178] Norwegian Meteorological Institute. B<locationforecast> - weather forecast for a specified place. Available Online: `http://api.yr.no/weatherapi/locationforecast/1.8/documentation`, June 2010. yr.no Weather Forecast Documentation, Last Accessed: 15. July 2013.

[179] N. F. Noy and D. L. McGuinness. Ontology Development 101: A Guide to Creating Your First Ontology. Technical report, Stanford Knowledge Systems Laboratory, 2001. Technical Report KSL-01-05.

[180] OASIS. OBIX Version 1.1. Available Online: `https://www.oasis-open.org/committees/download.php/48683/oBIX-1-1-spec-wd08.pdf`, March 2013. Working draft 08, Last Accessed: 15. April 2013.

[181] M. Obitko. *Translations between Ontologies in a Multi-Agent System*. PhD thesis, Czech Technical University in Prague, February 2007.

[182] C. K. Ogden and I.A. Richards. *The meaning of meaning*. Trubner & Co, 1923.

[183] F. Oldewurtel, D. Gyalistras, M. Gwerder, C. N. Jones, V. Stauch, B. Lehmann, and M. Morari. Increasing Energy Efficiency in Building Climate Control using Weather Forecasts and Model Predictive Control. In *REHVA World Congress Clima*, 2010.

[184] Ontology Summit 2013 Communique. Towards Ontology Evaluation across the Life Cycle. Available Online: `http://ontolog.cim3.net/OntologySummit/2013/communique.html`, June 2013. Last Accessed: 26. October 2013.

[185] OSGi Alliance. About the OSGi Service Platform. Technical Whitepaper Rev. 4.1, OSGi Alliance, June 2007.

[186] F. Pan. A Temporal Aggregates Ontology in OWL for the Semantic Web. In *Proc. of the AAAI Fall Symposium on Agents and the Semantic Web*, pages 30–37, 2005.

[187] J. Pathak, D. Caragea, and V. G. Honavar. Ontology-Extended Component-Based Workflows-A Framework for Constructing Complex Workflows from Semantically Heterogeneous Software Components. In *Proc. of the International Workshop on Semantic Web and Databases*, pages 41–56. Springer-Verlag, 2004.

[188] A. Pease. Suggested Upper Merged Ontology (SUMO) Website. Available Online: `http://www.ontologyportal.org/`, April 2013. Last Accessed: 12. April 2013.

[189] D. Poole and A. Mackworth. *Artificial Intelligence: Foundations of Computational Agents*. Cambridge University Press, New York, NY, USA, 2010.

[190] C. W. Potter, A. Archambault, and K. Westrick. Building a smarter smart grid through better renewable energy information. In *Proc. of the Power Systems Conference*, pages 1–5, 2009.

[191] M. J. Pratt. Introduction to ISO 10303 - the STEP Standard for Product Data Exchange. *Journal of Computing and Information Science and Engineering*, 1(1):102–103, 2001.

[192] B. Qiao, K. Liu, and C. Guy. A Multi-Agent System for Building Control. In *Proc. of the International Conference on Intelligent Agent Technology*, pages 653–659. IEEE Computer Society, 2006.

[193] M. Quillian. *Semantic Memory*. PhD thesis, Carnegie Institute of Technology, 1966.

[194] D. Randall. Living Inside a Smart Home: A Case Study. In R. Harper, editor, *Inside the Smart Home*, pages 227–246. Springer London, 2003.

[195] P. Rashidi and D. J. Cook. Keeping the resident in the loop: Adapting the smart home to the user. *Transactions on Systems, Man and Cybernetics*, 39(5):949 – 959, 2009.

[196] R. J. Raskin and M. J. Pan. Knowledge representation in the semantic web for Earth and environmental terminology (SWEET). *Computers & Geoscience*, 31(9):1119–1125, 2005.

[197] A. Rector. Representing Specified Values in OWL: “value partitions“ and “value sets“ . Available Online: `http://www.w3.org/TR/swbp-specified-values/`, May 2005. W3C Working Group Note 17. May 2005, Last Accessed: 19. July 2013.

