Goal-based Workflow Adaptation for Role-based Resources in the Internet of Things

Dissertation

zur Erlangung des akademischen Grades
Doktoringenieur (Dr.-Ing.)

vorgelegt an der
Technischen Universität Dresden
Fakultät Informatik

eingereicht von

Dipl.-Medieninf. Steffen Huber
geboren am 06.02.1986 in Gießen

Betreuender Hochschullehrer
Prof. Dr.-Ing. Thomas Schlegel
(Hochschule Karlsruhe - Technik und Wirtschaft, Deutschland)

Zweitgutachter
Prof. Dr.-Ing. Dennis Pfisterer
(Universität zu Lübeck, Deutschland)

Fachreferent
Prof. Dr. rer. nat. habil. Dr. h. c. Alexander Schill
(Technische Universität Dresden, Deutschland)

Tag der Verteidigung: 28.11.2017
Dresden, den 10.09.2018

Confirmation

I confirm that I independently prepared this thesis with the title *Goal-based Workflow Adaptation for Role-based Resources in the Internet of Things* and that I used only the references and auxiliary means indicated in the thesis.

Dresden, 10.09.2018

Dipl.-Medieninf. Steffen Huber

Acknowledgement

First, I would like to express my deep gratitude and special thanks to my thesis advisor Thomas Schlegel for giving me the opportunity to advance both as a researcher and person. He supported me in the pursuit of my research goals from the very beginning and taught me more than I could ever give him credit for here. Also, I am thankful for the comments and feedback from my second adviser Alexander Schill over the course of my work. Additionally, I would like to thank my external advisor Dennis Pfisterer for the crucial support in the final phase of my thesis.

I am very thankful to my former colleagues at the SEUS chair for the warm welcome and many fruitful debates over the last years. Special thanks to Ronny Seiger who introduced me to the specifics of workflows in Cyber-physical Systems and had a large impact on the progression of my thesis as well. He often played the important role of the devil's advocate in generating new ideas and concepts for the contributions of my thesis. Moreover I like to thank Christine Keller for her great support and comments especially in the final year of my thesis. Very special thanks to Tomas Karnagel for providing comments and motivational support in the final phase of my thesis.

I also like to thank all the past and current members of the RoSI research training group for creating such a unique research atmosphere. The many in-depth discussions about the purpose of roles and context but also off-topic chats were always a pleasure. Special thanks go to my RoSI colleagues Ismail, Jan, Max, Stephan, Thomas, and Tobias for their helpful comments. In addition, I like to thank Thomas Springer for his comments on the evaluation of my work. I am also grateful to the DFG for providing the funding for such an extraordinary research training group.

I am also very thankful to Vasileios Theodorou for introducing me to the field of goal-based modeling. He provided many helpful insights for applying the goal-modeling approach to high-level user goals for workflow activities. In addition, I am grateful for the excellent work of André Kühnert in helping with the implementation of the Semantic Access Layer middleware. I also wish to acknowledge the opportunity given by my colleagues Gordon Lemme and Hajo Wiemer to finish the writing of my thesis while starting a new employment.

I am grateful to all my friends and family members who endured me in this often stressful time. Most importantly, I wish to thank my wife Kathrin for her love, support and endurance over the last years and my wonderful daughter Eva for being an endless source of joy.

Abstract

In recent years, the Internet of Things (IoT) has increasingly received attention from the Business Process Management (BPM) community. The integration of sensors and actuators into Process-Aware Information Systems (PAIS) enables the collection of real-time data about physical properties and the direct manipulation of real-world objects. In a broader sense, IoT-aware workflows provide means for context-aware workflow execution involving virtual and physical entities. However, IoT-aware workflow management imposes new requirements on workflow modeling and execution that are outside the scope of current modeling languages and workflow management systems. *Things* in the IoT may vanish, appear or stay unknown during workflow execution, which renders their allocation as workflow resources infeasible at design time. Besides, capabilities of *Things* are often intended to be available only in a particular real-world context at runtime, e. g., a service robot inside a smart home should only operate at full speed, if there are no residents in direct proximity. Such contextual restrictions for the dynamic exposure of resource capabilities are not considered by current approaches in IoT resource management that use services for exposing device functionalities. With this work, we aim at providing the modeling and runtime support for defining such restrictions on workflow resources at design time and enabling the dynamic and context-sensitive runtime allocation of *Things* as workflow resources. To achieve this goal, we propose contributions to the fields of resource management, i. e., *resource perspective*, and workflow management in the Internet of Things (IoT), divided into the *user perspective* representing the workflow modeling phase and the *workflow perspective* representing the runtime resource allocation phase.

In the resource perspective, we propose an ontology for the modeling of *Things*, *Roles*, capabilities, physical entities, and their context-sensitive interrelations. The concept of *Role* is used to define non-exclusive subsets of capabilities of *Things.* A *Thing* can play a certain *Role* only under certain contextual restrictions defined by Semantic Web Rule Language (SWRL) rules. At runtime, the existing relations between the individuals of the ontology represent the current state of interactions between the physical and the cyber world. Through the dynamic activation and deactivation of *Roles* at runtime, the behavior of a *Thing* can be adapted to the

current physical context. In the user perspective, we allow workflow modelers to define the goal of a workflow activity either by using semantic queries or by specifying high-level goals from a Tropos goal model. The goal-based modeling of workflow activities provides the most flexibility regarding the resource allocation as several leaf goals may fulfill the user specified activity goal. Furthermore, the goal model can include additional Quality of Service (QoS) parameters and the positive or negative contribution of goals towards these parameters. The workflow perspective includes the Semantic Access Layer (SAL) middleware to enable the transformation of activity goals into semantic queries as well as their execution on the ontology for role-based *Things*. The SAL enables the discovery of fitting *Things*, their allocation as workflow resources, the invocation of referenced IoT services, and the continuous monitoring of the allocated *Things* as part of the ontology.

We show the feasibility and added value of this work in relation to related approaches by evaluation within several application scenarios in a smart home setting. We compare the fulfillment of quantified criteria for IoT-aware workflow management based on requirements extracted from related research. The evaluation shows, that our approach enables an increase in the context-aware modeling of *Things* as workflow resources, in the query support for workflow resource allocation, and in the modeling support of activities using *Things* as workflow resources.

Publications

This thesis is partially based on the following peer-reviewed publications:

- Steffen Huber, Ronny Seiger, André Kühnert and Thomas Schlegel. Using semantic queries to enable dynamic service invocation for processes in the internet of things. In *Semantic Computing (ICSC), 2016 IEEE International Conference on*, pages 214–221, Feb 2016.
- Steffen Huber, Ronny Seiger, André Kühnert, Vasileios Theodorou and Thomas Schlegel. Goal-based semantic queries for dynamic processes in the internet of things. In *International Journal of Semantic Computing (IJSC), 2016*, volume 10, No. 2, pages 269–293.
- Steffen Huber, Ronny Seiger, André Kühnert and Thomas Schlegel. A context-adaptive workflow engine for humans, things and services. In *Proceedings of the 2016 ACM International Joint Conference on Pervasive and Ubiquitous Computing: Adjunct*, pages 285–288, Sep 2016.
- Ronny Seiger, Steffen Huber, and Thomas Schlegel. Proteus: An integrated system for process execution in cyber-physical systems. In Khaled Gaaloul, Rainer Schmidt, Selmin Nurcan, Sergio Guerreiro, and Qin Ma, editors, *Enterprise, Business-Process and Information Systems Modeling*, volume 214 of *Lecture Notes in Business Information Processing*, pages 265–280. 2015.
- Ronny Seiger, Steffen Huber, and Peter Heisig. PROtEUS++: A self-managed IoT workflow engine with dynamic service discovery. In *Proceedings of the 9th Central-European Workshop on Services and their Composition, Lugano, Switzerland, February 13-14, 2017.*, CEUR Workshop Proceedings, CEUR-WS.org 2017.

The following peer-reviewed publications cover work that is closely related to the content of the thesis, but not contained herein:

- Ronny Seiger, Steffen Huber, Peter Heisig, and Uwe Aßmann. Enabling self-adaptive workflows for cyber-physical systems. In *Enterprise, Business-Process and Information Systems Modeling: 17th International Conference, BPMDS 2016, 21st International Conference, EMMSAD 2016, Held at CAiSE 2016, Ljubljana, Slovenia, June 13-14,2016 , Proceedings*, pages 3–17. 2016.
- Ronny Seiger, Steffen Huber, and Thomas Schlegel. Toward an execution system for self-healing workflows in cyber-physical systems. In *Software & Systems Modeling*, pages 1–22. 2016.
- Ronny Seiger, Steffen Huber, and Thomas Schlegel. An execution system for self-healing workflows in cyber-physical systems. In *Lecture Notes in Informatics Proceedings, Series of the Gesellschaft fr Informatik (GI)*, Volume P-267, pages 75–76. 2017.

Contents

1 Introduction

In recent years, the *Internet of Things (IoT)* gained more and more attention from both industry and academia alike. The prospects of interconnected intelligent real world objects that bridge the gap between the physical and the cyber world, raised the interest in IoT-technology to the state of *Peak of Inflated Expectations*, referring to the Gartner Hype Cycle [Pan16] illustrated in Fig. 1.1. Gartner predicts the IoT to reach the *Plateau of Productivity* in 5 to 10 years, while the overall number of IoT-enabled devices is expected to exceed 20 billion in the year 2020 [vdM15]. This includes the number of consumer devices, which is forecasted to be larger than 13 billion in the year 2020 [vdM15].

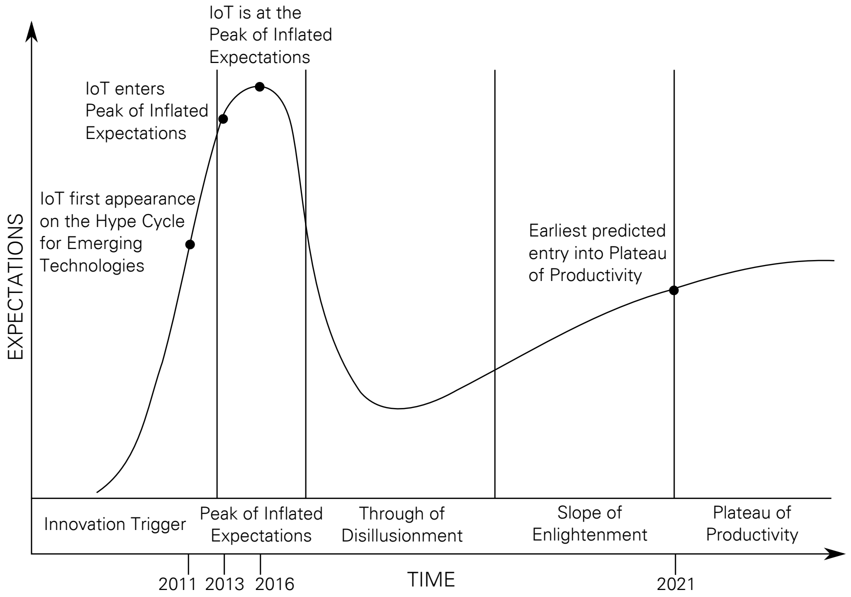


Figure 1.1: The Gartner Hype Cycle for the Internet of Things [Pan16].

Consequently, IoT-enabled consumer devices will become *ubiquitous* and the underlying technologies will reach a *productive* state of application in the near future.

After nearly three decades since Mark Weiser published his vision of *Ubiquitous Computing* [Wei91], its wide-spread adoption and realization may be closely coupled to these emerging IoT technologies. *Ubiquitous Computing* aims at providing assistance and fulfilling user goals in a highly adaptive and context-sensitive environment. These ad-hoc tasks often need context-information both from the cyber and the physical perspective of the user's environment. IoT Services can encapsulate sensors, actuators, and more complex compounds of mixed sensor, actuator, and computing systems like robots. In addition to the software context and effects of a *Ubiquitous Computing* application, these IoT Services can provide real-world context data and execute physical tasks. Therefore, IoT is a key enabling technology for *Ubiquitous Computing.*

Beyond the physical technology and the service abstraction, *Ubiquitous Computing* relies on the seamless execution of user-centric system behavior [Wei91]. This includes the interaction between the user and ubiquitous system components in a goal-oriented manner. Today, workflows are increasingly used to specify and execute system behavior in Cyber-physical Systems (CPS) and Systems of Systems (SoS). The use of a Workflow Management System (WfMS) in these application areas promises controllable, reusable and user-friendly specification of recurring tasks. Ad-hoc Workflows are specially designed to cope with the dynamic nature of smart environments. However, research in ad-hoc workflows is mostly focused on structural adaptation. In contrast, activity-level adaptation is still necessary to cope with the dynamic nature of an ubiquitous system. Activity adaptation comprises service discovery, dynamic resource allocation and subsequent context-sensitive service invocation. Using workflows and a WfMS to model and execute behavior in ubiquitous systems facilitates the vision of *Ubiquitous Computing.* Therefore, the guiding research question is:

> *How can a WfMS fulfill user goals on the activity level while adapting to context-sensitive IoT resources at runtime?*

In this thesis, we will discuss this question in detail and provide a solution that enables the modeling and execution of goal-based context-sensitive workflows in the IoT integrating *Things* as workflow resources. In the following we present a short introduction into IoT-aware workflows in Section 1.1, illustrate the challenges and necessary steps for context-sensitive activity-level workflow adaptation in Section 1.2, define the research aim and objective of this thesis in Section 1.3, list the developed contributions in Section 1.4, and conclude the introduction by presenting the structure of this thesis 1.5.

1.1 Background

In this section[1], we briefly introduce the concept of IoT-aware workflows and their prospected benefits. The IoT hype has increasingly received attention from the Business Process Management (BPM) community. The integration of sensors and actuators into *Process-Aware Information Systems (PAIS)* enables the collection of real-time data about workflow resources and the direct manipulation of real world objects. In a broader sense, IoT-enabled workflows provide means for context-aware workflow execution involving virtual and physical entities. We refer to this class of workflows as *IoT-aware workflows* [MRM13]. In general, IoT device capabilities can be classified as either sensing or actuating capabilities [BBDL$^+$13]. IoT-aware workflows integrate sensing capabilities in the form of special activities representing sensing tasks. These tasks generate data objects that may trigger the control flow within the workflow. As these concepts are not directly supported by BPMN 2.0, several extensions to BPMN's control-flow and data perspective have been proposed [MRM13, MRH15, GEPF11]. Actuator capabilities can be accessed by using IoT services, e. g., through service-based method calls within activities. However, these approaches assume that required workflow resources are identifiable at design time. Workflow modelers have to consider the runtime context of devices and *Things* and already know the respective resource identifiers at design time. While this is possible in controlled environments, it becomes infeasible for large-scale ubiquitous systems as the availability of IoT devices may vary significantly at runtime [RSDS12, KSS$^+$10].

1.2 Motivation

To illustrate the difference between cyber and physical workflow resources, we provide an application scenario of an IoT-aware workflow in the Smart Home domain. Figure 1.2 illustrates an automatic emergency call workflow. It defines the following system behavior: After the workflow has started, the WfMS listens for events that indicate an unresponsive resident. In this case, the WfMS tries to contact the resident. Depending on the response, the workflow either finishes immediately or triggers an automatic emergency call, leading to the treatment of the possibly injured resident. From this example, several requirements arise for using workflows in a smart home setting and in general IoT scenarios. Under the assumption, that the shown workflow resources, e. g., Smart Watch ID:1, are part of the workflow

---

[1] The following sections are partially based on [HSK$^+$16] and [HSS16]

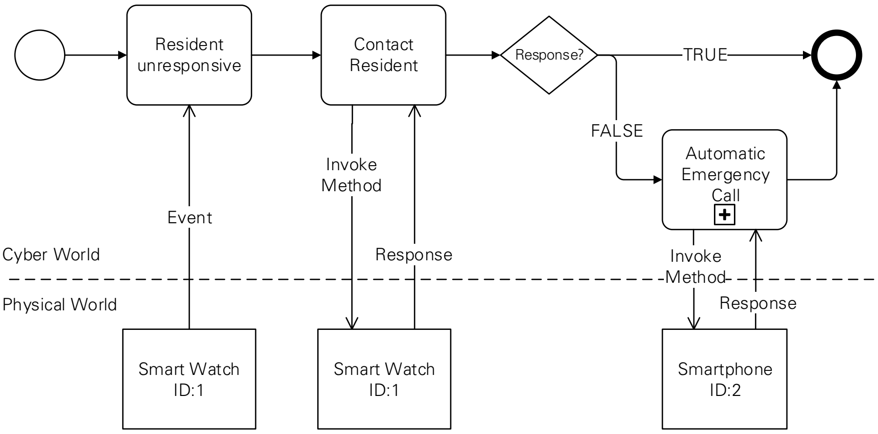


Figure 1.2: Automatic emergency call scenario workflow in BPMN.

model, the modeler used implicit knowledge about the IoT devices and their context. Whenever there occurs a mismatch at runtime between the workflow model and the actual context, the workflow may halt or produce unwanted system behavior. For example, in case the resident does not wear the Smart Watch which detected a *false positive* situation, the resident is in fact responsive but cannot be contacted by the WfMS. In this case, an unwanted automatic emergency call is triggered. Furthermore, if a *true positive* detection of a resident in need of medical assistance occurred but the modeled workflow resource, e. g., Smartphone ID:2, for calling the emergency is unavailable at this moment, the workflow will halt even if other devices with the same *capability* are available. From this simple example follows:

- Knowledge of **IoT device capabilities** and their **context** has to be modeled explicitly.
- Workflow resources have to be **allocated at runtime** depending on the requirements of the activity and runtime context.
- The **requirements of activities** have to be modeled explicitly.

Due to the runtime volatility and context-sensitive capabilities of *Things*, their integration as workflow resources poses several problems. First, as *Things* exist in and can manipulate the real world their produced data has to be interpreted in the location and time context of the measurement. In addition, actuators can change their location in the real world, rendering design time allocation of such resources infeasible. In general, all IoT service invocations are context-dependent and need to be selected and invoked according to their current runtime context, which is not supported by current resource allocation approaches within existing WfMS. An extensive requirements analysis for IoT-aware workflow management regarding the integration of *Things* as workflow resources is provided in Chapter 3.

1.3 Aim and Objective

The overall goal of IoT-aware workflows is to integrate sensor data and actuators into the proven concepts of BPM. Our goal is to use the proven concepts of workflow modeling and WfMS to define recurring system behavior in highly adaptive ubiquitous environments using *Things* as workflow resources. Ultimately, we aim to combine WfMS and IoT technology to at least partially implement Mark Weiser's vision of *Ubiquitous Computing*.

1.3.1 Research Questions and Scope

The following research questions guide the structure of this work and partially result from the example scenario in Section 1.2:

- **RQ1** What are the differences between workflow resources in the cyber and the physical world from the perspective of a WfMS?
- **RQ2** How to model context-sensitive capabilities of workflow resources in the IoT?
- **RQ3** How can a WfMS cope with the uncertainty of IoT resource availability during workflow execution?
- **RQ4** How to minimize failures during workflow execution caused by unavailable resources?
- **RQ5** How to allow for flexible as well as correct workflow execution in the IoT and what are the tradeoffs?

We limit the scope of this work to the activity-level adaptation of workflows and do not investigate structural workflow adaptation, e. g., control-flow adaptation.

Furthermore, we do not aim at developing a WfMS. However, we use a WfMS to execute our modeled workflows and to trigger the context-sensitive resource allocation.

1.3.2 Research Goals

To overcome the disadvantages of existing approaches, this thesis aims at fulfilling the following goal:

> *Enabling the goal-based modeling and execution of*
> *IoT-aware workflows by using a context-sensitive resource*
> *allocation mechanism for* Things *at runtime.*

This goal entails several subgoals to be fulfilled by the contribution of this work. First, we want to provide a **modeling foundation** for expressing the context-sensitive relations between *Things* and their capabilities. Second, these relations need to reflect the **runtime state** of the physical and cyber world interactions such that physical context changes as well as virtual resource allocations can be reflected in the model. Third, a workflow modeler needs to be able to **specify goals** of activities. Finally, these goals need to provide enough information for a subsequent **resource allocation** using the modeling foundation.

1.4 Contribution

This thesis includes contributions within the fields of resource and workflow management in the IoT. The resource management, we propose an **ontology for role-based *Things*** with context-sensitive capabilities. In this modeling context, the role-based approach allows for a fitting concept of defining context-sensitive subsets of capabilities and enabling their activation and deactivation dynamically at runtime. Furthermore, we propose an **update mechanism for context changes** in the developed ontology based on SWRL rules. In the workflow management perspective, we contribute a **workflow metamodel extension** to allow for the modeling of goal-based workflow activities referring to a domain-specific Tropos goal model. In addition, we propose the **Semantic Access Layer (SAL)** as a middleware between the WfMS and the actual IoT services. The SAL provides the functionalities for resource discovery and allocation based on an ontological knowledge base.

1.5 Outline

The remainder of this work is illustrated in Fig. 1.3. Chapter 2 provides the basic concepts and context of this work. This includes an introduction to the Internet of Things, context and role modeling, goal modeling as well as existing workflow modeling languages, WfMS and their ability to execute workflows in the IoT. In Chapter 3, existing requirements for integrating *Things* as workflow resources are extracted and discussed from related work. After limiting the requirements to the scope of this work defined by the research question from Chapter 1.3.1, we review their fulfillment by state-of-the-art approaches in IoT-aware workflow management. Then, we present our approach and contribution to fill the identified research gaps. Chapter 4 introduces three perspectives, resource, user, and workflow, to structure our contributions and presents the concept of adaptive activity-level workflow adaptation in the IoT. Chapter 5 introduces the semantic model for representing *Thing*, their context-dependent capabilities and roles as part of the resource perspective. In addition, the modeling of user goals and workflow activities is presented. Chapter 6 provides an overview of the system architecture for adaptive workflow activities in the IoT. Chapter 7 describes the implementation of the middleware layer including aspects of all three perspective. Here, the resource perspective includes the knowledge base, the user perspective includes the goal model and the workflow perspective the actual activities resulting in service calls to the middleware. Chapter 8 presents evaluation results in terms of performance measures and adaptation characteristics which are then compared to the requirements Chapter 3.1 as well as the research questions in Chapter 1.3.1. Chapter 9 discusses the overall results and contributions of this work regarding the research questions and goals. Furthermore, the limitations of our approach is discussed in detail. Finally, Chapter 10 summarizes the results of this thesis, gives an outlook on future work and concludes.