[198] A. L. Rector. Modularisation of Domain Ontologies Implemented in Description Logics and related formalisms including OWL. In *Proc. of the International Conference on Knowledge Capture*, pages 121–128, 2003.

[199] C. Reinisch. *A Framework for Distributed Intelligence for an Energy Efficient Operation of Smart Homes*. PhD thesis, Vienna University of Technology, 2012.

[200] C. Reinisch, W. Granzer, F. Praus, and W. Kastner. Integration of Heterogeneous Building Automation Systems using Ontologies. In *Proc. of the Annual Conference of the IEEE Industrial Electronics Society*, pages 2736–2741, 2008.

[201] C. Reinisch and W. Kastner. Agent based Control in the Smart Home. In *Proc. of the Annual Conference of the IEEE Industrial Electronics Society*, pages 334–339, 2011.

[202] C. Reinisch, M. J. Kofler, F. Iglesias, and W. Kastner. ThinkHome Energy Efficiency in Future Smart Homes. *EURASIP Journal on Embedded Systems*, 2011. Article ID: 104617.

[203] C. Reinisch, M. J. Kofler, and W. Kastner. ThinkHome: A Smart Home as Digital Ecosystem. In *Proc. of the International Conference on Digital Ecosystems and Technologies*, pages 256–261, 2010.

[204] D. Retkowitz and M. Pienkos. Ontology-based configuration of adaptive smart homes. In *Proc. of Workshop on Reflective and adaptive middleware*, pages 11–16, 2008.

[205] D. Retkowitz and M. Stegelmann. Dynamic Adaptability for Smart Environments. In *Proc. of the International Conference on Distributed Applications and Interoperable Systems*, volume 5053 of *Lecture Notes in Computer Science*, pages 154–167. Springer-Verlag, 2008.

[206] R. J. Robles and T. Kim. Applications, Systems and Methods in Smart Home Technology: A Review. *International Journal of Advanced Science and Technology*, 15:37–48, 2010.

[207] B. Rodriguez-Castro, M. Ge, and M. Hepp. Alignment of Ontology Design Patterns: Class As Property Value, Value Partition and Normalisation. In *OTM Conferences (2)*, volume 7566 of *Lecture Notes in Computer Science*, pages 682–699, 2012.

[208] J. Rowley. The wisdom hierarchy: representations of the DIKW hierarchy. *Information Science*, 33(2):163–180, 2007.

[209] S. Runde, H. Dibowski, A. Fay, and K. Kabitzsch. A semantic requirement ontology for the engineering of building automation systems by means of OWL. In *Proc. of the IEEE International Conference on Emerging Technologies and Factory Automation*, pages 413–420, 2009.

[210] M. Ruta, F. Schioscia, E. Di Schiascio, and G. Loseto. Semantic-Based Enhancement of ISO/IEC 14543-3 EIB/KNX Standard for Building Automation. *Transactions on Industrial Informatics*, 7(4):731–739, 2011.

[211] F. Santana, D. Schober, Z. Medeiros, F. Freitas, and S. Schulz. Ontology patterns for tabular representations of biomedical knowledge on neglected tropical diseases. *Bioinformatics*, 27(13):i349–i356, July 2011.

[212] K. Sato. Context sensitive interactive systems design: a framework for representations of contexts. In *Proc. of the International Conference on Human-Computer Interaction*, pages 1323–1327, 2003.

[213] D. Molitor R. Sattler, U. Calvanese. *The Description Logic Handbook: Theory, Implementation, and Applications*, chapter 4, pages 137–177. Cambridge University Press, 2003.

[214] G. Schneider and J. P. Winters. *Applying Use Cases - A Practical Guide*. Addison Wesley Longman, Inc., 1998.

[215] F. Shayeganfar, A. Anjomshoaa, and A. M. Tjoa. A Smart Indoor Navigation Solution Based on Building Information Model and Google Android. In *Proc. of the International Conference on Computers Helping People with Special Needs*, pages 1050 – 1056, 2008.

[216] D. Shotton, C. Catton, and G. Klyne. Ontologies for Sharing, Ontologies for Use. The Ontogenesis Knowledge Blog, Available Online: `http://ontogenesis.knowledgeblog.org/312`, January 2010. Last Accessed: 20. August 2013.

[217] G. Singla, D. J. Cook, and M. Schmitter-Edgecombe. Recognizing independent and joint activities among multiple residents in smart environments. *Journal of Ambient Intelligence and Humanized Computing*, 1(1):57–63, 2010.

[218] F. Singleton. The Beaufort scale of winds - its relevance, and its use by sailors. *Weather*, 63(2):37–41, February 2008.

[219] E. Sirin, B. Parsia, B. 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.

[220] A. Sloman. Why We Need Many Knowledge Representation Formalisms. In *Proc. of the BSC Expert Systems Conference*, pages 163–183, 1985.

[221] B. Smith. *The Blackwell Guide to the Philosophy of Computing and Information*. Blackwell Philosophy Guides. Blackwell Publishing, 2004.

[222] J. C. Smith, M. R. Milligan, E. A. DeMeo, and B. Parsons. Utility Wind Integration and Operating Impact State of the Art. *IEEE Transactions on Power Systems*, 22(3):900–908, August 2007.

[223] I. Sokolov. Utilizing gbXML with AECOsim Building Designer and speedikon. White paper, Bentley Building Products, October 2011.

[224] L. Sommaruga, A. Perri, and F. Furfari. DomoML-env: an ontology for human home interaction. In *Proc. of Italian Semantic Web Workshop*, pages 14–16, 2005.

[225] Y. Son, T. Pulkkinen, K. Moon, and C. Kim. Home Energy Management System based on Power Line Communication. *IEEE Transactions on Consumer Electronics*, 56:1380–1385, 2010.

[226] J. F. Sowa. Conceptual graphs. In *Information Processing in Mind and Machine*, pages 39–44. Addison-Wesley, 1984.

[227] Standard Upper Ontology Working Group. IEEE P1600.1 Standard Upper Ontology Working Group (SUO WG) Home Page. Available Online: http://suo.ieee.org/, December 2003. Last Accessed: 17. November 2013.

[228] P. Staroch. A Weather Ontology for Predictive Control in Smart Homes. Master's thesis, University of Technology, Vienna, 2013.

[229] T. G. Stavropoulos, D. Vrakas, D. Vlachava, and N. Bassiliades. BOnSAI: a smart building ontology for ambient intelligence. In *Proc. of the International Conference on Web Intelligence, Mining and Semantics*, page Article No. 30, 2012.

[230] S. Steyskal. Defining an Actor Ontology for Increasing Energy Efficiency in Smart Homes. Master's thesis, University of Technology, Vienna, 2014. to be published.

[231] H. R. Straub. *Das interpretierende System*. Z/I/M Verlag, 2001.

[232] H. Stuckenschmidt. *Ontologien: Konzepte, Technologien und Anwendungen (Informatik Im Fokus)*. Springer, Berlin, 1 edition, 2009.

[233] A. L. Stumpf, H. Kim, and E. M. Jenicek. Early Design Energy Analysis Using Building Information Modeling Technology. Final report, US Army Corps of Engineers, 2011. Corporate Author: Engineer Research and Development Center Champaign IL Construction Engineering Research Lab.

[234] S. Taduri, G. T. Lau, K. H. Law, and J. P. Kesan. A patent system ontology for facilitating retrieval of patent related information. In *Proc. of the International Conference on Theory and Practice of Electronic Governance*, pages 146–157, 2012.

[235] M. Takada, S. Tano, M. Iwata, and T. Hashiyama. A proposed home agent architecture to infer user feeling from user action pattern. In *Proc. of the International Conference on Systems, Man, and Cybernetics*, pages 4818–4824, 2006.

[236] J. Tauberer. What is RDF and what is it good for? Available Online: http://www.rdfabout.com/intro/, 2008. Last Accessed: 27. November 2013.