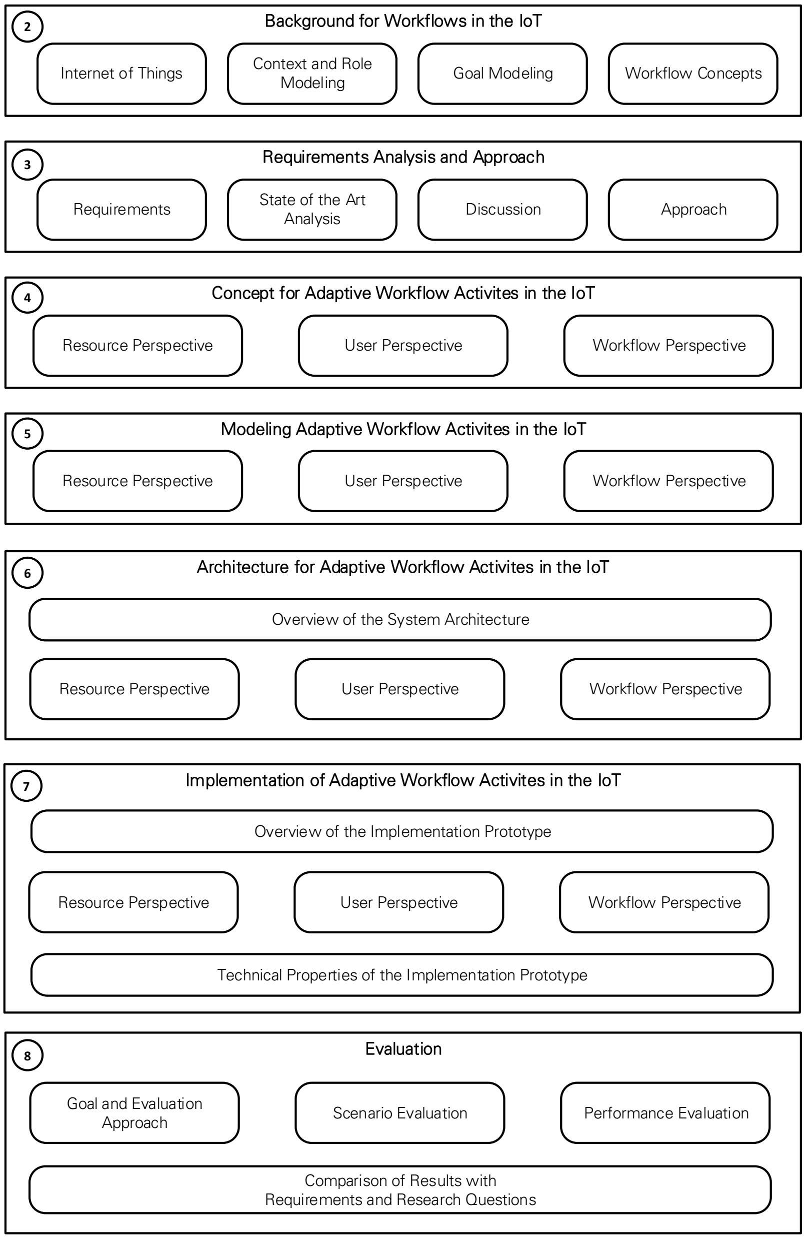

2
Background for Workflows in the IoT
Internet of Things
Context and Role Modeling
Goal Modeling
Workflow Concepts
3
Requirements Analysis and Approach
Requirements
State of the Art Analysis
Discussion
Approach
4
Concept for Adaptive Workflow Activites in the IoT
Resource Perspective
User Perspective
Workflow Perspective
5
Modeling Adaptive Workflow Activites in the IoT
Resource Perspective
User Perspective
Workflow Perspective
6
Architecture for Adaptive Workflow Activites in the IoT
Overview of the System Architecture
Resource Perspective
User Perspective
Workflow Perspective
7
Implementation of Adaptive Workflow Activites in the IoT
Overview of the Implementation Prototype
Resource Perspective
User Perspective
Workflow Perspective
Technical Properties of the Implementation Prototype
8
Evaluation
Goal and Evaluation Approach
Scenario Evaluation
Performance Evaluation
Comparison of Results with Requirements and Research Questions

Figure 1.3: The structure of this thesis including chapter numbers.

2 Background for Workflows in the IoT

In this chapter [1], we present relevant basic concepts and some preliminary definitions related to the contributions of this thesis. First, we introduce IoT as the technological foundation for IoT-aware workflows. We then present the basic concepts of modeling context with *Semantic Web* technologies and role-based modeling languages. Finally, we introduce relevant goal modeling and workflow concepts. We divided the concepts into three perspectives of IoT-aware workflow management. Figure 2.1 gives an overview of the relevant concepts within the resource, user, and workflow perspectives. The resource perspective is rooted within the research field of resource management while the user and workflow perspectives are rooted in the field of workflow management. In this context, the user perspective represents the workflow modeler as an integral part of the workflow modeling phase within workflow management.

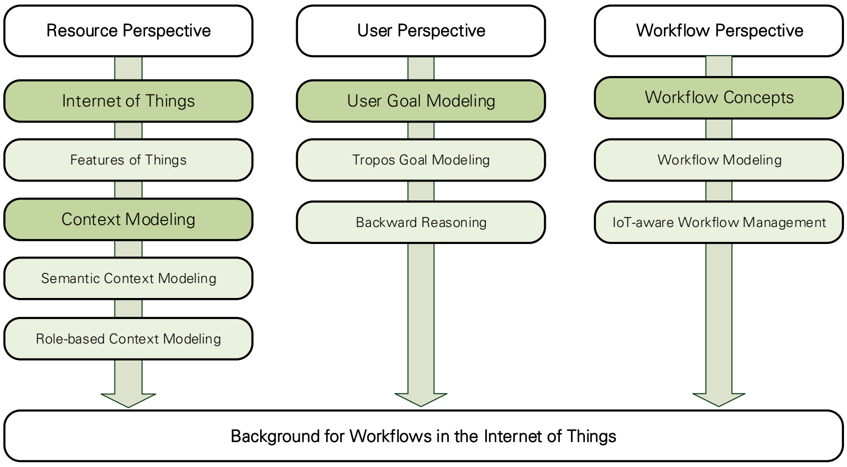


Figure 2.1: Overview of Topics in the Background Chapter.

---

[1] This chapter is partially based on [HSK$^+$16]

2.1 Resource Perspective

The resource perspective includes the introduction to the IoT providing the field of application of this work. This includes the introduction to typical features of *Things* as well as the main aspects of the IoT vision. Furthermore, the context and role modeling are presented as a foundation for the modeling of context-sensitive capabilities of *Things*.

2.1.1 Internet of Things

The term *Internet of Things* semantically means “a world-wide network of interconnected objects uniquely addressable, based on standard communication protocols” [INF08]. Therefore, the IoT comprises a large number of heterogeneous devices with communication abilities, i. e., *Things*. These can be active participants in business, information and social processes exchanging information sensed about the environment while reacting autonomously to the physical world [SGFW10]. In the IoT vision, humans will be completely immersed in the world of technology [Bor14]. Therefore, *Things* will be ubiquitous, following the ubiquitous computing paradigm [Wei91], and can provide services by cooperating and interacting. *Things* are defined by their basic technical requirements for communicating, sensing and acting. Miorandi et al. specified *Things* as smart objects with the following features [MSDPC12]:

- A physical embodiment and a set of associated physical features.
- A minimal set of communication functionalities, such as the ability to be discovered and to accept incoming messages and reply to them.
- A unique identifier.
- Association to at least one name and one address. The name is a human-readable description of the object and can be used for reasoning purposes. The address is a machine-readable string that can be used to communicate to the object.
- Some basic computing capabilities. This can range from the ability to match an incoming message to a given footprint (as in passive RFIDs) to the ability of performing rather complex computations, including service discovery and network management tasks.

- Optional means to sense physical phenomena (e. g., temperature, light, electromagnetic radiation level) or to trigger actions having an effect on the physical reality (actuators).

Example domains for the application of IoT technology include smart homes, smart cities, the manufacturing industry, healthcare, agriculture, the energy sector, robotics and most user-oriented ubiquitous computing scenarios. Figure 2.2 shows an excerpt of an exemplary IoT reference model that highlights the relevance of identification through tags and the ability to sense and to act upon a physical entity [BBDL$^+$13]. Here, a physical entity is represented by virtual entity associated to a IoT service. The service can expose a resource which in turn is composed of devices. These devices include actuators, tags, and sensors. They can contain themselve to represent any hierarchical structure. An Actuator can act on a physical entity which in turn can be monitored by a sensor. The *Thing* concept is not represented explicitly but included in the intersection of the concepts physical entity, virtual entity and on-device resource. Capabilities of *Things* are represented directly via the hosted devices and exposed to applications by IoT services.

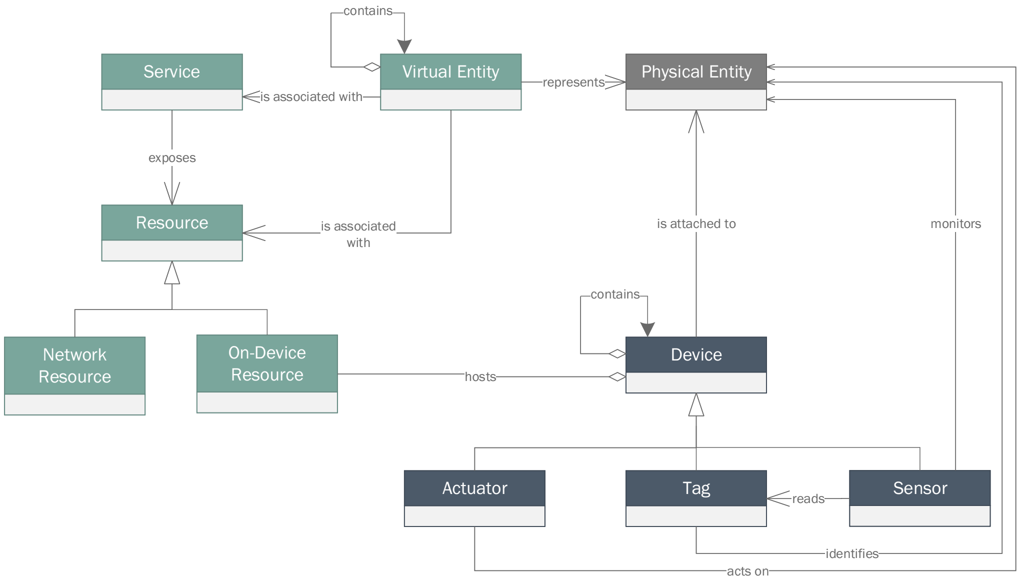


Figure 2.2: Example IoT reference model [BBDL$^+$13].

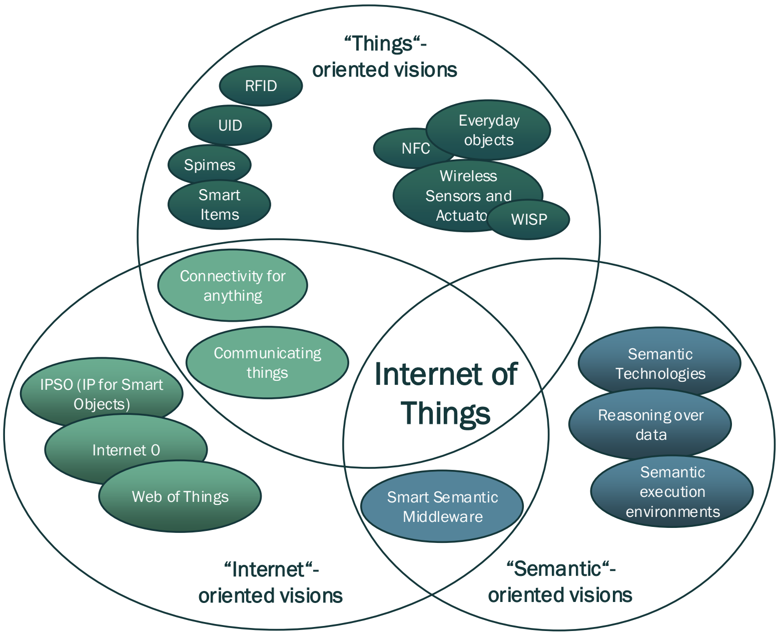


Figure 2.3: Aspects of the IoT notion [AIM10].

Figure 2.3 shows the concept of IoT as an intersection of three perspectives: *Things*-oriented, *Internet*-oriented and *Semantic*-oriented [AIM10].

The *Things* perspective covers the technical aspects of identifiyng, sensing and acting. Prominent technologies for augmenting objects with an Unique Identifier (UID) are Radio Frequency Identification (RFID) and Near Field Communication (NFC). A RFID system consists of at least one active or passive transponder, called *tag*, and one tag reader. While active tags support longer ranges for data transmission, passive tags have a much higher life span and use the power from the electromagnetic field of the reader. The tag stores the UID and other metadata for wireless data transfer and can be attached to various kinds of objects, including workpieces in manufacturing and products in an entrepot. NFC builds upon RFID technology but allows for bidirectional data transfer between two NFC enabled devices. Beyond the identification of real world objects, *Things* are often able to sense and/or manipulate their environment. Furthermore, technologies from the Wireless Sensor Network (WSN) domain are used to facilitate the mobility of such devices.

Regarding the communication abilities of *Things*, the *Internet* perspective of the IoT provides several useful standards and protocols like the Internet Protocol for Smart Objects (IPSO) [DV08] or the *Internet 0* to route IP over any given physical transmission medium [KG04]. There exists a multitude of IoT related protocols for most Open Systems Interconnection model (OSI model) layers, e. g., Internet Protocol Version 6 (IPv6) [Dee98] as an internet layer with 2^{128} IP addresses to support the predicted increase of connected devices, and IPv6 over Low power Wireless Personal Area Network (6LoWPAN) to enable wireless data transmission with minimal power consumption [Mul07].

The third perspective on the IoT includes *Semantic Web* technologies to allow for the interpretation of and reasoning about the gathered data as well as the triggering of IoT-mediated actions in the real world. One of the benefits of modeling *Things* with *Semantic Web* technologies is the gained interoperability of highly heterogeneous devices. Our approach also includes these technologies for modeling *Things* in relation to their real world context with the help of roles. The foundations of these concepts are introduced in the following.

2.1.2 Context and Role Modeling

In 2001, Dey provided the following context definition closely related to ubiquitous systems: “Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves.” [Dey01]. As IoT components exist in, and interact with the physical world, they inherently have associated context parameters, e. g., spatial context. In addition, *Things* with sensing capabilities can act as sensors for physical context phenomena. Actuator capabilities can even trigger a change in the physical context. Regarding the workflow resource perspective, *Things* have to be treated with special consideration for their contextual parameters and capabilities. These have to be modeled explicitly such that a WfMS can use the available context information to find the fitting IoT resources for a specific activity. Otherwise, a workflow execution may cause unwanted side effects. Thus, a workflow execution within the IoT domain can be categorised as context-aware computing, following the definition: “A system is context-aware if it uses context to provide relevant information and/or services to the user, where relevancy depends on the users task.” [Dey01]. Regarding the approaches for IoT resources modeling within this work, we will introduce both to the semantic and role-based context modeling.

Semantic Context Modeling

Within this work, we use the term *semantic context model* for describing a context model based on semantic web technology, i. e., ontology [2]. The term *ontology* stems from the field of philosophy and is an instrument of description, where the "targets for description are the things in themselves in any domain, and the relations existing among them." [Zúñ01]. However, it is "neither reducible to, nor identical with language or its formalism." [Zúñ01]. In contrast, in the domain of information systems, an ontology is regarded as "a document or file that formally defines the relations among terms" [BLHL+01]. It represents a basic component of the *Semantic Web* and is characterised by a taxonomy and a set on inference rules [BLHL+01]. Like a Unified Modeling Language (UML)[3] model [RJB04], the taxonomy of an ontology defines "classes of objects and relations among them." [BLHL+01]. Inference rules can express knowledge about relations and objects of the taxonomy. By applying an inference rule to an ontology, hidden or *implicit* knowledge can be made explicit in the form of new instances of objects and relations. In the following, we present an example ontology for *Things* to illustrate the OWL concepts and their possibilities for the representation of and reasoning about contextual knowledge. Figure 2.4 shows a graphical representation of a semantic context model for *Things*.

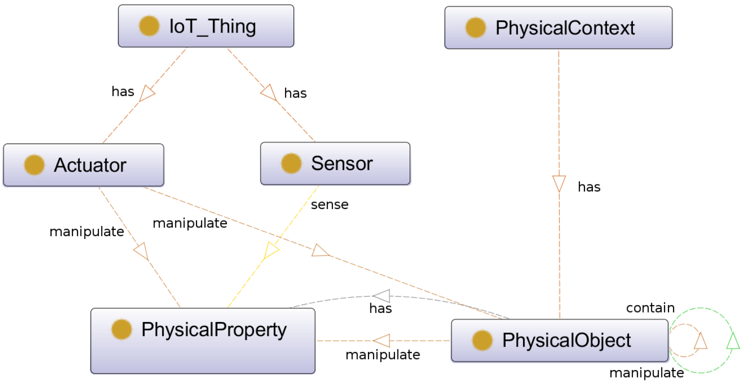


Figure 2.4: Example semantic context model for *Things*.

---

[2] Within this work, we use the term *ontology* synonymously to models specified with the Web Ontology Langauge (OWL) 2.0.

[3] "UML is an industry standard modeling language with a rich graphical notation, and comprehensive set of diagrams and elements. It is used to specify, visualize, modify, construct and document the artifacts of an object-oriented software-intensive system under development." [Lee12]

Here, only the *terminological component* (TBox) of the model is shown. It includes classes and relations, much like the M1 level of the Meta-Object Facility [4]. Relations include object, data and annotation properties. The instance level (M0) is represented by the *assertion component* (ABox). Ontology instances are called *Individual*. The informal description of the model is as follows: A *Thing* has some sensor and/or some actuator. The sensor can sense a physical property of a real-world object. The actuator may manipulate a physical property or a real-world object. An object can manipulate another object or a physical property and may contain other objects. Groups of objects can belong to different physical contexts [5]. Furthermore, relations in an ontology can be specified using formal semantic characteristics: *Reflexivity, transitivity, symmetry, functionality, equivaleny, disjointness, inversity.*

```
<Ontology>
...
        <SubObjectPropertyOf>
        <ObjectProperty IRI=”http://www.semanticweb.org/role_based_iot_ontology#
            manipulate”/>
        <ObjectProperty abbreviatedIRI=”owl:topObjectProperty”/>
    </SubObjectPropertyOf>
        <TransitiveObjectProperty>
        <ObjectProperty IRI=”http://www.semanticweb.org/role_based_iot_ontology#
            manipulate”/>
    </TransitiveObjectProperty>
</Ontology>
```

Listing 2.1: Example transitive object property axiom

Listing 2.1 shows an excerpt of the example ontology, defining the transitive object property *manipulate*. Regarding the example ontology from Figure 2.4, the transitivity of the *manipulate* relation is useful for the expression of the following behavior: An *Actuator* can manipulate a *PhysicalProperty* via a *PhysicalObject*. The following Table 2.1 includes the descriptions of OWL object properties [6][7]:

---

[4] http://www.omg.org/mof/

[5] For the full ontology specification, please refer to the appendix Section 10.2.

[6] Syntax is following the Turtle - Terse RDF Triple Language Schema, available online at: https://www.w3.org/TR/turtle/

[7] OWL2 Syntax online available at: https://www.w3.org/TR/owl2-syntax/

Table 2.1: Overview of OWL 2.0 object properties.

Object Property	OWL Syntax	Description
Reflexivity	owl:ReflexiveProperty *antonym:* owl:IrreflexiveProperty	A reflexive object property P states that each individual is connected by P to itself. Therfore, (x,x) is an instance of P.
Transitivity	owl:TransitiveProperty	If property P is a transtitive property, then if (x,y) is an instance of P and (y,z) is an instance of P, (x,z) is also an instance of P.
Symmetry	owl:SymmetricProperty *antonym:* owl:AsymmetricProperty	If property P is a symmetric property, then if (x,y) is an instance of P, (y,x) is also an instance of P.
Functionality	owl:FunctionalProperty *antonym:* owl:InverseFunctionalProperty	If property P is a functional property, then if x,y are distinct individuals, there can be at most one instance (x,y) of P. If (x,y) and (x,z) are instances of P, then y=z.
Equivalency	owl:EquivalentProperty	If object properties P1 and P2 are equivalent, then $P1 => P2$ and $P2 => P1$ hold.
Disjointness	owl:DisjointProperty	If object properties P1 and P2 are disjoint, then if (x,y) is an inctance of P1 and (x,y) is an instance of P2, x=y. Therefore, no individual x can be connected to an individual y by both P1 and P2.
Inversity	owl:inverseOf	If object property P1 is the inverse of object property P2, then if (x,y) is an instance of P1, (y,x) is an instance of P2.

To illustrate the benefits of reasoning over a semantic context model, we provide the following simple example in the smart home domain, using the example ontology in Figure 2.4. The scenario includes a smart curtain, with an actuator for closing and opening, and a sensor for detecting ambient light. Therefore, we used the example ontology to create the following individuals, as shown in Figure 2.5.

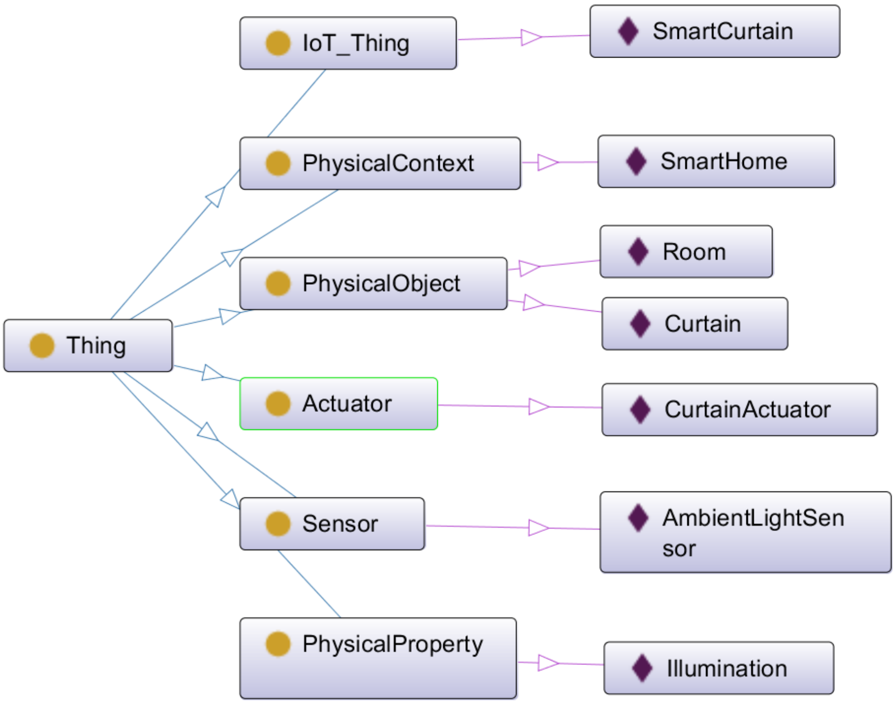


Figure 2.5: Individuals of example semantic context model for *Things*.

Furthermore, we modeled the following relations between these individuals: The *SmartHome* has a *Room* which contains a *Curtain* and has an *Illumination*. The *Curtain* can manipulate the *Illumination*. The *SmartCurtain* has an *AmbientLightSensor* for sensing the *Illumination* and a *CurtainActuator* for manipulating the *Curtain*. As we modeled the manipulation relation as transitive, a reasoner should deduce that the *CurtainActuator* manipulates the *Illumination*. As mentioned before, the main benefit of using a semantic context model is the possibility to use reasoning and inference rules.

```
<ObjectPropertyAssertion>
        <ObjectProperty IRI=”http://www.semanticweb.org/role_based_iot_ontology#
            manipulate”/>
        <NamedIndividual IRI=”http://www.semanticweb.org/role_based_iot_ontology#
            CurtainActuator”/>
        <NamedIndividual IRI=”http://www.semanticweb.org/role_based_iot_ontology#
            Curtain”/>
</ObjectPropertyAssertion>
<ObjectPropertyAssertion>
        <ObjectProperty IRI=”http://www.semanticweb.org/role_based_iot_ontology#
            manipulate”/>
        <NamedIndividual IRI=”http://www.semanticweb.org/role_based_iot_ontology#
            Curtain”/>
        <NamedIndividual IRI=”http://www.semanticweb.org/role_based_iot_ontology#
            Illumination”/>
</ObjectPropertyAssertion>
```

Listing 2.2: Example transitive object property assertion

Listing 2.2 shows the explicitly modeled axioms for relating the *CurtainActuator* with the *Curtain* and the *Curtain* with the *Illumination* via the manipulate relation. After running the HermiT [8] reasoner, an axiom was created for relating the *CurtainActuator* with the *Illumination* via the manipulate relation, as shown in Listing 2.3. Therefore, a smart home application which integrates this ontology as a knowledge base can use the inferred axiom and find an actuator (service) to manipulate the illumination of the room.

```
<ObjectPropertyAssertion>
        <ObjectProperty IRI=”http://www.semanticweb.org/role_based_iot_ontology#
            manipulate”/>
        <NamedIndividual IRI=”http://www.semanticweb.org/role_based_iot_ontology#
            CurtainActuator”/>
        <NamedIndividual IRI=”http://www.semanticweb.org/role_based_iot_ontology#
            Illumination”/>
</ObjectPropertyAssertion>
```

Listing 2.3: Example inferred object property assertion

---

[8] HermiT v1.3.8.3 - online available at http://www.hermit-reasoner.com/

Besides the interpretation of object property assertions, a rule engine can execute DL-safe [9] rules on top of an ontology. These SWRL rules express additional knowledge on top of the taxonomy and may create new axioms. We illustrate the application of SWRL rules in the following example: At first, we add two new object property assertions to the ontology in Figure 2.4. These are the transitive *affect* and its inverse relation *monitor*. They express that an actuator may affect a sensor by altering a physical property sensed by the sensor. In turn, a sensor can monitor the effects of an actuator. However, we do not include these axioms *statically* into the existing individuals in Figure 2.5, as in a dynamic ubiquitous system, these relations are subject to constant change. Instead, we define the following DL-safe SWRL rule:

$$manipulate(?x, ?z) \wedge sense(?y, ?z) \rightarrow affect(?x, ?y)$$

The rule states that, if (x,z) are related by the *manipulate* relation and (y,z) are related by the *sense* relation, then x is *affecting* y. A reasoner also creates the inverse *monitor* relation. After exetung the SWRL rule on the ontology, the desired axioms are created by the reasoner and are shown in the Listing 2.4. This rule may be executed arbitrarily at runtime to reflect the actual state of the modeled context.

```
<ObjectPropertyAssertion>
        <ObjectProperty IRI=”http://www.semanticweb.org/role_based_iot_ontology#
            affect”/>
        <NamedIndividual IRI=”http://www.semanticweb.org/role_based_iot_ontology#
            CurtainActuator”/>
        <NamedIndividual IRI=”http://www.semanticweb.org/role_based_iot_ontology#
            AmbientLightSensor”/>
</ObjectPropertyAssertion>
<ObjectPropertyAssertion>
        <ObjectProperty IRI=”http://www.semanticweb.org/role_based_iot_ontology#
            monitor”/>
        <NamedIndividual IRI=”http://www.semanticweb.org/role_based_iot_ontology#
            AmbientLightSensor”/>
        <NamedIndividual IRI=”http://www.semanticweb.org/role_based_iot_ontology#
            CurtainActuator”/>
</ObjectPropertyAssertion>
```

Listing 2.4: Example inference with SRWL rule.

---

[9] Description Logic (DL)-safe rules are a decidable subset of SWRL rules restricted to known individuals. They are used to contain the decidability of OWL2 DL.

Besides reasoning, a semantic context model specified with OWL, Resource Description Framework (RDF) [10] or Resource Description Framework Schema (RDFS) [11] supports the powerful query language SPARQL Protocol and RDF Query Language (SPARQL). SPARQL queries can be useds to retrieve information about the Terminological component (TBox) and the Assertion component (ABox). To illustrate the application of SPARQL, we specified the following query for retrieving all (actuator, sensor) tuples, that are related by the inferred *affect* relation.

```
PREFIX rdf: <http://www.w3.org/1999/02/22−rdf−syntax−ns#>
PREFIX iot: <http://www.semanticweb.org/role_based_iot_ontology#>
SELECT ?actuator ?sensor
WHERE {?actuator rdf:type iot:Actuator.
          ?sensor rdf:type iot:Sensor.
          ?actuator iot:affect ?sensor}
```

Listing 2.5: Example SPARQL query.

As shown in Listing 2.5, the tuples (actuator, sensor) are selected based on the existence of the relation *iot:affect*. As intended, the execution of the SPARQL query results in the finding of one tuple (CurtainActuator, AmbientLightSensor). The presented technologies and concepts are employed in this work for modeling context-sensitive *Things* in the IoT. By reasoning, inference rules and querying, a knowledge base designed within the OWL also provides tool support for evaluation.

Role-based Context Modeling

The term *Role* has ambiguous definitions from a variety of domains. The Cambridge Dictionary defines *role* as “the position or purpose that someone or something has in a situation, organization, society, or relationship” [Pre17] in addition to the definition in the context of acting: “an actor’s part in a film or play” [Pre17]. Furthermore, the Merriam-Webster dictionary defines *role* as “a socially expected behavior pattern usually determined by an individual’s status in a particular society” [MW17].

All of these definitions include aspects, which define a *role* as a mostly temporary, context-dependent behaviour of someone or something. In the field of computer

---

[10] RDF supports simple triple based (subject, predicate, object) statements and is based on the Extensible Markup Language (XML).

[11] RDFS is a language for specifying groups of RDF statements, e. g., classes.

science, roles have gained general awareness as an important first class modeling concept in several domains [Ste00]. One of the first explicit notion of role in computer science was introduced in the field of data modeling by Bachman and Daya in 1977 [BD77]. They motivated the introduction of roles by stating that files from file records "typically deal with employees, customers, patients, or students, all of which are role types" [BD77]. They each represent some *aspect* of entities of the real world and therefore should not be represented by entities themselves. Furthermore, Bachman and Daya explain that "the reason for the confusion is understood when it is realized that neither the roles of the real world nor the entities of the real world are a subset of the other" [BD77]. This separation has been made explicit by distinguishing role types and natural types by the notions of *rigidity* and *foundness* [Gua92]. The term *rigidity* is closely related to the identity of a concept. A natural type is rigid, i. e., an instance of a natural type exists on its own, not depending on a relation to any other instances of role or natural types. Furthermore, the instance cannot drop the natural type without loosing its identity [Ste00], e. g., *Bird* is a natural type and an instance of this natural type will alway be a bird. However, if a bird is caged it may play the role of a *Pet*. This role is depending on the *foundation* of the relation between the bird and its owner. Therefore, the role *Pet* is founded and lacks rigidity, because it cannot exist on its own without the bird and the person that plays the role of a pet owner. Figure 2.6 shows the example as an UML diagram. The *canPlay* relation is regarded as part of the M1 level of the Meta Object Facility (MOF). It is meant to express, that instances of the associated natural and role types can be bound by a *play* relation at runtime. The *play* relation is therefore part of the M0 level of the MOF.

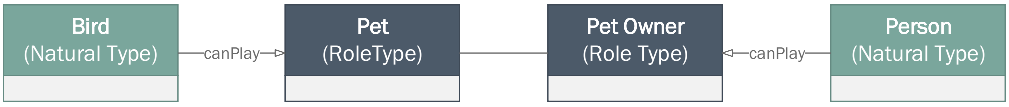


Figure 2.6: Example natural types and role types.

Steimann surveyed existing approaches on roles in object-oriented as well as conceptual modeling and identified 15 features as a classification scheme for role-based modeling [Ste00]. The following Table 2.2 gives an overview of the 15 classifying features. In the context of this work, they will be used to identify the relevant features for modeling role-based *Things* in the IoT.

Table 2.2: Role modeling features by Steimann, extracted from [Ste00].

Nr.	Definition
1	A role comes with its own properties and behaviour.
2	Roles depend on relationships.
3	An object may play different roles simultaneously.
4	An object may play the same role several times, simultaneously.
5	An object may acquire and abandon roles dynamically.
6	The sequence in which roles may be acquired and relinquished can be subject to restrictions.
7	Objects of unrelated types can play the same role.
8	Roles can play roles.
9	A role can be transferred from one object to another.
10	The state of an object can be role-specific.
11	Features of an object can be role-specific.
12	Roles restrict access.
13	Different roles may share structure and behaviour.
14	An object and its roles share identity.
15	An object and its roles have different identities.

The classification scheme has been extended by Kühn et al. to include 11 more features which are focused on the relations between roles [KLG$^+$14]. Kühn et al. introduced the notion of *compartments* as contextual templates for role collaboration [KLG$^+$14]. Table 2.3 shows the added role modeling features.

Table 2.3: Role modeling features by Kühn et al., extracted from [KLG$^+$14].

Nr.	Definition
16	Relationships between roles can be constrained.
17	There may be constraints between relationships.
18	Roles can be grouped and constrained together.
19	Roles depend on compartments.
20	Compartments have properties and behaviors.
21	A role can be part of several compartments.
22	Compartments may play roles like objects.
23	Compartments may play roles which are part of themselves.
24	Compartments can contain other compartments.
25	Different compartments may share structure and behavior.
26	Compartments have their own identity.

As some of the presented features are contradicting, there exists no role-based modeling language which supports all 26 features. In addition, Kühn et al. developed a metamodel family for role-based modeling [KLG+14]. They also provide a modeling tool for feature selection and metamodel generation of a role-based modeling language [KBRA16].

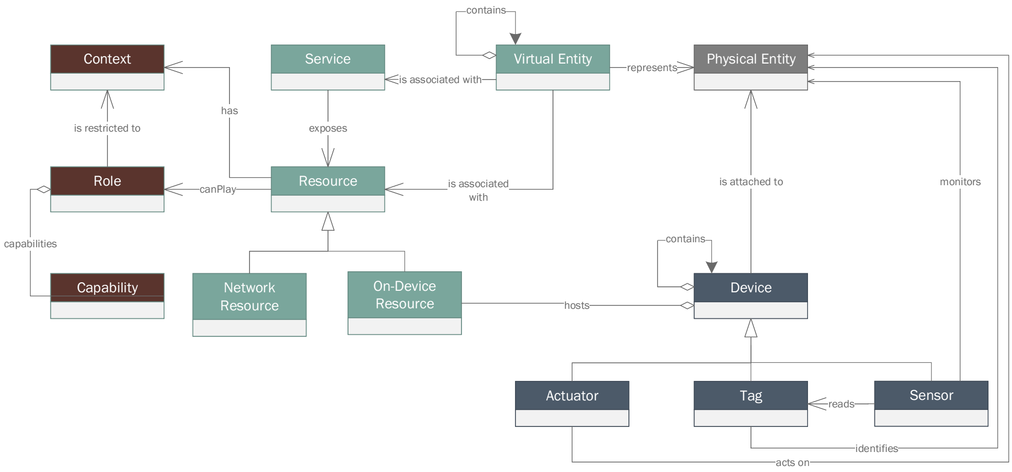


Figure 2.7: Extended role-based IoT reference model, based on [BBDL+13].

In relation to this work, roles can be used to represent contextual capabilities of *Things*. For example, a service robot inside a smart home may only operate at full speed, if there are no residents in direct proximity. A role model can group subsets of all capabilities of a *Thing* and define contextual constraints for their activation. Figure 2.7 shows an extension of the IoT reference model to include roles, capabilities and context restrictions.

2.2 User Perspective

The user perspective is related to the workflow modeling phase in workflow management. Here, users are workflow modelers using activites and control as well as data flow to define a certain logic or behaviour within an *IoT* application scenario. Some approaches consider workflows as programming languages within the IoT domain [SSOK13, GEPF11].

2.2.1 Goal Modeling

Goal modeling is an approach to requirements engineering and widely used in the requirements analysis phase of a software product [KL04]. Goal models can support all phases of requirements engineering, i. e., requirements elicitation, requirements negotiation, requirements specification, and requirements validation, by defining an easy to *understand* graph of interconnected goals, representing conflicting as well as supporting stakeholder requirements. Besides this main application domain, goal models have been used to model and guide the behaviour of self-adaptive and context-aware systems [ADG09, MS14b]. In this work, we use goal models to capture the relations between high-level user goals and low level IoT service invocations. Therefore, users can specify workflow activity goals which are then translated into service invocations considering the runtime context.

2.2.2 Tropos Goal Modeling Language

We provide an introduction to the Tropos goal modeling methodology and motivate its application in the context of IoT-aware process modeling. Tropos is an agent-oriented software development methodology, based on the concepts of agent, goal, and their relations [BPG$^+$04, CKM02]. Tropos uses *goal graphs* to represent interrelations between these concepts. A *goal graph* is defined as a pair $\langle \mathcal{G}, \mathcal{R} \rangle$, where $\mathcal{G}$ is a set of nodes representing the goals and $\mathcal{R}$ is a set of goal relations [SGM04]. The *goal graph* is subject to the following two restrictions: (1) each goal has at most one incoming Boolean relation; (2) every loop contains at least one non-Boolean relation arc [SGM04]. In [GMS05], Tropos is extended with a formal goal model to relate functional and non-functional requirements of the system-to-be. The formal goal model developed in [GMNS03, SGM04] enables the performance of two kinds of analytical reasoning tasks. On the one hand, *forward reasoning* can check if all root goals are fulfilled based on a set of fulfilled leaf goals. On the other hand, *backward reasoning* can find a set of leaf goals that fulfill all root goals. In this work, we are mainly interested in *backward reasoning* as it allows the selection of leaf goals (i. e., goals referring to IoT device capabilities) based on certain root goals (i. e., activity goals). In addition, the backward reasoning can be extended to include non-functional requirements (e. g., privacy, clarity, and accuracy) from the goal model. These may be provided optionally by the process modeler for each process activity containing goal descriptions. *Backward reasoning* can be reduced to the problem of propositional satisfiability (SAT) [SGM04]. In [GMS05] the backward reasoning formula Φ is introduced:

$$\Phi := \Phi graph \wedge \Phi outval \wedge \Phi backward \, [\wedge \Phi optional]$$

$\Phi graph$ encodes a goal graph, the ground axioms for the invariants and the propagation rules (i. e., goal relation axioms) defined in [GMS05]. $\Phi outval$ defines the desired final output values and $\Phi backward$ encodes the backward reasoning as introduced in [SGM04]. $\Phi optional$ defines an optional formula for constraints on possible values of the goals. In our work, we apply the presented *backward reasoning* formula to find the best fitting leaf goals (i. e., IoT device capabilities) based on activity goals.

2.3 Workflow Perspective

The workflow perspective introduced the relevant workflow concepts including modeling languages and the concept of IoT-aware workflow management.

2.3.1 Workflow Concepts

Workflows are a key technology for the automation of business processes. A business process is defined as “a kind of process in the domain of business organisational structure and policy for the purpose of achieving business objectives” [HH95] by the Workflow Management Coalition (WfMC). Business processes are composed of a set of activities, i. e., atomic process steps, which as a whole contribute to achieve the underlying objective of the process. In the context of this work, we consider user-oriented processes in the IoT instead of traditional business processes. Therefore, our focused application domain is a smart environment with ubiquitous access to IoT enabled devices via IoT services. We utilize and build upon workflow related technologies from the domain of business processes.

According to the WfMC, a “workflow is concerned with the automation of procedures where documents, information or tasks are passed between participants according to a defined set of rules to achieve, or contribute to, an overall business goal” [HH95]. In a shorter definition, a workflow is “the computerised facilitation or automation of a business process, in whole or part.” [HH95]. Workflow Management Systems (WfMS) enable this automation by *enacting*, i. e., instantiating, and executing workflows and managing the required and produced process resources.

Figure 2.8 shows the basic characteristics of a WfMS as specified by the Workflow Management Coalition (WfMC) [HH95]. A WfMS may support several phases from process design until enactment and runtime resource allocation as well as user interaction.

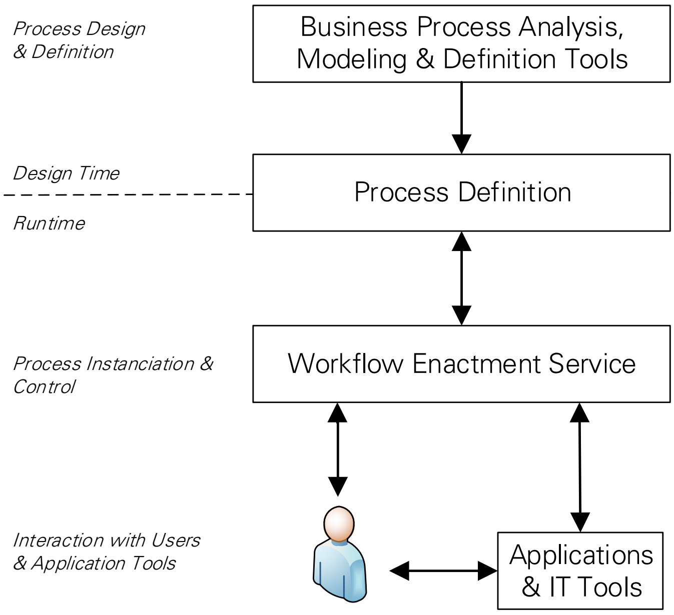


Figure 2.8: Basic characteristics of a WfMS [HH95].

2.3.2 Workflow Modeling

Before a workflow can be enacted, it has to be specified in form of a workflow model. This can be achieved with the help of workflow modeling tools. The effort of specifiying a workflow largely depends on the desired level of automation. A workflow model is build from atomic modeling elements at design time and represents an executable implementation of a process. In general, workflows can be modeled either in an imperative or declarative fashion. The declarative workflow model is a constraint-based model. Therefore, it restricts the set of possible control flows and data flows between activities. Just like in declarative programming, the declarative workflow model defines *what* is to be computed, rather than *how* it is to be computed [FLM$^+$09]. In contrast, an imperative workflow model describes the allowed or wanted behaviour by defining an explicit control flow and data flow between workflow activites. Therefore, it is concerned with the logic of interacting process resources. Even though, the declarative modeling approach meets the requirements of smart environments by allowing ad-hoc adaptations of control flow, it

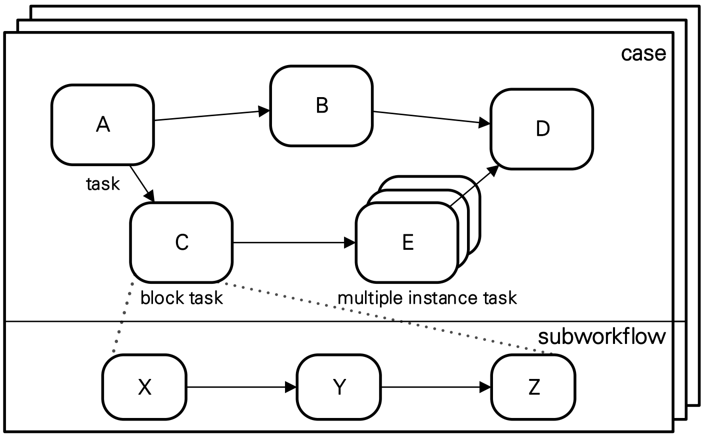


Figure 2.9: General components of a workflow model [RTHEvdA04b].

is not widely adopted. Fahland et al. propose, that of “two semantically equivalent process models, establishing sequential information will be easier on the basis of the model that is created with the process modeling language that is relatively more imperative in nature.” [FLM+09]. On the other side, “establishing circumstantial information will be easier on the basis of a declarative process model than with an imperative process model.” [FLM+09].

Figure 2.9 illustrates the typical components of an imperative workflow model. In general, a workflow model is a directed graph, where the nodes are tasks, also called *activities*, and the vertices define the control-flow. An instance of a workflow model is called *case*. Tasks can be atomic, block or multiple instance tasks. Block tasks incorporate a subworkflow, enabling hierarchical workflow modeling. A multiple instance task may generate arbitrary equal task instances at runtime. A generic workflow metamodel includes logical and control-flow structures, e. g., *IF*, *AND*, *OR* and WHILE. In combination with the explicit modeling of data flow, a workflow model can be used as a programming language. In fact, when abstracting from capacity constraints, any workflow language is turing complete [vdAtH05].

Business Process Model and Notation

The Business Process Model and Notation (BPMN) and its extensions are widely used to model domain-specific business processes. One of the major benefits that the current industry standard BPMN 2.0 provides, is the generation of machine-readable process models from graphical notations. Nevertheless, BPMN lacks the ability to natively model types of process data and other semantic information, e. g., resource capabilities and context factors. Therefore, a dynamic mapping of process activities to available resources at runtime cannot be deduced directly from a BPMN process model. Wohed et al. analysed the support for workflow patterns in BPMN [WvdAD$^+$06]. The results indicate that the scope of BPMN does not include appropriate modeling notations for process resources in general. As stated above, knowledge about process resources, e. g., resource capabilities and constraints, is crucial for allowing the intelligent matching of activities to qualified resources. In addition, the modeling, enactment and adaptation of workflows in the IoT strongly depends on process resource semantics. While the data flow and orchestration can be modeled comprehensively through a BPMN process model, there still exists a gap between the modeling of capabilities and inherent runtime specification of process data types. However, type-safety of process data is an essential feature when supporting runtime adaptation and model evolution [Sch09].

Business Process Execution Language

The Web Services Business Process Execution Language (WS-BPEL) has been the de facto standard for implementing executable processes until it was superseded by BPMN 2.0. In contrast to the notations of BPMN and UML, BPEL contains strong and complete execution semantics [ACD$^+$03]. Although there exist approaches, which enable a mapping from BPEL to BPMN [VDAL08], these transformations inherently lose information. Furthermore, an ontological representation of BPEL processes has been developed, in order to reason about executable process models [NWVL07]. This approach does not only support the explicit knowledge representation, which usually gets lost during the transition from design time to runtime, it also enables the use of this knowledge, e. g., semantically annotated BPEL activities, for process adaptation. However, WS-BPEL is limited to strictly web service-based system architectures, which is not necessarily given in an ubiquitous environment or the IoT. Furthermore, process modeling with WS-BPEL is rather execution centric, e. g., activities are directly tied to resources at design time, and therefore lacks the desired abstraction level we want to support with respect to the dynamic adaptations of activities and the dynamic assignment of resources at runtime. Human resources can be integrated and interact with workflows via WS-HumanTask [AAD$^+$07] and BPEL4People [KKL$^+$05]. These approaches only support a limited representation of user capabilities and their interaction context.

Yet Another Workflow Language

Yet Another Workflow Language (YAWL) was introduced in 2005 as a new workflow modeling language based on high-level Petri nets [vdAtH05]. The motivation for creating a novel workflow language derived from an exhaustive feature comparison of existing workflow languages based on identified workflow patterns [vdAtH05, vDATHKB03]. The YAWL notation was developed to overcome the shortcomings of existing workflow languages [vdAtH05], e. g., WS-BPEL and BPMN. A first, the introduced workflow patterns only captured the control-flow perspective of workflow modeling. Then, they were extended to the data flow and resource perspective [RTHEvdA04a, RTHEvdA04b]. YAWL does not only integrate the support for those workflow patterns, but provides a strong formalisation through its origin based on high-level Petri nets.

2.3.3 Internet of Things-aware Workflow Management

The aim of IoT-aware workflow management is to allow the representation and useful integration of IoT components as workflow resources. Then, IoT components can become part of an integrated workflow model, allowing for intelligent and dynamic task-specific connections among them. One of the major challenges of IoT-aware workflow management is to support the heterogeneity of *Things* and to cope with the dynamic nature of ad-hoc ubiquitous systems [MRM13, HSS16]. In the following, we will shortly introduce the relevant concepts for workflow adaption in the IoT.

Workflow Adaptation: The IoT and ubiquitous systems introduce additional challenges for process management regarding the dynamic and heterogeneous nature of these systems. Usually, not all factors, properties and components of the system are known at design time. In order to customize and evolve the workflows at runtime, capabilities for process evolution, runtime modeling and end-user development have to be provided. The dynamic nature of the IoT and their components requires flexible processes capable of being adapted at runtime considering context factors and other requirements. Current formal process modeling methods, e. g., high-level Petri nets and semantic modeling approaches, allow static verification of and reasoning about process models at design time. However, the runtime adaptation of processes introduces new sources of uncertainty, which static verification methods cannot address. The integration of user interactions into processes at design time and runtime even increases the complexity of these issues. Several approaches and classifications of process model adaptations have been proposed and investigated in recent years. A comprehensive overview is provided in [SMR$^+$08]. The various concepts for process flexibility have been classified into four classes representing different problem areas. Those classes have been identified as *flexibility by design*, *deviation*, *underspecification* and *change*. The *flexibility by design* perspective comprises approaches for handling modifications at design-time and is inherently provided by process modeling environments. Process flexibility by *deviaton* includes short-term or *ad-hoc* control-flow adaptations at runtime, while long term process model evolution is covered by the flexibility class *change*. Furthermore, the *underspecification* of workflow models can enable the late binding of process resources and activities. Within this work, we limit the scope for workflow flexibility to the class of *underspecification*.