[237] D. Tsarkov and I. Horrocks. FaCT++ Description Logic Reasoner: System Description. In *Proc. of the International Joint Conference on Automated Reasoning*, volume 4130 of *Lecture Notes in Artificial Intelligence*, pages 292–297. Springer, 2006.

[238] D. Tuhus-Dubrow and M. Krarti. Comparative Analysis of Optimization Approaches to Design Building Envelope for Residential Buildings. *ASHRAE Transactions*, 115(2):554, 2009.

[239] E. Turban and J. Aronson. *Decision Support Systems and Intelligent Systems*. Prentice Hall, Inc., 1988.

[240] T. Ueno, R. Inada, O. Saeki, and K. Tsuji. Effectiveness of an energy-consumption information system for residential buildings. *Applied Energy*, 83(8):868 – 883, 2006.

[241] M. Uschold and M. Grüninger. Ontologies Principles, Methods and Applications. *Knowledge Sharing and Review*, 11(2):93–136, June 1996.

[242] M. Uschold and M. King. Towards a Methodology for Building Ontologies. In *Workshop on Basic Ontological Issues in Knowledge Sharing*, 1995.

[243] S. van den Elshout, H. Bartelds, H. Heich, and K. Léger. Citeair II - Common Information to European Air - CAQI Air quality index. Project report, European Union - European Regional Development Fund, 2012.

[244] R. van Gerwen, S. Jaarsma, and R. Wilhite. Smart Metering. Available Online: `http://www.leonardo-energy.org/sites/leonardo-energy/files/root/pdf/2006/SmartMetering.pdf`, July 2006. Last Accessed: 20. January 2014.

[245] M. Venugopal, C. Eastman, and J. Teizer. An Ontological Approach to Building Information Model Exchanges in the Precast/Pre-Stressed Concrete Industry. In *Proc. of the Construction Research Congress 2012*, pages 1114–1123, 2012.

[246] E. Vildjiounaite, O. Kocsis, V. Kyllönen, and B. Kladis. Context-Dependent User Modelling for Smart Homes. In *Proc. of the International Conference on User Modeling*, pages 345–349, 2007.

[247] S. von Roon, T. Gobmaier, and M. Huck. Demand Side Management in Haushalten - Methoden zur Potenzialanalyse und Kostenabschätzung. Technical report, Forschungsstelle für Energiewirtschaft e.V., 2010. Available Online: `http://www.ffe.de/publikationen/fachartikel/296-demand-side-management`, Last Accessed: 06. November 2013.

[248] W3C. OWL Web Ontology Language Reference. Available Online: `http://www.w3.org/TR/owl-ref/`, February 2004. W3C Recommendation 10 February 2004, Last Accessed: 24. January 2013.

[249] W3C. RDF Primer. Available Online: `http://www.w3.org/TR/rdf-primer/`, February 2004. W3C Recommendation 10 February 2004, Last Accessed: 24. January 2013.

[250] W3C. RDF Vocabulary Description Language 1.0: RDF Schema. Available Online: `http://www.w3.org/TR/rdf-schema/`, February 2004. W3C Recommendation 10 February 2004, Last Accessed: 25. January 2013.

[251] W3C. RDF/XML Syntax Specification (Revised). Available Online: `http://www.w3.org/TR/REC-rdf-syntax/`, February 2004. W3C Recommendation 10 February 2004, Last Accessed: 27. November 2013.

[252] W3C. Resource Description Framework (RDF): Concepts and Abstract Syntax. Available Online: `http://www.w3.org/TR/rdf-concepts/`, February 2004. W3C Recommendation 10 February 2004, Last Accessed: 27. November 2013.

[253] W3C. SWRL: A Semantic Web Rule Language Combining OWL and RuleML. Available Online: http://www.w3.org/Submission/SWRL/, May 2004. W3C Member Submission 21 May 2004, Last Accessed: 4. February 2013.

[254] W3C. Time Ontology in OWL. Available Online: http://www.w3.org/TR/2006/WD-owl-time-20060927/, September 2006. W3C Working Draft 27 September 2006, Last Accessed: 12. April 2013.

[255] W3C. Semantic Web, and Other Technologies to Watch. Available Online: http://www.w3.org/2007/Talks/0130-sb-W3CTechSemWeb/#(24), January 2007. Last Accessed: 24. November 2013.

[256] W3C. SPARQL Query Language for RDF. Available Online: http://www.w3.org/TR/rdf-sparql-query/, January 2008. W3C Recommendation 15 January 2008, Last Accessed: 8. February 2013.

[257] W3C. OWL 2 Web Ontology Language. Available Online: http://www.w3.org/TR/2009/WD-owl2-overview-20090327/, March 2009. W3C Working Draft 27 March 2009, Last Accessed: 24. November 2013.

[258] W3C. OWL 2 Web Ontology Language. Available Online: http://www.w3.org/TR/owl2-overview/, December 2012. W3C Recommendation 11 December 2012, Last Accessed: 2. February 2013.

[259] W3C. OWL 2 Web Ontology Language - Manchester Syntax (Second Edition). Available Online: http://www.w3.org/TR/owl2-manchester-syntax/, December 2012. W3C Working Group Note 11 December 2012, Last Accessed: 18. May 2013.

[260] W3C. OWL 2 Web Ontology Language Profiles (Second Edition). Available Online: http://www.w3.org/TR/owl2-profiles/, December 2012. W3C Recommendation 11 December 2012, Last Accessed: 25. November 2013.

[261] W3C. OWL/Implementations. Available Online: http://www.w3.org/2001/sw/wiki/OWL/Implementations, August 2013. Last Accessed: 24. November 2013.

[262] W3C. RIF Core Dialect. Available Online: http://www.w3.org/TR/2013/REC-rif-core-20130205/, February 2013. W3C Recommendation 5 February 2013, Last Accessed: 24. November 2013.

[263] W3C. SPARQL 1.1 Update. Available Online: http://www.w3.org/TR/sparql11-update/, March 2013. W3C Recommendation 21 March 2013, Last Accessed: 23 March 2013.

[264] W3C. Turtle - Terse RDF Triple Language. Available Online: http://www.w3.org/TR/turtle/, February 2013. W3C Candidate Recommendation 19 February 2013, Last Accessed: 27. November 2013.

[265] C. A. Welty. *An Integrated Representation for Software Development and Discovery*. PhD thesis, Rensselaer Poytechnic Institute, 1995.

[266] P. Wolf, A. Schmidt, and M. Klein. Applying Semantic Technologies for Context-Aware AAL Services: What we can learn from SOPRANO. In *Workshop on Applications of Semantic Technologies, Informatik 2009*, volume 9 of *Lecture Notes in Informatics*. GI, 2009.

[267] M. Wooldridge. *An Introduction to MultiAgent Systems*. Wiley Publishing, 2. edition, 2009.

[268] M. Wooldridge and N. R. Jennings. Intelligent agents: Theory and practice. *Knowledge Engineering Review*, 10:115–152, 1995.

[269] A. Yurchyshyna, C. Faron-Zucker, N. L. Thanh, and A. Zarli. Towards an Ontology-enabled Approach for Modeling the Process of Conformity Checking in Construction. In *Forum at the CAiSE'08 Conference*, pages 21–24, 2008.

[270] V. M. Zavala, E. M. Constantinescu, and M. Anitescu. Economic impacts of advanced weather forecasting on energy system operations. In *Proc. of the Innovative Smart Grid Technologies Conference*, pages 1–7, 2010.

[271] L. Zemmouchi-Ghomari and A. R. Ghomari. Translating Natural Language Competency Questions into SPARQL Queries: A Case Study. In *Proc. of the International Conference on Building and Exploring Web Based Environments*, pages 81 – 86, 2013.

[272] D. Zhang, T. Gu, and X. Wang. Enabling Context-aware Smart Home with Semantic Web Technologies. *Human-friendly Welfare Robotic Systems*, 6(4):12–20, 2005.

[273] H. Zhang, F.-Y. Wang, and Y. Ai. An OSGi and agent based control system architecture for smart home. In *Proc. of the International Conference on Networking, Sensing and Control*, pages 13–18, March 2005.