2.4 Summary

In this chapter, we introduced the relevant concepts and definitions on which the contributions of this work are built upon. We divided these concepts in three perspective covering the resource, user, and workflow perspective. From the resource perspective, we will use the IoT reference model [BBDL$^+$13] as a foundation for the creation of our ontology for role-based *Things*. In this, we also employ the introduced concepts of semantic and role-based context modeling. In the user perspective, we will employ the introduced Tropos goal modeling language to specfiy high-level user goals which in turn can be used for activities in a workflow model. The workflow perspective also introduced the notion of IoT-aware workflow management.

3 Requirements Analysis and Approach

In this chapter, we analyse the requirements for integrating *Things* as workflow resources. The goal of the analysis is to illustrate the differences between existing workflow management approaches and their application in the IoT. We focus our analysis on giving an overview of the challenges and requirements extracted from related work for the integration of *Things* as workflow resources. Also, we develop a quantified classification scheme for the fulfillment of these criteria and use it to compare existing approaches. Based on the results of the requirements analysis, we limit the scope of our contribution to identified gaps in research. Finally, we describe our approach for contributing towards IoT-aware workflow management and the relation of this approach to the identified gaps in research. Based on the application scenario presented in Section 1.2, the following basic requirements for integrating *Things* as workflow resources have been identified:

- Knowledge of **IoT device capabilities** and their **context** has to be modeled explicitly.
- Workflow resources have to be **allocated at runtime** using a late binding mechanism depending on the requirements of the activity and runtime context.
- The **requirements of activities** have to be modeled explicitly.

The explicit modeling of activity requirements and IoT device capabilities as well as their context build the foundation for the late binding of IoT workflow resources at runtime. The late binding mechanism can take advantage of the modeled context and activity constraints to find a fitting workflow resource. To provide a more comprehensive list of requirements, we performed the following requirements analysis.

3.1 Requirements

In this section, we introduce requirements for integrating *Things* in the IoT as workflow resources. Therefore, we review literature from both the perspective of resource management and the perspective of workflow management in the IoT.

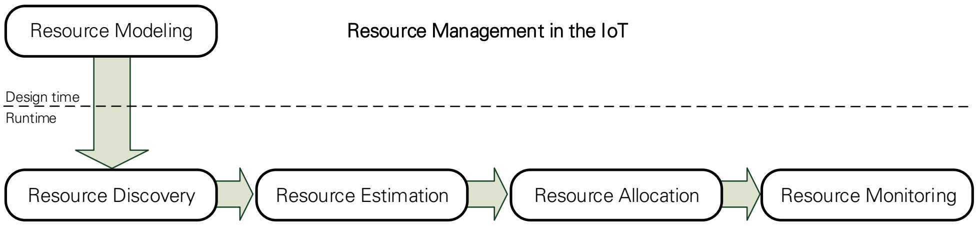


Figure 3.1: Workflow for resource management in the IoT [DPB17].

3.1.1 IoT Resource Perspective

Resource management in the IoT covers managing an ecosystem of "heterogeneous interconnected devices, whose data and services (virtual and physical resources) are used by several different applications that access such pool of resources via network" [DPB17]. In the context of resource management, a resource is any object which can be allocated within a system [TW87].

Figure 3.1 shows the typical workflow for resource management in the IoT and involves the following phases: resource modeling, resource discovery, resource estimation, resource allocation and resource monitoring [DPB17]. Resource modeling implies the creation of a domain specific model of IoT resources. This can be based on any modeling language, e. g., object-oriented or semantic, which provides the necessary degree of abstraction, granularity and formalism [DPB17]. While resource modeling is completed at design time, the subsequent resource discovery is executed at runtime. Resource discovery is the process of searching and discovering available resources based on some resource requirements or restrictions. Resource estimation is a way of assuring the QoS for an application by estimating the required resources based on some calculation [DPB17, KRG$^+$15, MS14a]. Resource allocation is the activity of satisfying application requirements by binding resources as data source or data processing system component for a specific task. Resource monitoring includes the environmental variations and context changes within allocated resources. It is necessary to adapt and re-allocate resources which have become unavailable or obsolete with respect to the requirements for allocation. Resource monitoring also uses Quality of Context (QoC) parameters for detecting the need for resource adaptation.

One of the biggest challenges for resource management is the possible **large scale** of the IoT [IBGB16]. Applications in the IoT need to scale the number of resources involved to avoid the explosion of resources, exchanged data and operations [GBMP13]. This impacts both the modeling and the discovery phase of resource management.

The modeling effort can quickly rise to unfeasible levels when the **heterogeneity** of devices is considered. The challenge of heterogeneity also encompasses technologies, services and environments [Bor14]. Service discovery is also affected by a large-scale IoT scenario both regarding performance and architecture.

Existing service discovery techniques from the distributed systems domain are not directly applicable, as a service abstraction does not cover the **context-aware** nature of *Things*. Besides, the **dynamics** in the availability of *Things* and their interconnections have to be considered within the discovering and monitoring phases. While discovering resources, it has to be ensured that the states of the physical world and the virtual world are synchronized. Otherwise, a discovered resource may not be allocated successfully. This ongoing synchronization process is part of the monitoring phase.

Furthermore, resources need to be uniquely identifiable within a **flexible identification** scheme [PP12]. Flexible identification is concerned with identifying, naming and addressing IoT devices and their provided resources. Even though the IPv6 supports a very large address space, it is designed to identify devices, not their encapsulated resources [PP12]. Therefore, a resource management system needs to provide an extended identification scheme. Furthermore, applications and users which employ a resource management system need means to specify quantifying and qualifying queries for resource discovery.

The **query support** should allow for exact matches and range queries [PP12]. While traditional Web services are virtual entities providing computational resources, an IoT resource is **context-aware** as it exists in the real world [WJ12]. An IoT service extends the physical part (hardware) of a *Thing* with a virtual part (software) to provide connectivity capabilities. An IoT service can therefore be categorized as a Cyber-physical system.

In addition, **Quality of Context** has to be considered, as inconsistent and outdated context information is likely to occur inside context models for real time [BG14]. Like the context-awareness requirement, QoC has an impact on the modeling, discovery and monitoring phase of resource management. The modeling and discovery of context-aware resources are enabled by including context parameters in the resource model and using these parameters for discovery. The monitoring phase continuously checks these context parameters for state changes.

Closely related to the support for context-awareness of *Things* is the **mobility of devices**. As the location of a device can be subject to constant change, the provision, quality and availability of resources may vary significantly [HSS16]. The support for **Quality of Service** parameters related to the estimation and allocation phases of resource management [KRG$^+$15]. These QoS parameters are used to find the best fitting resource within all discovered resources. Furthermore, the service level agreements have to be continuously monitored for allocated resources.

The requirement to process requests and analytics in **real-time** is mandatory for several application domains in healthcare and factory automation. Often, not only single data events but **data stream processing** is required to support online analytics tasks in the IoT. The support for real-time processing has an impact on all runtime phases of resource management. Closely related to the real-time and QoS requirement is the support for **application priorities**. In a highly distributed system, a failure of individual system components is likely and should not compromise the overall system function and performance [MS14a]. Therefore, the resource allocation and monitoring phases need to support some kind of **fault tolerance** [DPB17].

Load balancing is another requirement within the estimation phase of resource management and rooted in the distributed nature of the IoT. Additionally, resource management requires **cost minimization** strategies, e. g., regarding the energy consumption, as well as a **secure environment**, including robustness to communication attacks, privacy of data sources and sinks as well as data and device integrity.

Table 3.1 gives an overview of the introduced requirements for resource management in the IoT and highlights the relation of the requirements to the phases of resource management.

Table 3.1: Requirements for resource management in the IoT.

Requirement	Relation to Resource Management					Sources
	Modeling	Discovery	Estimation	Allocation	Monitoring	
Large scale	✓	✓				[IBGB16] [Bor14] [GBMP13] [MSDPC12]
Dynamics		✓			✓	[IBGB16] [Bor14]
Flexible ID	✓	✓				[PP12]
Query support		✓				[PP12]
Context-aware	✓	✓			✓	[PP12] [WJ12] [PZCG14] [Bor14] [MSDPC12]
Quality of Context	✓	✓			✓	[BG14]
Quality of Service			✓		✓	[KRG$^+$15] [MS14a] [Bor14] [GBMP13]
Heterogeneity	✓					[DPB17] [BS11] [Bor14] [GBMP13] [MSDPC12]
Real-time		✓	✓	✓	✓	[DPB17]
Data streams				✓	✓	[DPB17] [AKF$^+$14]
Application priority	✓		✓	✓		[DPB17]
Mobility of devices	✓	✓			✓	[DPB17] [BS11] [Bor14] [MSDPC12]
Fault tolerance				✓	✓	[DPB17] [MS14a]
Load balance			✓			[DPB17]
Cost minimization	✓	✓	✓	✓	✓	[DPB17] [Bor14] [GBMP13] [MSDPC12]
Secure environment		✓		✓	✓	[BS11] [GBMP13] [MSDPC12]

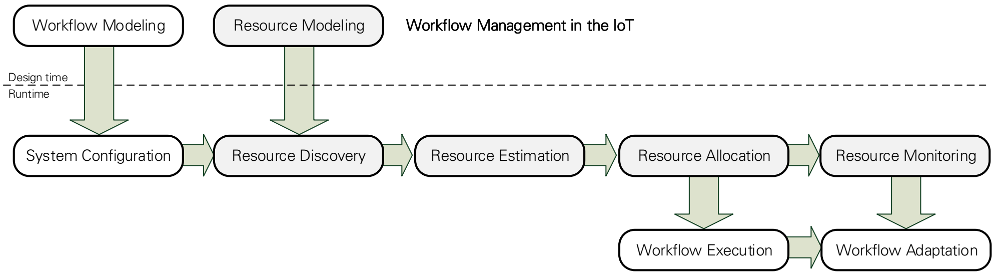


Figure 3.2: Workflow and resource management in the IoT.

3.1.2 Workflow Resource Perspective

From a workflow management perspective, IoT resources can be used as workflow resources to gather data about the physical context or to trigger actions in the real world [HSS16]. In this case, all the requirements defined in Table 3.1 also have to be met by a WfMS or a middleware layer between the WfMS and the actual IoT resources. However, there exist some additional requirements for the workflow perspective. In the following, we will introduce these requirements to complement the resource management perspective. Figure 3.2 shows the relation between workflow management (white activities) and resource management (grey activities) in the IoT [DPB17, VDATHW03]. In fact, all activities besides the modeling of workflows and resources are considered as responsibilities of a WfMS [VDAVH04]. In the following, we present the requirements for integrating *Things* as workflow resources. From a workflow modeling perspective, it is necessary to provide native **modeling support** for integrating *Things* as workflow resources [MRH15]. In many scenarios, encapsulating IoT devices with services for their subsequent integration as workflow resources is infeasible due to the missing consideration of the real world context. This missing context-information about the IoT device at the WfMS level can lead to unwanted system behaviour. The modeling perspective therefore needs to support a fitting level of *abstracting* the capabilities and service interfaces of *Things*. In addition, **flexibility support** for resource allocation is required, e. g., through late binding mechanisms for specifying context and resource-related constraints [BS11]. Workflows in ubiquitous systems will require the operation of highly dynamic and ad-hoc relationships. If resources are found at runtime, there has to exist a **resolution strategy** for multiple fitting resources [ST05]. When IoT resources are no longer responsive or fitting for executing a workflow activity, a **fault tolerance** mechanism is required, e. g., by re-discovering or re-allocation of resources [SHA17]. Furthermore, if unwanted actions have been triggered in the real world, **Cyber-physical Consistency** checks and according rollback strategies need to be considered [SHHA16]. Cyber-physical Consistency (CpC) is defined as a synchronous state between the real world context and the virtual representation of it. With respect to the process modelers, **transparency** of IoT resource and workflow interaction is required [CSB16]. Meyer et al. also see the **distributed execution** and the entailed distributed data management as a requirement for executing workflows in the IoT [MSMP11].

Table 3.2: Requirements for workflow management in the IoT.

Requirement	Relation to Workflow Management				Sources
	Modeling	Configuration	Execution	Adaptation	
Modeling support	✓				[MRH15] [CSB16] [MSMP11]
Flexibility support	✓		✓	✓	[BS11] [CSB16] [MSMP11]
Resolution strategy			✓		[ST05]
Fault tolerance				✓	[SHA17] [MSMP11]
Cyber-phys. Consistency	✓			✓	[SHHA16]
Transparency	✓				[CSB16]
Distributed execution		✓	✓		[MSMP11]

Table 3.2 gives an overview of requirements from the literature on workflow management in the IoT and their relation to the phases of workflow management.

3.1.3 Relation to Research Questions

To guide our research and to limit the scope of this work, we will map the aforementioned requirements to the following research questions, extracted from Section 1.3.1:

- **RQ1** What are the differences between workflow resources in the cyber and the physical world from the perspective of a WfMS?

- **RQ2** How to model context-sensitive capabilities of IoT workflow resources?
- **RQ3** How can a WfMS cope with the uncertainty of IoT resource availability during workflow execution?
- **RQ4** How to minimize failures during workflow execution caused by unavailable resources?
- **RQ5** How to allow for flexible as well as correct workflow execution in the IoT and what are the tradeoffs?

Table 3.3: Requirements and their relation to the research questions.

	Requirement	Relation to Research Questions				
		RQ1	RQ2	RQ3	RQ4	RQ5
Resource Management	Large scale					
	Dynamics	✓		✓	✓	✓
	Flexible ID	✓				
	Query support		✓	✓		✓
	Context-aware	✓	✓			
	Quality of Context	✓	✓			✓
	Quality of Service	✓				✓
	Heterogeneity	✓	✓			
	Real-time	✓				
	Data streams	✓				
	Application priority					
	Mobility of devices	✓	✓	✓	✓	✓
	Fault tolerance			✓	✓	✓
	Load balance					
	Cost minimization					
	Secure environment					
Workflow Management	Modeling support	✓	✓			
	Flexibility support			✓		✓
	Resolution strategy		✓			✓
	Fault tolerance			✓	✓	✓
	Cyber-phys. Consistency	✓		✓	✓	✓
	Transparency	✓				✓
	Distributed execution					

Table 3.3 shows the existing relations between the requirements for workflow and resource management in the IoT and the research questions in the context of this work. The **large scale** of the IoT in itself has no direct relation to any of the research questions. In addition, the support for **application priorities**, **load balancing**, **cost minimization**, **secure environment** and the **distributed execution** have no direct impact on this work. However, we note that especially the cost minimization, e. g., in terms of energy efficiency, and the security concerns for highly distributed, interconnected and potentially privacy invading IoT technology are key challenges for enabling a useful employment of *Things* in many application scenarios. As these topics are subject to research in their own fields, they are not considered in the following. The **mobility of devices** is highly relevant, but also covered by the requirements **context-aware** and **fault tolerance**. Therefore, we will not investigate the **mobility of devices** as a separate requirement. Furthermore, we shift the responsibility of **fault tolerance** in resource management to the workflow management. In the context of this work, a WfMS has to deal with runtime defects and connection losses of *Things*. The need for a **flexible identification** scheme has to be provided by the network layer of the OSI model. In this work, we presume that *Things* are uniquely addressable within the network layer and preserve their unique ID after connection losses. Therefore, we do not investigate the requirement of a flexible identification scheme any further. Even though the topic of **CpC** is of high relevance to this work, it is subject to related research by Seiger et al. [SHA17, SHHA16, SHS16, SNS14]. The remaining requirements are used as a quantifiable classification scheme for evaluating related work both from the field of workflow and resource management towards their support of IoT-related requirements. Furthermore, the classification scheme is used in the evaluation chapter to compare the results of this work to related research.

3.2 State of the Art Analysis

The main benefit of using Business Process Management (BPM) technology for modeling IoT applications is the seamless integration of IoT ecosystems with traditional information systems [DPB17]. Even though traditional workflow modeling languages, e. g., BPMN, BPEL and YAWL, and management systems are not tailored to IoT resources, there are approaches that extend these technologies towards the support of IoT ecosystems. In the following, we will introduce fulfillment criteria for the subsequent comparison of related work.

3.2.1 Fulfillment Criteria

To provide a comparable and quantified scheme for classifying existing approaches regarding their level of fulfillment of the requirements, we introduce the following criteria:

Table 3.4: Fulfillment criteria for the requirements.

	Requirement	Criteria for Level of Fulfillment		
		Low (*)	Medium (**)	High (***)
Resource Management	Dynamics	Syntactic resource discovery	Model-based resource discovery	Semantic resource discovery
	Query support	Exact match queries	Range queries	Semantic queries
	Context-aware	Location-based context model	Context-sensitive capabilities	Context-sensitive relations between Things
	Quality of Context	Context parameters	Update mechanism	Real-time consistency
	Quality of Service	Model support	QoS-aware resource discovery	Continuous QoS monitoring
	Heterogeneity	Model-based input and output data	Model-based *Things*	Semantic *Things*
	Real-time	Possibility for real-time processing	Real-time processing (use case specific)	Guaranteed real-time processing (contracts)
	Data streams	Possibility for data stream processing	Event stream processing	Real-time data stream processing
Workflow Management	Modeling support	IoT-sensors	IoT-actuators	Context-aware *Things* with complex capabilities
	Flexibility support	Late binding	Context and resource constraints	Continuous monitoring and re-allocation
	Resolution strategy	Includes context parameters	Includes QoC criteria	Includes QoS criteria
	Fault tolerance	Re-allocation	Virtual consistency	Cyber-physical consistency
	Transparency	Modeling resource requirements for activity	Modeling resource constraints for activity	Modeling intention of activity

Table 3.4 gives an overview of the fulfillment criteria. These have been derived from clustering existing approaches. The criteria levels are additive, such that the fulfillment level *high* entails the fulfillment of level *middle* and *low*.

To support the **dynamics** of the IoT, a WfMS can use resource discovery techniques. These are categorized as syntactic, model-based, or semantic. The syntactic resource discovery includes names and tags as query parameter. In addition, model-based resource discovery can use typed resources, e. g., with inheritance, to cover more expressive query parameters. Finally, semantic resource discovery employs semantic web technologies to define more expressive queries and find resources based on a semantic model. The employment of semantic web technologies is an essential part of the IoT vision [AIM10].

The **query support** for resource discovery can either facilitate exact match, range or semantic queries. Exact match queries can include parameters from a resource model, while range queries extend these with cardinality. Semantic queries support a vast range of parameters and may also use indirect or deducted parameters.

The **context-awareness** of *Things* can be supported by using location as context parameter. In addition, a context model can express context-sensitive capabilities of *Things*, e. g., availability of a capability is restricted to a certain location. Finally, a context model can define the context-sensitive relations between *Things*.

The **Quality of Context** can be supported by specifying context parameters in relation to *Things*. If these relations are continuously monitored, the quality of context increases as the parameters can be used as runtime information. When this update mechanism is operating at real-time, the context parameters are consistent with the real world.

The **Quality of Service** parameters need modeling support to be used for resource discovery. If QoS parameters are continuously monitored, they can be used for resource discovery and re-allocation.

The **Heterogeneity** of *Things* can be overcome by using a black-box approach, i. e., model input and output data of devices. In addition, *Things* can be specified with a model-based approach to define their characteristics and capabilities. Finally, a semantic model for *Things* can include the meaning and relation between these capabilities and context factors.

The potential for **real-time** processing of sensor data or actuator commands is given, if all system components and network protocols define finite time bounds. If these time bounds can be specified in relation to the workload and fall below a given definition of *real-time*, a use-case specific real-time processing is possible. If all possible processing operations consume less time than the given time boundary, real-time processing is guaranteed. However, there exists no general definition of a real-time processing latency. It is highly use-case specific and can range from milliseconds up to one second [OMSJ06]. In this classification scheme, an approach will be classified as being able to provide use-case specific real-time processing if such measurements and definitions are provided.

Data stream processing is necessary for some sensor-centric IoT applications. In addition, the data streams can be analyzed and transformed into high-level event streams. Finally, their generation and analyzation can be provided in real-time.

From a workflow management perspective, the **modeling support** for IoT has several stages. At a minimum, IoT-sensors have to be included as data sources. In addition, IoT-actuators can be triggered by a workflow activity enabling the manipulation of the real-world by the WfMS. At the highest level, *Things* and their capabilities for sensing and acting are modeled in relation to their context.

A basic **flexibility support** is given by employing late binding for workflow resources. In addition, context and resource constraints can be used for resource discovery. In combination with their continuous monitoring, a re-discovery and re-allocation of resources is feasible.

During the discovery phase, a **resolution strategy** may include context parameters to select a single fitting resource. In addition, QoC and QoS criteria may be included.

Fault tolerance can be ensured by re-allocating unavailable resources during runtime. Furthermore, the state of an ongoing activity can be preserved or rolled back to maintain virtual consistency between a stateful re-discovery and re-allocation of workflow resources. Finally, CpC represents the highest level of fault tolerance as the state of the physical world is considered during fault compensation.

The **transparency** for workflow modelers requires means for modeling IoT resource requirements per activity. Then, modelers can specify which resources are eligible to execute a specific activity. In addition, the modelers can use constraints for resource parameters. In the highest level of abstraction, workflow modelers specify the intent of an activity.

3.2.2 IoT-aware workflow management

The following approaches regard *Things* and IoT services either from a data-oriented perspective or as workflow resources. We will now review and classify the state-of-the-art in IoT-aware workflow management according to their level of support for the aforementioned requirements. In this context, we do not include approaches which are strictly limited to resource management.

Modeling

In the following, we review related research which is mainly rooted in the modeling phase of IoT-aware workflows.

IoT-aware BPMN: Meyer et al. present an approach for integrating IoT devices as business process resources [MSMP11, MRH15, MRM13]. They extend the BPMN 2.0 specification to include IoT devices as a *Lane* subclass and native services as subtype of the *Performer* class. While this approach enables the modeling and invocation of IoT services, resources have to be addressed explicitly or by a syntactic expression. Meyer et al. provide a model-based approach for specifying IoT devices, their parameters and service encapsulation. However, they do not address the context-sensitive nature of IoT devices and provide no context model, nor means to specify QoC or QoS parameters. Even though they provide no measurements for round-trip service invocation, a WfMS which executes such an extended BPMN model might be able to operate near real-time. As the IoT services employ an event based communication, there is no native support for data streams. Their main contribution is the modeling support for sensors and actuators from a workflow management perspective. In terms of flexibility support, they propose a resource discovery and allocation mechanism using late binding. As the approach is centered on modeling, they do not discuss the runtime oriented topics of resolution strategy or fault tolerance. Finally, workflow modelers are enabled to specify resource requirements for each activity.

WSN4BPMN: Sungur et al. extend BPMN for modeling WSN (Wireless sensor network) processes to facilitate WSN programming [SSOK13]. Their goal is to bridge the gap between the technical expertise needed to program sensor networks and the domain expertise required to design useful processes [SSOK13]. Sensor and Actuator functions (capabilities) are exposed such that a discovery mechanism can be employed. However, the discovery is done at design time, such that a process model can be transformed into executable code. This is then deployed onto the sensor and actuator nodes. In addition, QoS parameters can be specified in the form of performance annotations. These define a performance goal, which adapts the actual implementation of sensor and actuator tasks. The heterogeneity of sensors and actuators is considered by specifying an output target and a return operation inside a 6-tuple for each activity. The system performance has the ability to operate in real time, as the code can be optimized for performance and is distributed among the WSN nodes. Sungur et al. claim to support several resolution schemes for finding a best fit WSN node for a specific task. Furthermore, the workflow modeler can specify resource requirements and constraints for each task.

uBPMN: Yousfi et al. propose a BPMN extension for modeling ubiquitous business processes [YBSD16, YdFDS16]. This includes an extension to the MOF meta-model and the XML schema definition of BPMN. The approach is limited to sensor-based devices, such as general sensors, smart readers, cameras, microphones and virtual sensors (collectors). As their main focus is the modeling support, many of the requirements for IoT-aware workflow management are not considered. However, they provide solutions for covering the heterogeneity, modeling support and transparency. The heterogeneity is supported by providing a meta-model for activity input and output data. The modeling support is limited to sensor types and the transparency for the workflow modeler is restricted to use specific activity types for each supported sensor type.

BPMN4CPS: Graja et al. propose an approach for modeling Cyber-physical Systems (CPS)-aware processes as a BPMN extension [GKGK16]. They focus on modeling the physical effects and context of CPS-aware processes. The BPMN extension supports special task types for the modeling of sensor and actuator tasks. Furthermore, device capabilities can be grouped into roles as role parameters, and their availability or activation depends on real-world environment parameters. Therefore, the extension allows for the modeling of context-sensitive device capabilities. Although the runtime perspective is not part of their contribution, the approach enables a model-based resource discovery using context and resource constraints at runtime. The resolution of resource discovery is not addressed, but at least the context-sensitive capabilities can be used to minimize device ambiguity. The process modeler can use resource constraints as well as requirements to model a workflow activity.

CM4BPM: Saidini et al. introduce a modeling approach for context-sensitive BPM [SRN15]. Their aim is to formalize contextual knowledge relevant to the business process perspective and to allow for the integration into process modeling. The motivation for integrating a generic context model into process models is to use the additional contextual information for runtime process adaptation. As the modeling approach is centered on the context and not the actual devices or *Things*, the resource discovery is not natively supported. The context model is based on OWL and therefore supports querying with the SPARQL. However, as modeling of process resources is not provided, there is no possibility for query-based resource discovery. On the resource management perspective, contextual knowledge is only supported by location. Due to the employment of SWRL rules, the context model can be updated based on predefined rules. The availability of resources can be directly integrated in the context model and used in conjunction with the location to support a flexible late binding mechanism.

SPU: Appel et al. propose an approach for modeling and execution of event stream processing in business processes [AKF^{+}14, AFFB13]. They aim at integrating IoT-sensor data as event streams into business processes with the help of Event Stream Processing Units (SPUs). These are realized by extending both BPMN and Event-driven Process Chain (EPC). The SPUs are not designed to provide modeling support for *Things* in the IoT but for providing a data-centric integration of sensor data streams. Therefore, the heterogeneity is addressed by providing modeling support for input and output data of sensors tasks. The main contribution is the integration of real-time sensor data streams into standard business processes.

Table 3.5: Analysis of related research in the modeling phase.

	Requirement	Approach					
		IoT-aware BPMN	WSN4BPMN	uBPMN	BPMN4CPS	CM4BPM	SPU
Resource Management	Dynamics	*	*	-	**	n.a.	n.a.
	Query support	-	-	-	-	n.a.	-
	Context-aware	-	-	-	**	*	-
	Quality of Context	-	-	-	*	**	-
	Quality of Service	-	**	-	-	-	-
	Heterogeneity	**	*	*	-	-	*
	Real-time	n.a.	*	n.a.	n.a.	n.a.	**
	Data streams	-	-	-	-	-	***
Workflow Management	Modeling support	**	**	*	**	-	*
	Flexibility support	*	-	-	**	**	-
	Resolution strategy	n.a.	*	-	*	-	-
	Fault tolerance	n.a.	n.a.	-	n.a.	-	-
	Transparency	*	**	*	**	n.a.	-

Table 3.5 gives an overview of related research focused on the modeling phase of IoT-aware workflows and shows their level of fulfillment regarding the introduced requirements.

Workflow Execution

In the following, we review related research which is mainly rooted in the execution phase of IoT-aware workflows. This includes the discovery and allocation of workflow resources and the actual data retrieval or invocation of services.

Decoflow: Loke et al. present a concept for service-oriented device ecology workflows within a home environment [Lok03]. Their approach uses WS-BPEL processes to model and manage smart devices and their interactions. However, devices and their capabilities are not represented in a separate model but direct service invocations. Therefore, the process modeler needs to know these services and their functionalities at design time. The hetereogeneity of devices is bridged by the service abstraction and their input and output modeling inside a decoflow process model. Within services, sensors as well as actuators are supported. One of the major benefits of the decoflow approach is the modeling of changes for device states. Therefore, fault tolerance can be supported by fault handlers, referring to the device states and their transition. If software agents observe the real world state of devices and perform these state transitions, a cyber-physical consistency between the real world and the workflow runtime perspective is achieved.

Presto: Giner et al. propose a software architecture for developing mobile workflow support in the IoT [GCFP10]. They aim at providing developer support for workflow development in the IoT. Their approach is centered on the software architecture of Presto and does not provide modeling support for *Things* or service capabilities. The architecture itself does support context-aware physical service selection within a certain location-based and task related context. These services are discovered and allocated at runtime by a late binding mechanism.

BPEL4IoT: Glombitza et al. present a programming-in-the-large approach using BPEL to realize business processes for an IoT [GEPF11]. The BPEL processes are used to generate custom tailored IoT applications for different target platforms. The approach is focused on fast development and providing real-time performance through the application of a lightweight Web Service transport protocol (LTP) for messages between WSN and servers. As this approach is focused on code generation, most of the resource management requirements are not considered. The heterogeneity of devices can be bridged by using input and output data modeling for activities of the BPEL processes. Furthermore, due to the reduced overhead of the generated IoT application code, it can operate and react in real time boundaries. However, without using QoS contracts on the allocated services, the real time processing of requests can not be guaranteed.

MOPAL: Peng et al. provide an approach for assignment and execution of business processes from cloud to mobile [PRS+13]. They propose a BPMN extension for the support of context constraint, e. g., location and hardware resources, to allow process assignment and execution on mobile devices. Although, their approach is not targeted at integrating *Things* as workflow resources, the integration of mobile devices as context-sensitive workflow resources has similar requirements. Mobile devices also suffer from unrealiable connection. They exist and move in the real world, restricting their provided capabilities to situational contexts. Moreover, the person operating the mobile device defines the available capabilities for executing a specific task. MOPAL supports constraint based dynamic resource discovery. A workflow activity includes a service definition and a context-constraint definition for dynamic runtime activity assignment. Moreover, a fault tolerance mechanism is supported for re-assigning resources to activities upon execution failure. The failure state is monitored such that virtual consistency is supported. The resolution strategy for ambiguous service invocations uses context constraints, which are part of the workflow model and increase the transparency of adaptive behaviour for the process modeler.

ROA: Dar et al. propose a resource oriented integration architecture for the IoT [DTB+15]. Their aim is to provide a business process-based development of IoT applications by defining the overall control-flow of the application and generating service desciption files for the late binding of IoT services. The architecture provides service registry and service discovery based on standard languages like Web Services Description Language (WSDL) and Web Application Description Language (WADL). Therefore a unified integration of *Things* can be provided through describing service methods and their input as well as output data. The volatility of IoT services is adresses by a service replacement facility. The service discovery itself is not model-based, but a list of predefined services for each service type. As the approach relies on WSDL, the definition of QoS characteristics and their usage for QoS-sensitive service discovery is enabled. The ability to use Constrained Application Protocol (CoAP) as internet message protocol allows for real-time performance of ROA-based IoT applications. The business process modeling perspective support sensors and actuators with service encapsulations. However, the context-sensitive nature of provided capabilities is not regarded. The service replacement supports a virtually consistent fault tolerance by transferring stateful service invocations from a failing to a newly allocated service. Process modelers can model requirements for IoT services as part of an activity.

makeSense: Casati et al. present an approach for orchestrating the physical enterprise with wireless sensor networks using business processes [CDD+12]. Their aim is to provide a unified programming framwork and code generator for deploying applications on WSN nodes. MakeSense supports a model-based discovery of WSN using capability and context requirements. The context is limited to the location of WSN. QoS parameters of WSN services are considered in terms of performance requirements. The heterogeneity of sensors and actuators is bridged by using an application capability model. This specifies coarse-grained descriptions of the WSN including available sensors actuators and their operations. Real time operation of a WSN-based application is possible due to the generation and deployment of low level code. In the process model, sensors and actuators as well as their context constraints can be modeled in a new activity type.

Table 3.6: Analysis of related research in the execution phase.

	Requirement	Approach					
		Decoflow	Presto	BPEL4IoT	MOPAL	ROA	makeSense
Resource Management	Dynamics	-	-	-	**	*	**
	Query support	-	-	-	-	-	-
	Context-aware	-	*	-	**	-	*
	Quality of Context	-	-	-	-	-	-
	Quality of Service	-	-	-	-	**	*
	Heterogeneity	*	-	*	-	*	**
	Real-time	n.a.	-	**	-	**	*
	Data streams	-	-	-	-	-	-
Workflow Management	Modeling support	**	-	*	-	**	**
	Flexibility support	-	*	-	**	*	**
	Resolution strategy	-	n.a.	-	*	-	-
	Fault tolerance	***	-	-	*	**	-
	Transparency	-	-	-	**	*	**

Table 3.6 gives an overview of related research focused on the execution phase of IoT-aware workflows and shows their level of fulfillment regarding the introduced requirements.

Workflow Adaptation

In the following, we review related research which is mainly rooted in the adaptation phase of IoT-aware workflows. This includes the context-sensitive discovery and re-allocation of workflow resources.

SAMProc: Schmidt et al. introduce a middleware for self-adaptive mobile processes in heterogeneous ubiquitous environment [SH07]. Developers can create self-adaptive mobile process descriptions which is then mapped by the middleware to Web services. The approach is focused on providing a Model-driven Architecture (MDA) development approach for self-adaptive mobile applications. The code generation takes into account the runtime context and state of the mobile devices. Once the application logic is modeled as a BPEL process, the code generation integrates migratable Web services. These can adapt themselves by changing their facet according to the runtime target location. The resource discovery for services is model-based and uses the location context of mobile devices. Due to the service oriented approach, input and output data can be modeled to bridge device heterogeneity. Re-allocation of resource upon a failure is based on context information and resource constraints. The state of a failing service is conserved and transferred to provide vitual consistency. Process modelers need to specify resource requirements for activities.

CANthings: In [DSR15] Davoudpour et al. argue that services of IoT systems must perceive the environmental context. The authors propose an IoT framework based on Timed Colored Petri Nets (TCPN), which allows for modeling service compositions of IoT systems by transforming Event-Condition-Action (ECA) rules into TCPN's [DSR15]. This enables time-sensitive context modeling that explicitly represents context changes and service compositions. CANthings includes an ontology for specifying *Things* and their context to enable both reusability and interoperability. The ontology-based definition of *Things* enables a semantic resource discovery and potential query support using SPARQL. However, using semantic queries for resource discovery is out of the scope of their approach. Also, context-aware capabilities of *Things* can be modeled within the ontology. Therefore, the heterogeneity is bridged by providing semantic interoperability. Furthermore, real-time processing is possible due to the employment of TCPN for modeling the context changes. Even though CANthings is not focused on workflow integration of IoT resources, the adaptation of context information relates to the workflow adaptation phase as a technical requirement.

Semantic BPEL4WS: Lee et al. propose an approach for supporting dynamic workflows in a ubiquitous environment based on semantic modeling [LYS07]. The semantic model is constructed into DAML-S ontologies and used by the BPEL4WS engine to support the dynamic discovery and invocation of semantic Web services. DAML-S descriptions include a service profil defining semantic service capabilities, an input, output, precondition and effect specification and a service model. Due to the semantic modeling of service capabilities, the possibility for a SPARQL query-based resource discovery is given at runtime. However, semantic queries are not part of the concept. The WSDL extension also allows for the specification of QoS parameters for services. The device heterogeneity inside ubiquitous systems is bridged by defining the input and output data of service methods. From the workflow management perspective, modeling support and late binding of service encapsulated IoT sensors and actuators is provided. In case of a service failure, a re-discovery and allocation is triggered. The process modeler can specify service requirements for activities.

SitOPT: Wieland et al. present a general purpose situation-aware workflow management system [WSBL15]. SitOPT aims at automatically adapting workflow behavior according to runtime situations. These are derived from low-level sensor data. Workflow models can include situational control-flow to design possible runtime variations. Situation templates are used to support the situation-recognition. The focus of SitOPT is the runtime control-flow adaptation of workflows. Therefore, the modeling of context-aware *Things*, as well as their dynamic integration as workflow resources is not considered. Nevertheless, the continuous monitoring of situational context is relevant in the workflow management perspective. Furthermore, fault tolerance is provided by adapting the control-flow and re-allocation of resources.

SCORPII: Chang et al. propose a middleware for discovering proximity-based service-oriented industrial internet of things [CSM15]. Their work aims at providing a middleware for balancing the context-sensitive task allocation between mobile devices and cloud services. Therefore, a resource efficient mashup of mobile processes can be supported. Functionality of *Things*, here smart objects, is described in service metadata. As the main focus is the resource efficient task distribution, *Things* are not considered as workflow resources. The service metadata allows for a location-sensitive model-based resource discovery. Furthermore, QoS requirements can be specified. The heterogeneity of devices is bridged by modeling input and output data of service methods. Workflow tasks are distributed at runtime according to context and resource constraints. The resolution strategy uses the service metadata and location information to provide a proximity-based resource discovery. Also, service failures are compensated by re-allocation of workflow resources.

Table 3.7: Analysis of related research in the adaptation phase.

	Requirement	Approach				
		SAMProc	CANthings	Semantic BPEL4WS	SitOPT	SCORPII
Resource Management	Dynamics	**	***	***	-	**
	Query support	-	n.a.	n.a.	-	-
	Context-aware	*	**	-	-	*
	Quality of Context	-	**	-	-	-
	Quality of Service	-	-	*	-	**
	Heterogeneity	*	***	*	-	*
	Real-time	-	*	-	-	-
	Data streams	-	-	-	-	-
Workflow Management	Modeling support	-	-	**	-	-
	Flexibility support	**	-	*	***	**
	Resolution strategy	-	-	-	-	*
	Fault tolerance	**	-	*	*	*
	Transparency	*	-	*	-	-

Table 3.7 gives an overview of related research focused on the adaptation phase of IoT-aware workflows and shows their level of fulfillment regarding the introduced requirements.

3.3 Discussion

By reviewing and classifying related work we aim at identifying existing research gaps in the field of IoT-aware workflow management. It is important to note that the heterogeneity of related research regarding their overall aim introduces a certain level of ambiguity to the classification. Therefore, approaches which have been classified into the same level of fulfillment for a certain criteria still may differ in several ways. First, the overall aim of the approach may limit its reusability in other IoT application scenarios, e. g., approaches using code generation can not take advantage of a WfMS at runtime. Second, some approaches provide the technical possibility to fulfill a certain requirement, but do not describe or integrate such features. For example, the query support for semantic SPARQL queries is generally

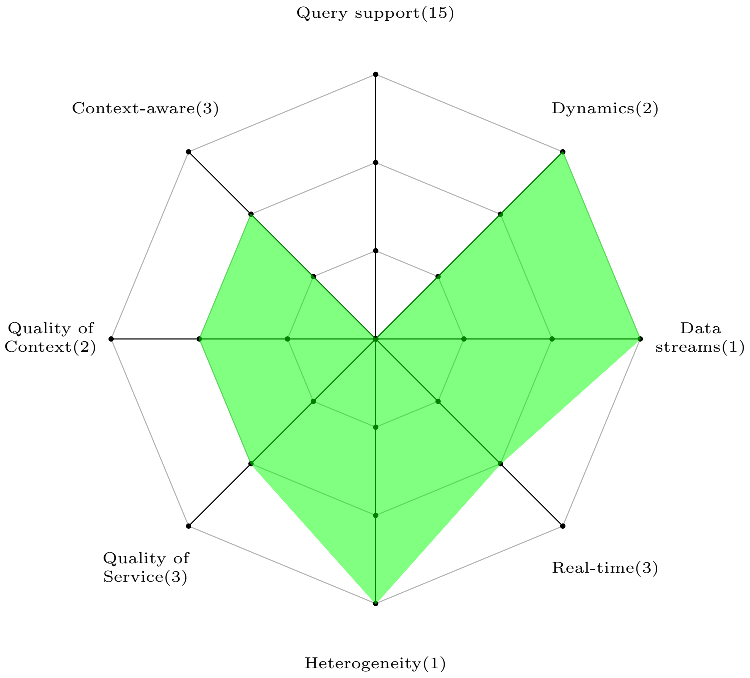


Figure 3.3: Maximum fulfillment of resource criteria.

supported for most ontology-based modeling approaches. However, none of the presented research applies semantic queries for resource discovery. Figure 3.3 shows the fulfillment requirements in resource management and the maximum reached level of fulfillment over all presented approaches. The number in parentheses represents the number of approaches which reach the maximum level. Based on the requirements analysis, we identified the following research gaps in IoT resource management:

Real-time: There exists no approach with support for contract-based real-time support in IoT resource processing. At least, use case specific real-time processing is supported by ROA, BPEL4IoT and SPU. ROA provides support for the low latency CoAP messaging protocol and generates target platform code to omit the use of a resource intensive WfMS [DTB+15]. This allows for real-time resource allocation and invocation of service methods. The round trip time of retrieving a reading from a single temperature sensor was 375 ms at average. Even though this latency is too high for most real-time systems, e. g., robots and autonomous cars, more relaxed settings in the Ambient Assisted Living (AAL) for detecting critical health conditions can be supported. For example, Dağtas et al. propose an approach for real-time and secure wireless health monitoring, where data packages are built and transferred at a minimum speed of 280 ms [DPS+08]. In general, the processing overhead of a WfMS and the standard HTTP-based messaging are not tailored for supporting real-time processing capabilities. Higher workflow complexity with an increasing number of control-flow structures only amplify this problem. In the course of this work, we will not aim at providing real-time data messaging and processing for IoT-aware workflows. However, we provide a performance evaluation of the resource discovery and invocation phase within this work.

Quality of Service: Also, QoS features are considered in many existing approaches to specify quality characteristics of services and use them for subsequent quality-aware resource discovery, these quality characteristics are mostly statically assigned. In an IoT-aware workflow scenario, services represent a real world *Thing* with associated context features. Therefore, service quality needs to be monitored and dynamically changed due to runtime context changes. For example, a smartphone may provide limited data processing capabilities when an energy-saving mode is activated. Therefore, we will consider the definition and employment of QoS characteristics for functionalities of *Things*. These are not comparable to existing QoS descriptions from the Service-oriented Architecture (SOA) field, as the quality characteristics refer to the sensing and manipulation of real world context.

Quality of Context: The QoC requirement is fulfilled by CANthings and CM4BPM, such that context parameters can be specified and there exists an update mechanism for a near real-time consistent representation of real-world context [DSR15, SRN15]. However, none of the approaches enable a real-time consistent representation of real-world context. In this work, we are also not aiming to fill this research gap, but we still consider a near real-time consistent context model as a necessity for allowing a dynamic and context-sensitive resource discovery in the IoT. In contrast, when omitting an update mechanism for *Thing*-related context information, resource discovery may result in the selection of unavailable or even unintended IoT services.

Context-aware: The requirement for context-aware modeling of *Things*, their capabilities and relations is key for enabling a context-sensitive resource discovery. The concepts in CANthings, MOPAL and BPMN4CPS allow for the modeling of location-based context and context-sensitive capabilities [DSR15, PRS+13, GKGK16]. None of the presented approaches consider the modeling of context-sensitive relations between *Things*. For example, an actuator for controlling the heating in a smart home has an effect on temperature sensor readings. Furthermore, if sensors and actuators are part of the same robot platform, this relation can

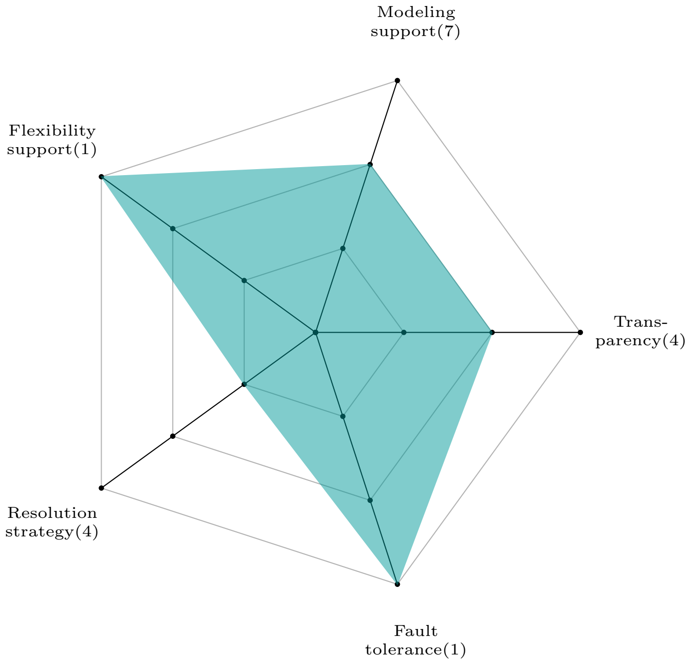


Figure 3.4: Maximum fulfillment of workflow criteria.

be represented as part of the context model and used for resource discovery. With this work, we aim at providing a context modeling approach to capture location-based context, context-sensitive capabilities as well as relation between *Things*.

Query support: Query support within the resource discovery phase in IoT-aware workflow management is not enabled by any of the presented approaches. Even though resource discovery often uses service descriptions, tags or other characteristics, queries provide a more expressive approach by specifying expressive context and quality related constraints. Within in this work, we aim at providing semantic query support for IoT resource discovery.

Figure 3.4 shows the fulfillment requirements in workflow management and the maximum reached level of fulfillment over all presented approaches. The number in parentheses represents the number of approaches which reach the maximum level. There exists no single approach covering all requirements at the shown maximum levels. Based on the requirements analysis, we identified the following research gaps in IoT-aware workflow management:

Modeling support: The modeling support for integrating *Things* as workflow resources is limited to IoT sensors and actuators within existing work. When using a service abstraction for encapuslating more complex capabilities of *Things*, the context-awareness of resource discovery is no longer given. The presented service-based approaches do not use a semantic description for more complex combinations of sensing and actuating tasks as methods of an IoT service. Especially the context-dependent availability of *Thing* capabilities are not considered yet. Therefore, this work aims at integrating context-aware *Things* with complex capabilities as workflow resources.

Transparency: The transparency for process modelers is supported for modeling resource requirements and constraints for each activity. However, in an ubiquitous system with ever growing number of devices and device types their context representation will also evolve in time. To decouple the evolving context-model from the definition of requirements and cosntraints inside a workflow model, a process modeler can specify the intention of each activity. Then, another model can provide the mapping between the intentions and the context model such that workflow models do not have to be adapted to changing environments. Therefore, this work aims at providing a concept for specifying the intention of an activity and to decouple the workflow model from the context model to increase reusability and separation of concerns.

Resolution strategy: The resolution strategy for resolving the ambiguity within the resource discovery phase in existing work is based on including context parameters and apply a random selection of one of the discovered resources. To further minimize the resource ambiguity, we aim at adding QoC and QoS constraints into our resolution strategy. The addition of an update mechanism for enabling a certain level of QoC results in a nea real-time consistent virtual representation of the physical context. Therefore, the selection of a fitting resource is more likely to result in an available resource within a fitting context for executing the workflow activity. Furthermore, the addition of QoS limits the search space and provides more accurate results with respect to the desired resource.

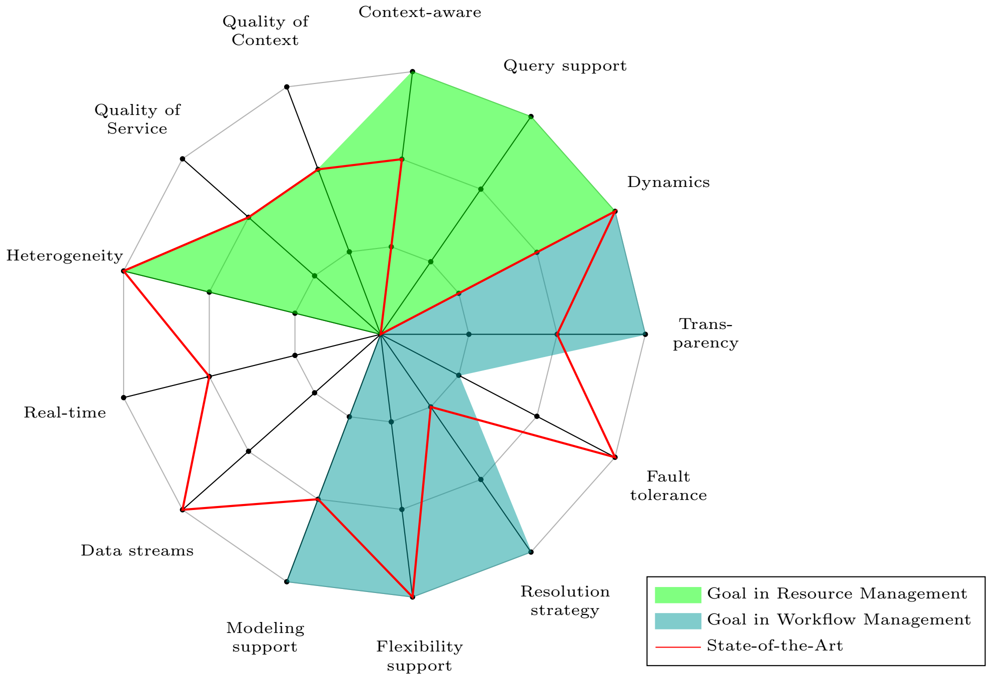


Figure 3.5: Target level of fulfillment within this work.

3.4 Approach

In the following, we present our approach to bridge the identified research gaps by fulfilling the requirement levels as shown in Figure 3.5. The green area shows the target level of requirement fulfillment in the field of resource management. The light blue area shows the target level of requirement fulfillment in the field of workflow management. The red line represents the maximum level of fulfillment for all requirements by all presented approaches. However, there exists no single approach that reaches all maximum levels in each category. As shown in Figure 3.5, this work mainly contributes to the fulfillment of the requirements: **Query support**, **context-awareness**, **modeling support** and **resolution strategy**. To reach the targeted levels of fulfillment in the respective requirements, we provide the following contributions to IoT-aware workflow management in alignment with the existing phases in the resource and workflow management perspectives.

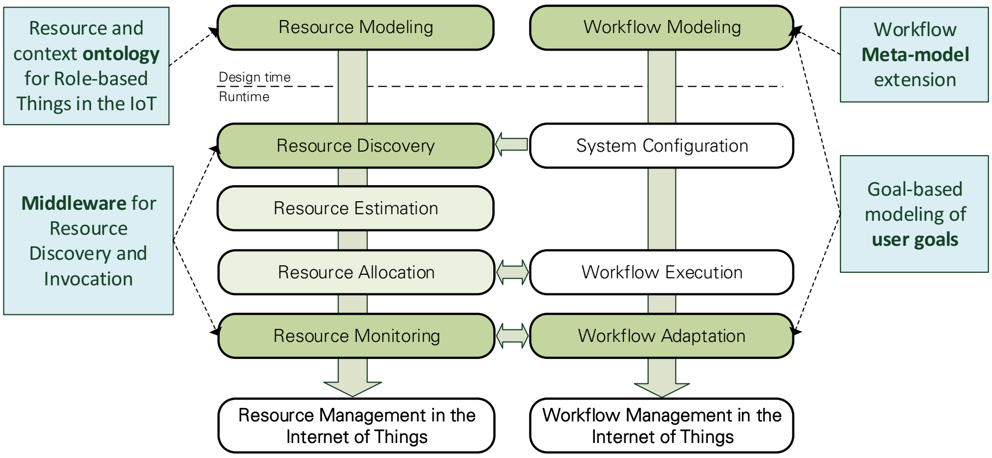

Resource and context **ontology** for Role-based Things in the IoT
Resource Modeling
Workflow Modeling
Workflow **Meta-model** extension
Design time
Runtime
Resource Discovery
System Configuration
Middleware for Resource Discovery and Invocation
Resource Estimation
Goal-based modeling of **user goals**
Resource Allocation
Workflow Execution
Resource Monitoring
Workflow Adaptation
Resource Management in the Internet of Things
Workflow Management in the Internet of Things

Figure 3.6: Approach in alignment with resource and workflow management.

3.4.1 Contribution to IoT-aware workflow management

Figure 3.6 shows the four contributions of this work and their classification into the phases of IoT-aware resource and workflow management. The contributions are:

- A **role-based** resource and context ontology
- A workflow **metamodel** extension
- A **goal model** for high level user goals
- A **middleware** for resource discovery and invocation

We partitioned the contributions into three separate perspectives, which are used to structure the following chapters. The resource perspective includes the role-based resource and context ontology to allow for the modeling of context-sensitive capabilities of *Things* and their relations. The workflow perspective includes the workflow meta-model extension to include query-based resource discovery and invocation as well as the more flexible high level user goals. The user perspective includes a goal model to specify such high level user goals in relation to the capabilities of role-based *Things*. The middleware is related to the runtime phase of each of the three perspectives. Table 3.8 provides an overview of the contributions and their approach to fulfill the requirements.

Table 3.8: Contributions and their relation to requirements.

Requirement	Notation	Related Contributions	Approach
Dynamics	RQ_Dyn	Middleware, Ontology	Semantic resource discovery based on role-based ontology.
Query support	RQ_QS	Middleware, Ontology, Workflow Meta-model	Specify SPARQL query in new activity type. Query includes ontology roles and constraints. Middleware executes query on knowledge base and invokes selected service method.
Context-aware	RQ_CA	Ontology	Modeling role-based capabilities of Things.
Quality of Context	RQ_QoC	Ontology, Middleware	Specify and execute SWRL rules to update the context ontology.
Quality of Service	RQ_QoS	Goal model, Workflow Meta-model, Middleware	Specify the quality characteristics in relation to high level user goals. Use goal-based activity types. Middleware translates high level user goals while results are ordered by fulfillment of quality characteristics.
Heterogeneity	RQ_Het	Ontology	Modeling Things and Roles.
Modeling support	RQ_MS	Workflow Meta-model	Specify requirements and constraints for Things in special activity types.
Flexibility support	RQ_FS	Ontology, Middleware	Trigger re-allocation for Things after context (role) change.
Resolution strategy	RQ_RS	Ontology, Goal model, Middleware	Use QoS from goal model and QoC from ontology to select best fit resources
Fault tolerance	RQ_FT	Middleware	Provide re-allocation for unavailable resources or context changes.
Transparency	RQ_Tra	Goal model, Workflow Meta-model	Use high level goals in special activity types.

3.5 Summary

In this chapter, we introduced challenges and requirements for integrating *Things* as workflow resources both from the resource management and the workflow management perspective. We reduced these challanges towards the focus of this work, defined in the research questions. Based on the requirements, we derived a classification scheme with four levels of fulfillment. Then, we reviewed and classified related work using the classification scheme. Subsequently, we limited the scope of our contributions to the identifed research gaps in the related work. Finally, we presented the four main goals for contributions of this work and the respective approaches to cover the target requirements.

4 Concept for Adaptive Workflow Activities in the IoT

In this chapter [1], we present the concepts for adaptive workflow activities in the IoT. We divided the contributions of this work into the three perspectives: *Resource*, *User* and *Workflow Perspective*. First, we introduce the relevant concepts for the context-sensitive modeling of *Things* as workflow resources in the *Resource Perspective*. Then, we present the concepts for modeling high-level user goals and linking them to the resource model as part of the *User Perspective*. Finally, we present the concepts for extending a workflow metamodel to support the integration of query-based and goal-based resource discovery in the IoT. In addition, we introduce the middleware concept for the *Semantic Access Layer* to implement the discovery and invocation of IoT services.

4.1 Resource Perspective

In this section, we provide the foundation for modeling *Things* in the IoT with the aim of enabling the integration of *Things* as workflow resources.

4.1.1 Role-based Things

As we identified the need for modeling context-sensitive capabilities of Things, their role-based modeling presents a suitable approach. Within a role-based model, the relation between a *Thing* and its capabilities can be expressed as context-dependent in the following way: A Things can have several capabilities, e. g., a smartphone can be used as a telephone but also for browsing the internet. The capabilities are grouped into several non-exclusive sets, called *Roles*. The ability of a *Thing* to play several *Roles* is covered at design time. At runtime, only the *Roles* that are allowed to be played in the current context are available to a *Thing*. In the following, we will discuss and select the relevant role modeling features by Steinmann and Kühn presented in Table 2.3 and Table2.3 in Chapter 2.1.2.

---

[1] This chapter is partially based on [HSK$^+$16, HSS16]

Table 4.1: Relevance of role modeling features in this work.

Nr.	Definition	Relevance
1	A role comes with its own properties and behaviour.	✓
2	Roles depend on relationships.	✓
3	An object may play different roles simultaneously.	✓
4	An object may play the same role several times, simultaneously.	✓
5	An object may acquire and abandon roles dynamically.	✓
6	The sequence in which roles may be acquired and relinquished can be subject to restrictions.	-
7	Objects of unrelated types can play the same role.	✓
8	Roles can play roles.	-
9	A role can be transferred from one object to another.	-
10	The state of an object can be role-specific.	-
11	Features of an object can be role-specific.	-
12	Roles restrict access.	✓
13	Different roles may share structure and behaviour.	✓
14	An object and its roles share identity.	✓
15	An object and its roles have different identities.	-
16	Relationships between roles can be constrained.	✓
17	There may be constraints between relationships.	-
18	Roles can be grouped and constrained together.	-
19	Roles depend on compartments.	✓
20	Compartments have properties and behaviors.	-
21	A role can be part of several compartments.	✓
22	Compartments may play roles like objects.	-
23	Compartments may play roles which are part of themselves.	-
24	Compartments can contain other compartments.	✓
25	Different compartments may share structure and behavior.	-
26	Compartments have their own identity.	-

Table 4.1 gives an overview of existing role modeling features [Ste00, KLG$^+$14] and their relevancy in this work. In the following we will discuss the role modeling features (RMF) in detail:

RMF 1: Regarding the survey of Steimann, this feature suggests that roles are represented as a first-class modeling concept (type) [Ste00]. In this work, roles are considered as types to enable the dynamic relationship between *Things* and their roles as well as the relationship between roles and capabilities. This feature is key for allowing the fulfillment of the requirement RQ_CA for context-awareness as shown in Table 3.8.

RMF 2: In this work, a role is only meaningful and can be utilized for resource discovery if there exists a relationship to a *Thing*. Therefore, we choose to model roles as depending on relationships. This feature contributes towards the fulfillment of RQ_Dyn and RQ_FS in Table 3.8.

RMF 3: Steimann identified this feature as one of the most broadly accepted properties of role modeling concepts [Ste00]. We also need to express that *Things* can play different roles simultaneously to cope with the concurrent access to IoT services within various contexts and workflows. For example, a smartphone can be used as an interaction device while offering a mobile hotspot. As workflows in IoT scenarios are likely to be executed concurrently, this feature can facilitate a high degree of capacity utilization.

RMF 4: The possibility for *Things* to play the same role i. e., to be related to several instances of the same role type by the play relation, is necessary to provide concurrent access to sensor data. For example, a temperature sensor can provide sensor data within concurrent workflows. However, in the case of actuating capabilities, we restrict the instantiation of roles to one at a time. Steimann argues that "the main reason to distinguish multiple occurrences in the same role is that each occurrence of the object in a role is associated with a different state" [Ste00]. In the case of *Things* in the real world, the state of an object is strictly singular. Therefore, if a role includes actuating capabilities, a *Thing* can only have one play relation to one instance of the role type.

RMF 5: In our approach, we use role instances to represent the runtime state of the interaction between a WfMS and the *Things* in the real world. That is, we instantiate roles and relate them to a *Thing* via the play relation to either retrieve sensor data or send commands to an encapsulated actuator in the context of one workflow execution. After the successful execution of a workflow, all workflow resources, i. e., traditional workflow resources and *Things*, are released. This results in the abandonment of related role instances. Therefore, a dynamic acquirement and abandonment of roles are required. This feature is key for the fulfillment of the requirement RQ_Dyn in Table 3.8.

RMF 6: Within our approach in the context of role-based *Things*, a sequential restriction for role acquisition and abandonment is not required. Roles are abandoned after each workflow execution, such that a preservation and restriction of related states is not useful.

RMF 7: Unrelated *Things* can provide the same capabilities, e. g., opening the shutters can provide an increase in room illumination during daytime while a light switch offers the same capability at nighttime.

RMF 8: We do not consider roles to be played by roles, as this modeling feature can lead to complex structures of role cascades which are not necessary for expressing the runtime state of relating a WfMS to *Things*.

RMF 9: Within our work, we do not consider a transfer of roles from one *Thing* to another due to the possibility of associated real world state. If a *Thing* becomes unavailable, the associated roles are abandoned and the affected workflow resources are re-allocated.

RMF 10: We do not consider the state of a *Thing* to be role-specific. Steimann argues that this feature suggests that each "role played by an object should be viewed as a separate instance of the object" [Ste00]. This modeling feature is not represented by the rigidity of the real world state of *Things*.

RMF 11: We do not consider roles to overwrite core behaviour of a *Thing*. Therefore, we do not employ the role modeling feature for specifying role-specific behaviour.

RMF 12: In the context of this work, roles restrict access to capabilities of *Things*. For example, a service robot in a smart home may not operate a full speed when a resident is in the room. This feature contributes towards the fulfillment of the requirement RQ_CA in Table 3.8.

RMF 13: We consider role inheritance for role types to be a useful feature. Within the resource discovery, the specification of a more general role can provide further flexibility for finding a resource at runtime.

RMF 14: Once a role is related to a *Thing* via the *play* relation, we consider the role to share the identity of the *Thing*.

RMF 15: This is already contradicted by using RMF 14.

RMF 16: In this work, we consider the restriction of role relationships as a useful feature to express mutual exclusive roles of *Things*.

RMF 17: We do not use the feature for inter-relationship constraints.

RMF 18: Roles of *Things* are considered individually in the context of this work.

RMF 19: In this work, roles depend on the current real world context in several ways. Depending on the real world context, the canPlay relation is updated such that the dynamics of available capabilities is captured. If a *Thing* plays a specific role at runtime and a context change removes the canPlay relation between the *Thing* and the role, it will be abandoned as well. Even though, Kühn et al. call these kind of situational context *compartments*, we continue to refer to them as *context*. This feature contributes towards the fulfillment of RQ_Dyn and RQ_CA in Table 3.8.

RMF 20: In this work, we consider context to be founded and situational. We do not include behavior as a part of a first-class modeling concept for context.

RMF 21: Especially, regarding sensing capabilities of *Things* the associated roles can be part of several contexts and workflow executions.

RMF 22: This is already prohibited by the negation of RMF 20.

RMF 23: This is already prohibited by the negation of RMF 22.

RMF 24: Contexts can be represented hierarchically within our work. For example, if the location of a *Thing* is a specific room in a smart home, then the assertion that the *Thing* is inside the smart home is also true.

RMF 25: This is already prohibited by the negation of RMF 20.

RMF 26: This is already prohibited by the negation of RMF 20.

The presented and selected role modeling features are the foundation for the creation of the semantic context model presented in Section 5.1.

4.1.2 Semantic Modeling Concepts

The semantic model represents the taxonomy of *Things* in relation to roles, capabilities, context and real world objects. In addition to the role modeling features discussed in the previous Section 4.1.1, the semantic model aims at bridging the heterogeneity of *Things* as stated in the requirement RQ_Het. For this purpose, we adapted the IoT reference model [BBDL^{+}13] towards the inclusion of a basic notion of context and *Things*.

Figure 4.1 shows the adapted IoT reference model[2]. At first, we omitted the service-related aspect of the original IoT reference model as our approach will include capabilities that relate to a specific service invocation. Next, we substituted the resource and virtual entity with the concept of *Thing*. The virtual entity of the IoT reference model enabled hierarchical structures of representing a physical entity. We shifted the hierarchy directly to the physical entity which in turn has attached devices. The devices themselves can be structured in a hierarchical manner as well via the contains relation. The simplification of the service and resource concepts of the original IoT reference model allows for a technology agnostic specification of *Things* and their capabilities. However, as the capabilities depend on roles we will introduce them as part of the modeling Section 5.1. For

---

[2] The IoT relevant part of the original IoT reference model is shown in Figure 2.2 in Section 2.1.1.

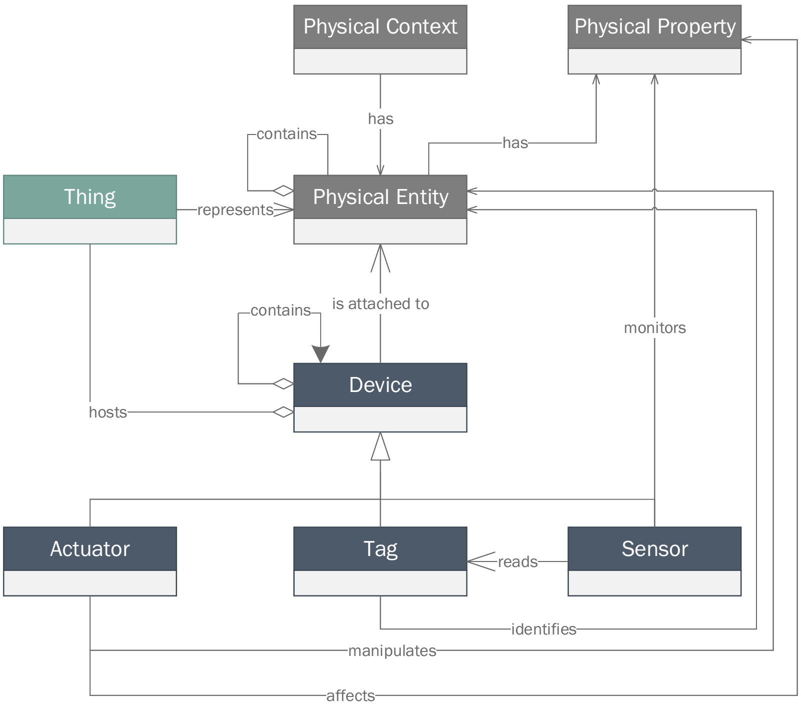


Figure 4.1: Adapted IoT reference model for heterogeneous *Things*.

now, the concept *Thing* is a virtual representation of a physical entity and hosts several devices in the form of actuators, tags and sensors. A *Thing* is modeled as a composition of these devices, because they are necessary features as part of the *Thing* specification introduced in Section 2.1.1. In addition, we added the concepts of physical context and physical property to the IoT reference model. The physical context includes a hierarchy of physical entities which in turn can have physical properties, e. g., temperature, humidity and illumination. The physical context is similar to the definition of compartments [KLG$^+$14]. However, the context properties are only related indirectly via the encapsulated physical entities. Moreover, a physical context does not include roles and does not represent situational templates but rather a virtual representation of the relation between physical entities. The presented conceptual model serves as a basis for the creation of the semantic model in Section 5.1 and contributes to the fulfillment of the requirement RQ_Het.

4.1.3 SWRL Modeling Concepts

The SWRL enables the expression of DL-safe[3] rules on top of a semantic model, i. e. OWL-DL ontology. Within this work, they are used to generate and update contextual knowledge of *Things*, capabilities, roles, and their relations. Due to the continuous update of these relations, the rule-based reasoning contributes to the fulfillment of the requirement RQ_QoC. The inferred part of the semantic model needs to update the existence of the relation between *Things* and playable roles and can be restricted by a physical context or a physical property of the represented physical entity. The modeling of the SWRL rules in alignment with the created ontology is presented in Section 5.1.

4.2 User Perspective

The *User Perspective* aims at supporting workflow modelers with both modeling of more precise semantic queries and modeling of high-level activity goals.

4.2.1 Semantic Queries in Workflow Activites

The integration of semantic queries, i. e., SPARQL queries, into workflow activities aims at enabling a dynamic resource discovery and invocation at runtime [HSS16]. The query support for resource discovery has been identified as a requirement of resource management in the IoT [PP12]. As we use an ontological knowledge base, we choose to support SPARQL queries in workflow activities to select fitting resources based on the requirements and constraints defined by the workflow modeler. The utilization of semantic queries contributes to the requirement RQ_QS and also increases the transparency of resource discovery for the workflow modeler as required by RQ_Tra.

4.2.2 Goals for Workflow Activites

In extension to the utilization of semantic queries within workflow activities, we aim at providing an additional abstraction layer for workflow modelers to specify the goal, and the execution context of an activity. This goal is usually implicit and hidden in the definition of a semantic query or direct service invocation. As it contributes to the concept of *separation of concerns*, we do not presume domain knowledge about IoT device capabilities from the workflow modeling perspective.

---

[3] An introduction to the concept of rule-based reasoning with SWRL is given in Section 2.1.2.

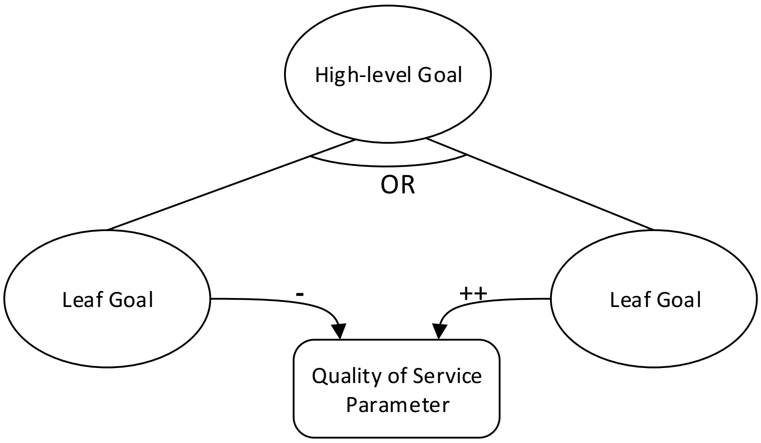


Figure 4.2: Relevant concepts of a Tropos goal model.

Therefore, we introduce the goal modeling perspective as an additional view on IoT-aware workflow modeling and contribution to the fulfillment of the requirement RQ_Tra. In this perspective, we model dependencies between informal activity goals and IoT device capabilities from the ontological domain model. Additionally, non-functional requirements, i. e., quality of service measures, can be expressed and contribute to the fulfillment of the requirement RQ_QoS. The informal goal definitions can be attributes of workflow activities and increase the *reusability* of workflow models. The goal model is represented by a Tropos goal graph, where leaf goals are executable activity descriptions. An additionally provided *mapping description* relates the leaf goals with ontological concepts of the domain ontology. Figure 4.2 shows an example goal model as it is used in this work. Here, high level goals can be logically composed (AND, OR) of leaf goals which in turn can contribute to or negate QoS parameter.

4.2.3 Mapping from Goals to Semantic Queries

The mapping description defines the relation of a leaf goal in a certain goal model to a partial SPARQL query using concepts of the domain ontology. For every leaf goal from the goal model, a mapping to a partial SPARQL query has to be defined. Based on these mappings, the Semantic Access Layer (SAL)[4] generates a domain specific SPARQL query that selects and invokes the intended IoT services to satisfy the activity goals. The generation also takes the user-specified context constraints into account, that are an optional part of the activity specification. Mapping descriptions are domain specific in the sense that they include prefixes and concepts of the domain ontology. When merging several partial SPARQL queries, prefixes, selected subjects and filters are accumulated.

4.3 Workflow Perspective

The workflow perspective includes the workflow modeling and the runtime resource management. To provide the workflow modeler the possibility to specify resource requirements and constraints per activity, we extend an existing workflow metamodel. Furthermore, the resource management, i. e., the resource discovery, estimation, allocation and monitoring, is realized through an external middleware. In the following, we introduce the concepts for the metamodel extension and the middleware layer.

4.3.1 Workflow metamodel Extensions

To provide the workflow modeler with support to specify the introduced semantic queries and goals per activity, we extended an existing generic workflow metamodel [SKNS14]. We choose this specific metamodel, as it already addresses workflow modeling for CPS. However, the metamodel extension is the only workflow modeling language specific aspect within our approach. Therefore, a minimal BPMN or WS-BPEL metamodel extension will enable the integration of our contributions with well established WfMS.

In the case of a goal-based activity, we propose the *GoalBasedInvoke* activity type extension as seen in Figure 4.3. To specify the goals of an activity, workflow modelers provide a goal from a goal model. Additionally, context constraints can be defined in the form of restrictions over ontological concepts as part of the *GoalBasedInvoke*. Thus, we are able to specify that only IoT devices in a certain context, e. g., within a specific location, are to be considered for the fulfillment of the specified goals. Workflow modelers are also enabled to specify non-functional requirements if the goal model supports quality metrics, e. g., a goal should be satisfied without violating privacy. When transferred to the SAL, the user-defined

---

[4] The Semantic Access Layer (SAL) is introduced in Section 4.3.2.

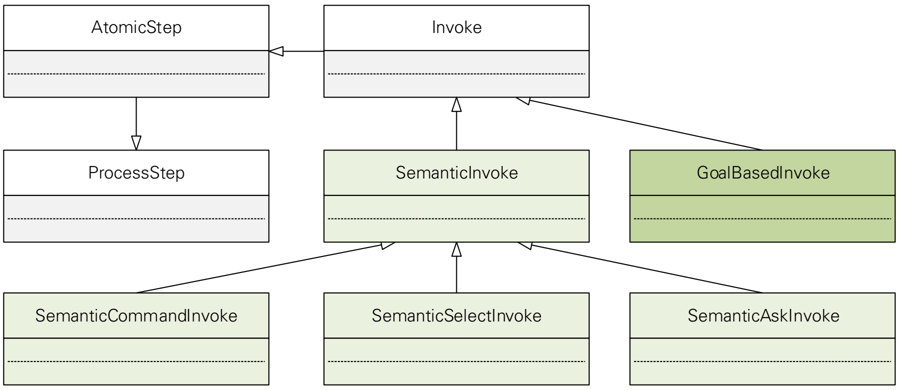


Figure 4.3: Extension of the PROtEUS activity metamodel

activity goals are used as input for the *backward reasoning* SAT solver.

In the case of a user-defined SPARQL query, we differentiate between actuating and sensing capabilities of IoT devices and introduce novel activity types for both, because the flow of the workflows depends on whether it is a consuming or producing activity. Activities that generate data, e. g., sensor data or device identifiers, need data ports to pass this information on to succeeding steps. This is realised by the *SemanticSelectInvoke*. In contrast, the *SemanticCommandInvoke* activity type includes SPARQL queries for invoking actuators and therefore does not produce data. Additionally, we introduce the *SemanticAskInvoke* activity type, which evaluates queries on a Boolean basis. The result is propagated to the activity's data ports and can then be used for control-flow adaptations. Figure 4.3 depicts the *SemanticInvoke* and GoalBasedInvoke as subclasses of the *AtomicStep* class, i. e., a non-composable workflow activity. *SemanticSelectInvoke*, *Semantic-CommandInvoke*, and *SemanticAskInvoke* are subclasses of *SemanticInvoke*. They can be instantiated by workflow modelers using an extended version of an existing workflow modeling tool [SKNS14]. Upon creating a activity of type *SemanticInvoke* in the workflow model, the modeler has to specify the SPARQL query.

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX iot: <http://www.semanticweb.org/role_based_iot_ontology#>
SELECT DISTINCT ?subject ?cap
WHERE {
   ?cap rdf:type iot:Capability .
   ?subject iot:has ?cap .
   ?cap iot:hasCommand ?cmd .
   ?cmd a iot:OnCommand .
}
```

Listing 4.1: Example SPARQL query with command to activate available *Things*

Listing 4.1 shows an example of an SPARQL query from a *SemanticCommandInvoke* activity. To execute this query, the SAL needs to include the ontology for modeling role based *Things*[5]. The example *SemanticCommandInvoke* statement includes a SPARQL query and a semantic command. When executed, the SAL generates a list of IoT services from the query result. The semantic command is then transformed into service calls for each IoT service. All of the introduced activity types use the SAL as a middleware service. The proposed metamodel extensions contribute towards the fulfillment of the requirements RQ_MS and RQ_Tra. The general modeling support for specifying resource requirements and constraints per activity is enabled. In addition, the transparency is maintained both for the explicit modeling of semantic queries and the implicit modeling of activity goals.

4.3.2 Middleware for Dynamic Resource Discovery and Allocation

The SAL is the middleware layer for the dynamic discovery of IoT resources and the subsequent service invocation. Therefore, the SAL provides the support for interpreting both query-based and goal-based activities. The queries are directly executed on an integrated knowledge base holding the IoT domain ontology. Goals are first translated into executable leaf goals via *backward reasoning*. The associated mapping descriptions provide the actual semantic queries, which are then again executed on the integrated knowledge base. The query execution results in a set of feasible *Things* for either retrieving the desired sensor data or activating an actuator. In this matter, the SAL contributes to the fulfillment of the requirement for query support RQ_QS and the requirement for dynamic resource discovery RQ_Dyn. In the case of a goal-based resource discovery, also QoS parameters are

---

[5] The ontology is presented in Section 5.1

considered. This contributes to the fulfillment of RQ_QoS. Furthermore, the SAL aims at providing the flexibility support and fault tolerance defined by the requirements RQ_FS and RQ_FT, by monitoring context changes and triggering the re-allocation of unavailable or unsuitable resources. To this end, its SWRL rules are executed continuously to reflect the real world context changes in the knowledge base. This includes the relation between *Things* and its playable roles. This continuous update of the context model contributes to RQ_QoC. The architecture of the SAL is presented in Section 6.1.

4.4 Summary

In this chapter, we presented the concepts of our contributions towards enabling an activity-level adaptation of workflows in the IoT. We identified the relevant role modeling features to include in the semantic model for IoT resources from a workflow management perspective. Furthermore, we adapted the IoT reference model to bridge the heterogeneity of *Things* and to provide a basic taxonomy for the creation of the semantic model. In addition, we introduced the concept of SWRL rules as a update mechanism for real world context changes. From the user perspective, we addressed the query-based and goal-based activity modeling as well as the mapping from goals to semantic queries. Consequently, we introduced an extension to a workflow metamodel to capture the goal-based and query-based activity types. Finally, we introduced the basic concepts of the SAL as a middleware layer between a WfMS and the actual IoT services.

5 Modeling Adaptive Workflow Activities in the IoT

In this chapter [1], we present our models for adaptive workflow activities in the IoT. First, we introduce the TBox of the ontology for role-based *Things* in the IoT as one of the main contributions of this work within the resource perspective. As the modeling of the ABox of the ontology, the user goal model, the mapping from goals to semantic queries and the workflow modeling are application domain specific, we revisit and extend the application scenario from Section 1.2 and provide the respective models within each perspective. We hereby show how a domain-specific ABox can be developed for other areas of application.

5.1 Resource Perspective

In this section, we present the ontology for modeling role-based *Things* with context-sensitive capabilities in the IoT including the SWRL rules for continuous context updates as one of the major contributions of this work. First, we introduce the taxonomy of the ontology and give an overview of the included concepts as well as the corresponding relations. Subsequently, we describe each class, object property, and data property in detail, following the description logic (DL) notation. Finally, we present the SWRL rules and their purpose for representing real-time context changes in the ontological model.

5.1.1 Role-based Modeling of Context-sensitive Things

Figure 5.1 gives an overview of the taxonomy of the created OWL ontology, showing classes as well as static and inferred object property assertions[2]. In general, a *Thing* is an object of the real world with augmented virtual properties, e. g., a smartphone, a service robot or a tagged product. A *Thing* can have several devices

---

[1] This chapter is partially based on [HSK+16, HSS16]

[2] The complete TBox of the created OWL ontology can be found in the Appendix Section10.2.

of the types *Sensor*, *Actuator* or *Tag.* A *Sensor* can read a *Tag* or sense a *PhysicalProperty* of a *PhysicalEntity.* The latter can contain themselves to represent arbitrary hierarchies, e. g., a house can contain rooms which in turn may contain residents and other physical objects. An *Actuator* can manipulate a *PhysicalEntity* or its *PhysicalProperty.* The manipulation of a *PhysicalProperty* may be modeled directly or indirectly via the *PhysicalEntity*, e. g., an actuator may turn on the heater which in turn *manipulates* the room temperature. An *Actuator* can affect a *Sensor* by manipulating a *PhysicalProperty* which is sensed by the *Sensor.* In turn, then the *Sensor* can monitor the effects of the *Actuator.* As these relations are highly dynamic for mobile *Things*, they are inferred rather than statically modeled. A *PhysicalContext* can have several *PhysicalEntities*, e. g., a Smart Home context includes all physical entities of the house or apartment. Furthermore, a *Thing* can be attached to a *PhysicalEntity*, e. g., a SmartWatch is attached to a person, and available in a certain *PhysicalContext*, e. g., a smartwatch which is attached to a resident who is at home, is available in the context of the Smart Home and may provide sensing capabilities for measuring the room temperature within this context. The object property assertion *isAvailableIn* can be modeled directly or be inferred from the relation *isAttachedTo* between a *Thing* and a *PhysicalEntity* and the relation *hasEntity* between a *PhysicalContext* and the same *PhysicalEntity.*

Most of the above concepts are adapted from the IoT reference model [BBDL$^+$13]. In contrast, the *Role* and *Capability* concepts are key contributions of this work. A *Thing* can be related to a *Role* by the *plays* and *canPlay* relation. Both are inferred from the runtime context, e. g., by continuously executing SWRL rules. These specify which *Thing* can play which *Role* in a certain situational context. The *plays* relation is created upon allocating a *Thing* in a specific *Role* for the execution of an activity. Furthermore, a *Role* can be played for a specific *PhysicalEntity* or *PhysicalContext.* Here, we also differentiate between the relations *canBePlayedFor* and *isPlayedFor.* The first is again created dynamically depending on the runtime context and domain specific SWRL rules. The latter is created upon allocating a *Role* within a specific activity. In general, the *canPlay* and *canBePlayedFor* object property assertions are changed dynamically according to the runtime context and reflect the real world state and possibility of interactions between a *Thing* and a *PhysicalEntity* or a *PhysicalContext.* In contrast, the *plays* and *isPlayedFor* object property assertions represent the virtual state of interactions between a *Thing* and a *PhysicalEntity* or a *PhysicalContext.* By allowing both states to exist in parallel inside the ontology, we allow for the detection of inconsistencies between the cyber and the physical states and provide the foundation for their resolution. Such inconsistencies are likely to occur when allowing *Things* and *PhysicalEntities*

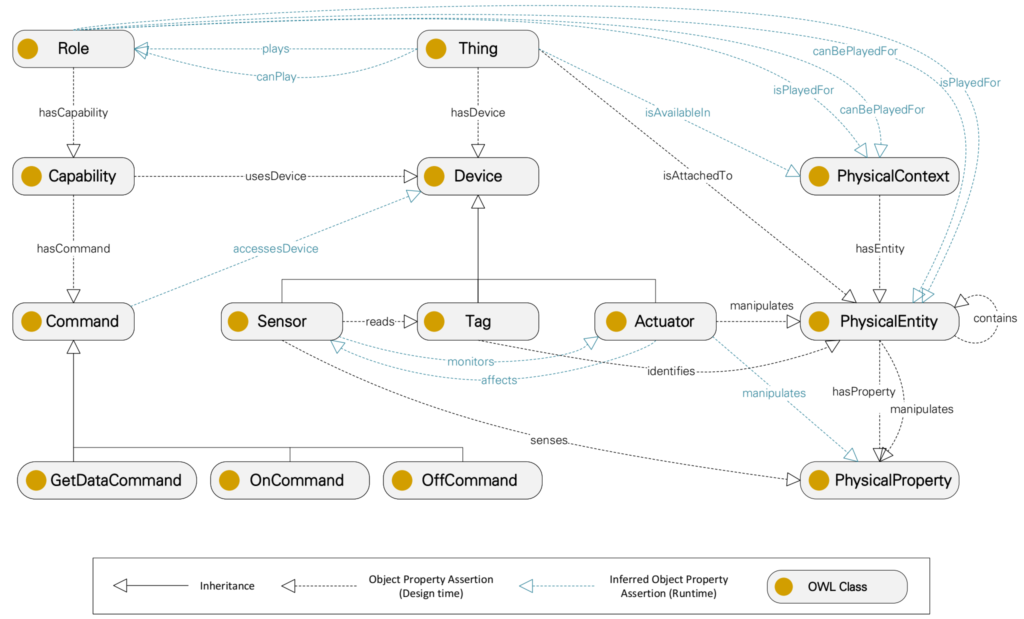


Figure 5.1: OWL ontology for role-based *Things* in the IoT.

to be mobile. For example, an interaction device may only provide its capabilities to a person when they are in the same physical context. Furthermore, a smartphone may only provide the capability to record a video or take a picture if the battery charge is above a certain threshold. Therefore, we modeled all capabilities to be context-sensitive, i. e., to be available only in some and not all possible situational contexts. A *Role* can have several *Capabilities* which in turn use some *Device*, i. e., a *Sensor*, *Actuator* or *Tag*. The *Capability* can have several *Commands* which refer to a functionality of a *Device*. The *accessesDevice* relation between a *Command* and a *Device* is inferred from the existence of the *usesDevice* and *hasCommand* object property assertions. Furthermore, *Commands* are classified to be either a *GetDataCommand*, *OnCommand* or *OffCommand*. The *GetDataCommand* can be used for service calls to retrieve data, e. g., a sensor reading. Also blocking data retrieval from user interactions are considered as a *GetDataCommand*. The *OnCommand* can be used to trigger an *Actuator* or to execute a service method, e. g., turning on the heater or invoking a REST service call. The *OffCommand* can be used for switching off a stateful *Actuator*. Even though these *Command* types are highly simplified from the variety of real world states, they provide a useful abstraction in many application scenarios, e. g., in the Smart Home domain.

5.1.2 Ontology Classes

In the following, we present the ontology classes in alphabetical order and discuss their purpose for modeling context-sensitive *Things* in the IoT.

Actuator

An *Actuator* is a subclass of the *Device* type. It can *manipulate* a *PhysicalEntity*, a *PhysicalProperty* and *affect* a *Sensor*. Typical actuators in a Smart Home scenario are heating controls, door openers, light switches but also applications on a smartphone. The *Actuator* class is defined by the following DL notation:

$$
\begin{aligned}
&\text{Actuator} \sqsubseteq \text{Device}\\
&\text{Actuator} \sqsubseteq \exists\ \text{manipulates PhysicalEntity}\\
&\text{Actuator} \sqsubseteq \exists\ \text{manipulates PhysicalProperty}\\
&\text{Actuator} \sqsubseteq \exists\ \text{affects Sensor}
\end{aligned}
$$

Capability

The *Capability* class is essential for specifying context-sensitive functionalities of *Things*. It represents an isolated *Device* capability, e. g., sensing some *PhysicalProperty*. A capability can have several *Commands* and is related to a specific *Device* via the *usesDevice* object property assertion. The *Capability* class is defined by the following DL notation:

$$
\begin{aligned}
&\text{Capability} \sqsubseteq \exists\ \text{hasCommand Command}\\
&\text{Capability} \sqsubseteq \exists\ \text{usesDevice Device}
\end{aligned}
$$

Command

The *Command* class serves as superclass for the specialized command types. It includes the inferred object property assertion *accessesDevice* which is created for individuals at runtime by executing a SWRL rule. Furthermore, a *Command* includes a *serviceMethodURL* to specify the related service method for invocation. The specialized command types inherit this relation. The *Command* class is defined by the following DL notation:

$$\text{Command} \sqsubseteq \exists\ \text{serviceMethodURL Datatype\#anyURI}$$
$$\text{Command} \sqsubseteq \exists\ \text{accessesDevice Device}$$

GetDataCommand

The *GetDataCommand* class inherits the *accessesDevice* relation and is used to retrieve sensor data or data in general from a *Device*. The *GetDataCommand* class is defined by the following DL notation:

$$\text{GetDataCommand} \sqsubseteq \text{Command}$$

OffCommand

The *OffCommand* class inherits the *accessesDevice* relation and is used define commands for switching off *Actuators*. The *OffCommand* class is defined by the following DL notation:

$$\text{OffCommand} \sqsubseteq \text{Command}$$

OnCommand

The *OnCommand* class inherits the *accessesDevice* relation and is used define commands for switching on *Actuators* and invoke other IoT services, e. g., use a service method within a mobile application. The *OnCommand* class is defined by the following DL notation:

$$\text{OnCommand} \sqsubseteq \text{Command}$$

PhysicalContext

The *PhysicalContext* class represents situational context within the real world. A *PhysicalContext* can include several *PhysicalEntities* via the *hasEnity* relation. As *PhysicalEntities* can *contain* themselves, the contained *PhysicalEntities* are also associated within the *PhysicalContext*. Therefore, it is only possible to create real subsets of *PhysicalEntities* within one *PhysicalContext* if the entities exist in a parallel hierarchy or on the same hierarchical level of one hierarchy. The *PhysicalContext* class is defined by the following DL notation:

$$\text{PhysicalContext} \sqsubseteq \exists\ \text{hasEntity PhysicalEntity}$$

PhysicalEntity

The *PhysicalEntity* class represents all physical objects and beings, which are not augmented by a cyber component, i. e., in contrast such entities are classified as *Things*. A *PhysicalEntity* may contain other *PhysicalEntities*, providing the possibility of arbitrary hierarchical structures. Furthermore, a *PhysicalEntity* may manipulate a *PhysicalProperty*, e. g., a heater increases the room temperature, and can have several *PhysicalProperty*, e. g., humidity. The *PhysicalEntity* class is defined by the following DL notation:

$$\begin{aligned}&\text{PhysicalEntity} \sqsubseteq \exists \text{ contains PhysicalEntity}\\&\text{PhysicalEntity} \sqsubseteq \exists \text{ manipulates PhysicalProperty}\\&\text{PhysicalEntity} \sqsubseteq \exists \text{ hasProperty PhysicalProperty}\end{aligned}$$

PhysicalProperty

The *PhysicalProperty* class represents properties of *PhysicalEntities* and has some data defined by the data property *physicalPropertyData*. The data value represents the current real world value of the property within individuals. The *PhysicalProperty* class is defined by the following DL notation:

$$\text{PhysicalProperty} \sqsubseteq = \text{physicalPropertyData}$$

Role

The *Role* class is a key concept for the modeling of context-sensitive *Capabilities* of *Things* in the IoT. A *Role* can have several *Capabilities* and creates subsets over all *Capabilities* of a *Device*. Access to *Roles* can be restricted to certain situational contexts or ranges of *PhysicalProperties* by the inferred *canPlay* relation. Therefore, the *Role* concept provides context-sensitive views of *Device Capabilities*. The *Role* class is defined by the following DL notation:

$$\text{Role} \sqsubseteq \exists \text{ hasCapability Capability}$$

Sensor

A *Sensor* is a subclass of the *Device* type. It can *read* a *Tag*, monitor an *Actuator* and *sense* a *PhysicalProperty*. Typical sensors in a Smart Home scenario are temperature, humidity, and illumination sensors but also user interfaces. The *Sensor* class is defined by the following DL notation:

$$\begin{aligned}&\text{Sensor} \sqsubseteq \text{Device}\\&\text{Sensor} \sqsubseteq \exists\ \text{reads Tag}\\&\text{Sensor} \sqsubseteq \exists\ \text{monitors Actuator}\\&\text{Sensor} \sqsubseteq \exists\ \text{senses PhysicalProperty}\end{aligned}$$

Tag

A *Tag* is a subclass of the *Device* type. It can provide identification for a *PhysicalEntity*. The *Tag* class is defined by the following DL notation:

$$\begin{aligned}&\text{Tag} \sqsubseteq \text{Device}\\&\text{Tag} \sqsubseteq \exists\ \text{identifies PhysicalEntity}\end{aligned}$$

Thing

The *Thing* class is an essential concept for modeling context-sensitive capabilities of *Things* in the IoT. It is a virtual representation of a physical object with an augmentation in the cyber world, e. g., a unique identifier or sensing capabilities to produce data. A *Thing* can be related to a *Role* by the *canPlay* and the *plays* object property assertion. The *canPlay* relation expresses the possibility to activate a *Role* for a certain *Thing* while the *plays* relation expresses such an activation at runtime. A *Thing* can have several *Devices*, i. e., sensors, actuators and tags. Furthermore, a *Things* can be attached to a *PhysicalEntity*. In this case, certain *PhysicalProperties* of the entity can be inferred for the *Thing*, e. g., location. The *Thing* class is defined by the following DL notation:

$$\begin{aligned}&\text{Thing} \sqsubseteq \exists\ \text{canPlay Role}\\&\text{Thing} \sqsubseteq \exists\ \text{plays Role}\\&\text{Thing} \sqsubseteq \exists\ \text{hasDevice Device}\\&\text{Thing} \sqsubseteq \exists\ \text{isAttachedTo PhysicalEntity}\end{aligned}$$

5.1.3 Ontology Object properties

In the following, we present the ontology object properties in alphabetical order and discuss their purpose for modeling context-sensitive *Things* in the IoT.

accessesDevice

The *accessesDevice* object property is inferred from the object property assertions *usesDevice* between a *Capability* and a *Device* as well as *hasCommand* between the same *Capability* and a *Command*. It is inferred between *Commands* and *Devices* to directly relate a specific command to the specific *Device* which can execute the *Command*. The *accessesDevice* object property is defined by the following DL notation:

$$
\begin{aligned}
&\text{accessesDevice} \sqsubseteq \text{topObjectProperty}\\
&\exists\ \text{accessesDevice Thing} \sqsubseteq \text{Command}\\
&\top \sqsubseteq \forall\ \text{accessesDevice Device}
\end{aligned}
$$

affects

The *affects* object property is inferred from the object property assertions *manipulates* between an *Actuator* and a *PhysicalProperty* as well as *senses* between a *Sensor* and the same *PhysicalProperty*. It is inferred between *Actuators* and *Sensors* to express that the effects of an *Actuator* can be measured by a *Sensor*. The *affects* object property is inverse to the *monitors* object property and defined by the following DL notation:

$$
\begin{aligned}
&\text{affects} \sqsubseteq \text{topObjectProperty}\\
&\text{affects} \equiv \text{monitors}^{-}\\
&\exists\ \text{affects Thing} \sqsubseteq \text{Actuator}\\
&\top \sqsubseteq \forall\ \text{affects Sensor}
\end{aligned}
$$

canBePlayedFor

The *canBePlayedFor* object property is inferred from the object property assertions *canPlay* between a *Thing* and a *Role* as well as *isAttachedTo* between the same *Thing* and a *PhysicalEntity* or *isAvailableIn* between the same *Thing* and a *PhysicalContext*. It is inferred between *Things* and *PhysicalEntity* or *PhysicalContext* to express the runtime availability of a playable *Role* for a certain entity or context. The *canBePlayedFor* object property is defined by the following DL notation:

$$
\begin{aligned}
&\text{canBePlayedFor} \sqsubseteq \text{topObjectProperty}\\
&\exists\ \text{canBePlayedFor Thing} \sqsubseteq \text{Role}\\
&\top \sqsubseteq \forall\ \text{canBePlayedFor PhysicalEntity}\\
&\top \sqsubseteq \forall\ \text{canBePlayedFor PhysicalContext}
\end{aligned}
$$

canPlay

The *canPlay* object property is inferred at runtime to express the possibility of a *Thing* to play a certain *Role*. The inference is domain specific and depends on the modeled SWRL rules to create a *canPlay* a certain situational context. The *canPlay* object property is defined by the following DL notation:

$$\begin{array}{l} \text{canPlay} \sqsubseteq \text{topObjectProperty} \\ \exists\ \text{canPlay Thing} \sqsubseteq \text{Thing} \\ \top \sqsubseteq \forall\ \text{canPlay Role} \end{array}$$

contains

The *contains* object property relates a *PhysicalEntity* to other *PhysicalEntities* and therefore enables hierarchical structuring of entities. The *contains* object property is transitive and defined by the following DL notation:

$$\begin{array}{l} \text{contains} \sqsubseteq \text{topObjectProperty} \\ \text{TransitiveProperty contains} \\ \exists\ \text{contains Thing} \sqsubseteq \text{PhysicalEntity} \\ \top \sqsubseteq \forall\ \text{contains PhysicalEntity} \end{array}$$

hasCapability

The *hasCapability* object property relates a *Role* to some *Capabilities* and is defined by the following DL notation:

$$\begin{array}{l} \text{hasCapability} \sqsubseteq \text{topObjectProperty} \\ \exists\ \text{hasCapability Thing} \sqsubseteq \text{Role} \\ \top \sqsubseteq \forall\ \text{hasCapability Capability} \end{array}$$

hasCommand

The *hasCommand* object property relates a *Capability* to some *Command* and is defined by the following DL notation:

$$\begin{array}{l} \text{hasCommand} \sqsubseteq \text{topObjectProperty} \\ \exists\ \text{hasCommand Thing} \sqsubseteq \text{Capability} \\ \top \sqsubseteq \forall\ \text{hasCommand Command} \end{array}$$

hasDevice

The *hasDevice* object property relates a *Thing* to some *Device* and is defined by the following DL notation:

$$\begin{aligned}&\text{hasDevice} \sqsubseteq \text{topObjectProperty}\\&\exists\ \text{hasDevice Thing} \sqsubseteq \text{Thing}\\&\top \sqsubseteq \forall\ \text{hasDevice Device}\end{aligned}$$

hasEntity

The *hasEntity* object property relates a *PhysicalContext* to some *PhysicalEntity* and is defined by the following DL notation:

$$\begin{aligned}&\text{hasEntity} \sqsubseteq \text{topObjectProperty}\\&\exists\ \text{hasEntity Thing} \sqsubseteq \text{PhysicalContext}\\&\top \sqsubseteq \forall\ \text{hasEntity PhysicalEntity}\end{aligned}$$

hasProperty

The *hasProperty* object property relates a *PhysicalEntity* to some *PhysicalProptery* and is defined by the following DL notation:

$$\begin{aligned}&\text{hasProperty} \sqsubseteq \text{topObjectProperty}\\&\exists\ \text{hasProperty Thing} \sqsubseteq \text{PhysicalEntity}\\&\top \sqsubseteq \forall\ \text{hasProperty PhysicalProperty}\end{aligned}$$

identifies

The *identifies* object property relates a *Tag* to some *PhysicalEntity* to express the ability of the *Tag* to provide identification information, e. g., a unique identifier (UID), for the specific *PhysicalEntity.* The *identifies* object property is defined by the following DL notation:

$$\begin{aligned}&\text{identifies} \sqsubseteq \text{topObjectProperty}\\&\exists\ \text{identifies Thing} \sqsubseteq \text{Tag}\\&\top \sqsubseteq \forall\ \text{identifies PhysicalEntity}\end{aligned}$$

isAllocatedTo

The *isAllocatedTo* object property is inferred at runtime from the object property assertions *plays* between a *Thing* and a *Role* as well as *isPlayedFor* between the same *Role* and a *PhysicalEntity* or *PhysicalContext*. It is inferred between *Things* and *PhysicalEntity* or *PhysicalContext* to express the runtime allocation of a *Thing* to a certain entity or context. The *isAllocatedTo* object property is defined by the following DL notation:

$$\begin{array}{l} \text{isAllocatedTo} \sqsubseteq \text{topObjectProperty} \\ \exists \text{ isAllocatedTo Thing} \sqsubseteq \text{Thing} \\ \top \sqsubseteq \forall \text{ isAllocatedTo (PhysicalEntity} \sqcup \text{PhysicalContext)} \end{array}$$

isAttachedTo

The *isAttachedTo* object property can be modeled between *Things* and *PhysicalEntities* to express their physical attachment. The relation can be used to infer *PhysicalProperties* from a *PhysicalEntity* to the attached *Thing*, e. g., location. The *isAttachedTo* object property is defined by the following DL notation:

$$\begin{array}{l} \text{isAttachedTo} \sqsubseteq \text{topObjectProperty} \\ \exists \text{ isAttachedTo Thing} \sqsubseteq \text{Device} \\ \top \sqsubseteq \forall \text{ isAttachedTo PhysicalEntity} \end{array}$$

isAvailableIn

The *isAvailableIn* object property is either modeled directly between a *Thing* and a *PhysicalContext* or inferred at runtime from the object property assertions *isAttachedTo* between a *Thing* and a *PhysicalEntity* as well as *hasEntity* between the a *PhysicalContext* and the same *PhysicalEntity*. It expresses the runtime availability of a *Thing* within a *PhysicalContext*. The *isAvailableIn* object property is defined by the following DL notation:

$$\begin{array}{l} \text{isAvailableIn} \sqsubseteq \text{topObjectProperty} \\ \exists \text{ isAvailableIn Thing} \sqsubseteq \text{Thing} \\ \top \sqsubseteq \forall \text{ isAvailableIn PhysicalContext} \end{array}$$

isPlayedFor

The *isPlayedFor* object property is inferred from the object property assertions *plays* between a *Thing* and a *Role* as well as *isAttachedTo* between the same *Thing* and a *PhysicalEntity* or *isAvailableIn* between the same *Thing* and a *PhysicalContext*. It is inferred between *Things* and *PhysicalEntity* or *PhysicalContext* to express the runtime allocation of a playable *Role* for a certain entity or context. The *isPlayedFor* object property is defined by the following DL notation:

$$\begin{array}{l}\text{isPlayedFor} \sqsubseteq \text{topObjectProperty}\\ \exists\ \text{isPlayedFor Thing} \sqsubseteq \text{Role}\\ \top \sqsubseteq \forall\ \text{isPlayedFor PhysicalContext}\\ \top \sqsubseteq \forall\ \text{isPlayedFor PhysicalEntity}\end{array}$$

manipulates

The *manipulates* object property can be modeled directly between an *Actuator* and a *PhysicalEntity*, *PhysicalEntity* and a *PhysicalProperty* as well as an *Actuator* and a *PhysicalProperty*. In addition, it can be inferred at runtime from the object property assertions *manipulates* between an *Actuator* and a *PhysicalEntity* as well as *manipulates* between the same *PhysicalEntity* and a *PhysicalProperty*. It expresses the potential effect an *Actuator* or a *PhysicalEntity* can cause. The *manipulates* object property is defined by the following DL notation:

$$\begin{array}{l}\text{manipulates} \sqsubseteq \text{topObjectProperty}\\ \text{TransitiveProperty manipulates}\\ \exists\ \text{manipulates Thing} \sqsubseteq \text{Actuator} \sqcup \text{PhysicalEntity}\\ \top \sqsubseteq \forall\ \text{manipulates (PhysicalEntity} \sqcup \text{PhysicalProperty)}\end{array}$$

monitors

The *monitors* object property is the inverse of the *affects* object property and is defined by the following DL notation:

$$\begin{array}{l}\text{monitors} \sqsubseteq \text{topObjectProperty}\\ \text{affects} \equiv \text{monitors}^{-}\\ \exists\ \text{monitors Thing} \sqsubseteq \text{Sensor}\\ \top \sqsubseteq \forall\ \text{monitors Actuator}\end{array}$$

plays

The *plays* object property relates a *Thing* to some *Role* at runtime to execute some workflow activity. It is defined by the following DL notation:

$\text{plays} \sqsubseteq \text{topObjectProperty}$
$\exists \text{ plays Thing} \sqsubseteq \text{Thing}$
$\top \sqsubseteq \forall \text{ plays Role}$

reads

The *reads* object property relates a *Sensor* to some *Tag* and is defined by the following DL notation:

$\text{reads} \sqsubseteq \text{topObjectProperty}$
$\exists \text{ reads Thing} \sqsubseteq \text{Sensor}$
$\top \sqsubseteq \forall \text{ reads Tag}$

senses

The *senses* object property relates a *Sensor* to some *PhysicalProperty* and is defined by the following DL notation:

$\text{senses} \sqsubseteq \text{topObjectProperty}$
$\exists \text{ senses Thing} \sqsubseteq \text{Sensor}$
$\top \sqsubseteq \forall \text{ senses PhysicalProperty}$

usesDevice

The *usesDevice* object property relates a *Capability* to some *Device* and is defined by the following DL notation:

$\text{usesDevice} \sqsubseteq \text{topObjectProperty}$
$\exists \text{ usesDevice Thing} \sqsubseteq \text{Capability}$
$\top \sqsubseteq \forall \text{ usesDevice Device}$

5.1.4 Ontology Data properties

In this ontology, only *PhysicalProperties* are represented by data, and only *Commands* include a service method URL as data. These data properties are defined by the following DL notation:

$\exists \text{ physicalPropertyData Datatype\#Literal} \sqsubseteq \text{PhysicalProperty}$
$\top \sqsubseteq \forall \text{ serviceMethodURL \#anyURI} \sqsubseteq \text{Command}$

5.1.5 DL-safe SWRL Rules

The following DL-safe SWRL rules are defined as part of the ontology's TBox as they provide some core reasoning for any application domain. However, in a discrete application scenario one has to specify additional rules depending on the modeled individuals. The additional rules are essential to express the situational contexts in which *canPlay* relations have to be created. Without these, the *plays* relations are not created by the Semantic Access Layer.

accessesDevice Inference

$$usesDevice(?x, ?y) \wedge hasCommand(?x, ?z) \rightarrow accessesDevice(?z, ?y)$$

affects Inference

$$manipulates(?x, ?y) \wedge senses(?z, ?y) \rightarrow affects(?x, ?z)$$

isAvailableIn Inference

$$isAttachedTo(?x, ?y) \wedge hasEntity(?z, ?y) \rightarrow isAvailableIn(?x, ?z)$$

isAllocatedTo Inference

$$plays(?x, ?y) \wedge isPlayedFor(?y, ?z) \rightarrow isAllocatedTo(?x, ?z)$$

canBePlayedFor Inference

$$canPlay(?x, ?y) \wedge isAttachedTo(?x, ?z) \rightarrow canBePlayedFor(?y, ?z)$$

$$canPlay(?x, ?y) \wedge isAvailableIn(?x, ?z) \rightarrow canBePlayedFor(?y, ?z)$$

isPlayedFor Inference

$$plays(?x, ?y) \wedge isAttachedTo(?x, ?z) \rightarrow isPlayedFor(?y, ?z)$$

$$plays(?x, ?y) \wedge isAvailableIn(?x, ?z) \rightarrow isPlayedFor(?y, ?z)$$

5.2 Discussion of Role Modeling Features

In the following, we briefly discuss how the relevant role modeling features (RMF) identified in Section 4.1.1 have been fulfilled by the created ontology.

Table 5.1: Fulfillment of the relevant role modeling features.

Nr.	Definition	Fulfillment
1	A role comes with its own properties and behaviour.	Roles are classes of the ontology with properties and behaviour.
2	Roles depend on relationships.	A role has to be played by a *Thing* to be meaningful.
3	An object may play different roles simultaneously.	Both the *canPlay* and the *plays* relations can assert a *Thing* to several *Roles*.
4	An object may play the same role several times, simultaneously.	A *Thing* can play a *Role* for several contexts or entities.
5	An object may acquire and abandon roles dynamically.	The *plays* relation is dynamically created and deleted at runtime.
7	Objects of unrelated types can play the same role.	Possible for specializations of the *Thing* class.
12	Roles restrict access.	Roles restrict the access to *Device Capabilities*
13	Different roles may share structure and behaviour.	It is allowed to create several identical *Roles* as distinct individuals.
14	An object and its roles share identity.	*Capabilities* are accessed through the role, therefore the identity is shared.
16	Relationships between roles can be constrained.	SWRL rules can constrain the creation of the *canPlay* relation for related *Roles*.
19	Roles depend on compartments.	Roles are related to a *PhysicalContext* by the *isAvailableIn* relation.
21	A role can be part of several compartments.	A *Role* can be played for several *PhysicalContexts*.
24	Compartments can contain other compartments.	Indirectly via the hierarchical structure of *PhysicalEntities*.

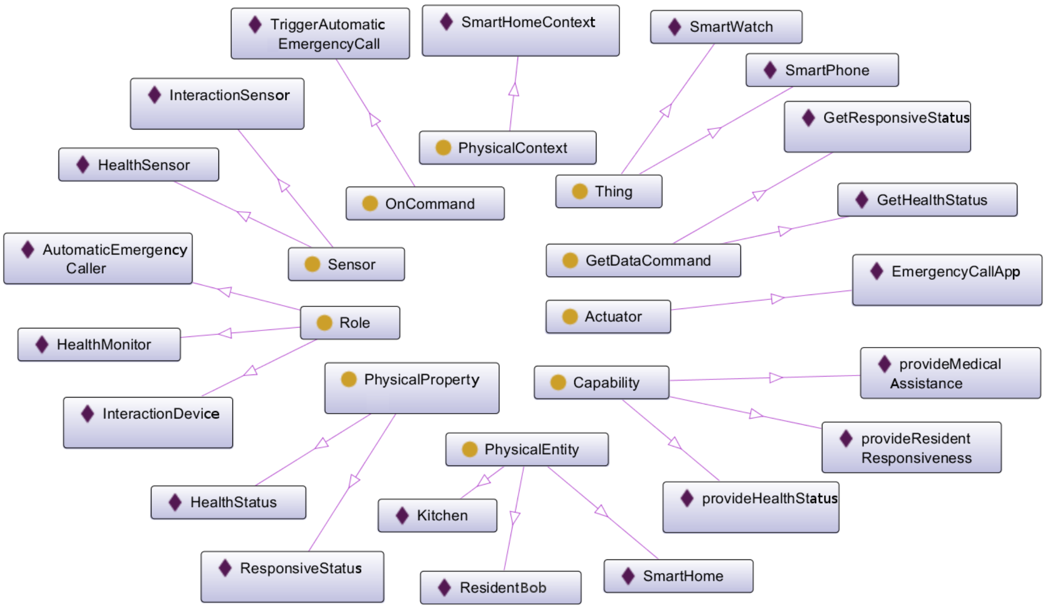


Figure 5.2: Individuals by classes for example application scenario.

5.3 Example Application Scenario Modeling

As the modeling of the ontology individuals, most SWRL rules, SPARQL queries, activity goals, and workflows are domain-specific and depend on the application scenario; we illustrate the respective modeling approaches for the application scenario introduced in Section 1.2. The implicit goal of the scenario workflow is to provide medical help to residents in a critical health condition within a smart home context. This is implemented in three activities. First, the health status of residents is monitored. In case of a critical health event, the next activity is executed and tries to get feedback from the resident. If the resident is considered as unresponsive, the final activity triggers an automatic emergency call. In the following, we provide the relevant models for this application scenario within the resource, user and workflow perspectives according to our contributions.

5.3.1 Resource Perspective

The resource perspective includes the modeling of the ontology ABox and the domain-specific SWRL rules.

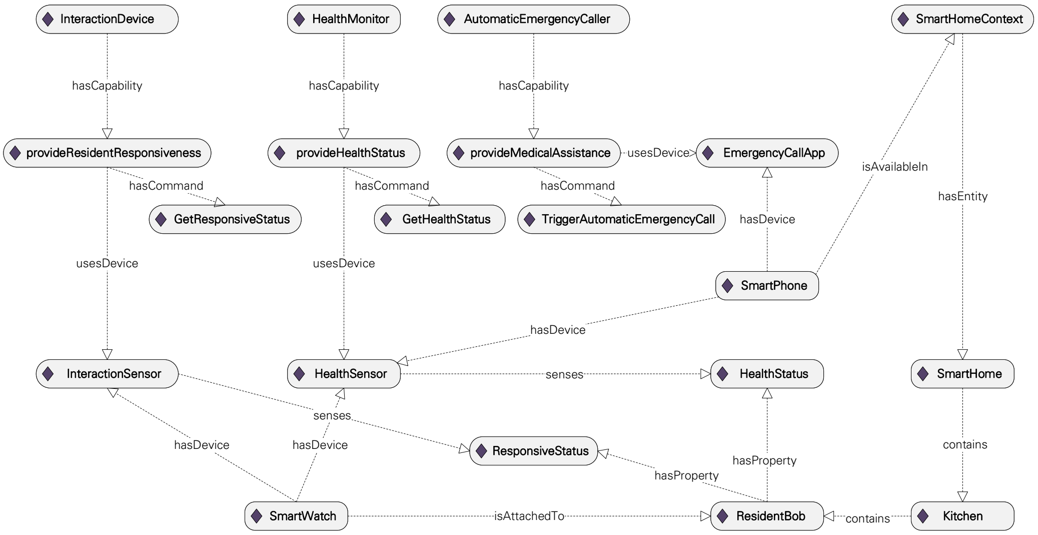


Figure 5.3: Relations of individuals in example application scenario.

ABox of the Application Scenario

Figure 5.2 gives an overview of the created individuals and their types while Figure 5.3 shows the object property assertions between the individuals [3]. We created a *SmartWatch* and a *SmartPhone* as individuals of the class *Thing*. These are the main devices for detecting the critical health status, interacting with the resident and making the emergency call. Consequently, these context-sensitive functionalities are modeled as the *Roles*: *AutomaticEmergencyCaller*, *HealthMonitor* and *InteractionDevice*. The available *Capabilites* are *provideMedicalAssistance*, *provideResidentResponsiveness* and *provideHealthStatus*. These *Capabilities* are related to the *Devices*: *EmergencyCallApp*, *HealthSensor*, and *InteractionSensor*. The *EmergencyCallApp* is an *Actuator* for making an emergency call via the *OnCommand TriggerAutomaticEmergencyCall*. The *HealthSensor* and *InteractionSensor* are *Sensors* for detecting the *PhysicalProperties HealthStatus* and *ResponsiveStatus* with the *GetDataCommands getHealthStatus* and *getResponsiveStatus*. The *PhysicalContext SmartHomeContext* includes the *PhysicalEntity SmartHome* via the *hasEntity* relation. In addition, the *SmartHome* contains the *PhysicalEntity*

---

[3] The complete ABox of the created OWL ontology for the application scenario can be found in the Appendix Section10.2.

Kitchen which in turn contains the *PhysicalEntity ResidentBob*. In this example, the workflow aims at monitoring Bob's health status within the smart home.

SWRL Rules of the Application Scenario

The following SWRL rules are necessary to create and update the *canPlay* object property assertions between *Things* and *Roles* at runtime. The creation and updating of the *canBePlayedFor* object property assertion is not necessary as it is already implemented in the SWRL rules of the ontology. However, we added the logic to the following rules for the sake of completion.

HealthMonitor Role

The creation and update of the *canPlay* relation for the *HealthMonitor Role* is modeled both for the *SmartWatch* and the *SmartPhone*. They both provide the necessary sensors to detect the *HealthStatus* of *ResidentBob*. However, in the case of the smartwatch the *canPlay* relation is only created when the smartwatch is currently attached to Bob, i. e., is located at her wrist:

$$\begin{gathered} isAttachedTo(SmartWatch, ResidentBob) \wedge \\ hasDevice(SmartWatch, HealthSensor) \wedge \\ usesDevice(provideHealthStatus, HealthSensor) \wedge \\ hasCapability(HealthMonitor, provideHealthStatus) \rightarrow \\ canPlay(SmartWatch, HealthMonitor) \wedge \\ canBePlayedFor(HealthMonitor, ResidentBob) \end{gathered}$$

In case of the smartphone, the *canPlay* relation is only created when the smartphone is currently available in the *SmartHomeContext* and Bob is related to the context via the *hasEntity* and the transitive *contains* relation:

$$\begin{gathered} isAvailableIn(SmartPhone, SmartHomeContext) \wedge \\ hasEntity(SmartHomeContext, ?x) \wedge contains(?x, ResidentBob) \wedge \\ hasDevice(SmartPhone, HealthSensor) \wedge \\ usesDevice(provideHealthStatus, HealthSensor) \wedge \\ hasCapability(HealthMonitor, provideHealthStatus) \rightarrow \\ canPlay(SmartPhone, HealthMonitor) \wedge \\ canBePlayedFor(HealthMonitor, ResidentBob) \end{gathered}$$

AutomaticEmergencyCaller Role

The creation and update of the *canPlay* relation for the *AutomaticEmergencyCaller Role* is modeled for the *SmartPhone*. It provides the necessary *Actuator EmergencyCallApp* to place an emergency call. However, the *canPlay* relation is only created when the smartphone is currently available in the *SmartHomeContext*:

$$\begin{gathered}
isAvailableIn(SmartPhone, SmartHomeContext) \wedge \\
hasDevice(SmartPhone, EmergencyCallApp) \wedge \\
usesDevice(provideMedicalAssistance, EmergencyCallApp) \wedge \\
hasCapability(AutomaticEmergencyCaller, provideMedicalAssistance) \rightarrow \\
canPlay(SmartPhone, AutomaticEmergencyCaller) \wedge \\
canBePlayedFor(AutomaticEmergencyCaller, SmartHomeContext)
\end{gathered}$$

InteractionDevice Role

The creation and update of the *canPlay* relation for the *InteractionDevice Role* is modeled for the *SmartWatch*. It provides the necessary *Sensor InteractionSensor* to sense the *HealthStatus* of *ResidentBob*. However, the *canPlay* relation is only created when the smartwatch is currently attached to Bob, i. e., is located at her wrist:

$$\begin{gathered}
isAttachedTo(SmartWatch, ResidentBob) \wedge \\
hasDevice(SmartWatch, InteractionSensor) \wedge \\
usesDevice(provideResidentResponsiveness, InteractionSensor) \wedge \\
hasCapability(InteractionDevice, provideResidentResponsiveness) \rightarrow \\
canPlay(SmartWatch, InteractionDevice) \wedge \\
canBePlayedFor(InteractionDevice, ResidentBob)
\end{gathered}$$

5.3.2 User Perspective

Within the user perspective, the goal model and the mapping descriptions are modeled for the application scenario.

Modeling User Goals

Figure 5.4 shows the goal model for the application scenario. It includes the goals **Get Health Data from SmartWatch**, **Get Health Data from SmartPhone**, **Get Interaction via SmartWatch**, and **Use Emergency Call App** as *leafs*. These are executable in the sense that there exists a mapping to semantic queries.

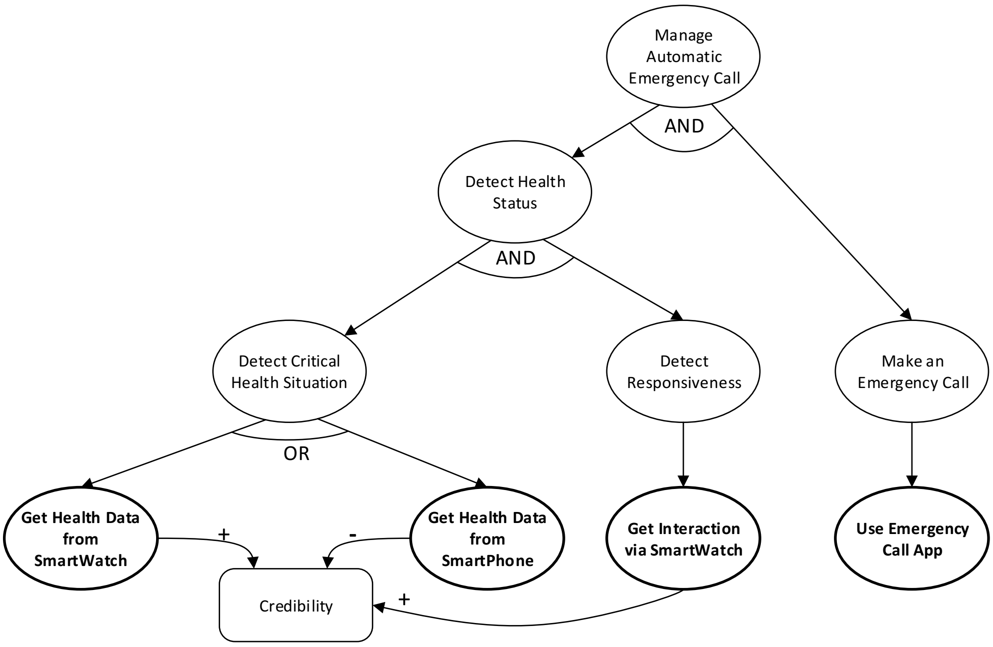

Manage Automatic Emergency Call
AND
Detect Health Status
AND
Detect Critical Health Situation
Detect Responsiveness
Make an Emergency Call
OR
Get Health Data from SmartWatch
Get Health Data from SmartPhone
Get Interaction via SmartWatch
Use Emergency Call App
+
-
Credibility
+

Figure 5.4: Goal model for example application scenario.

On the next level there exist the goals **Detect Critical Health Situation**, **Detect Responsiveness** and **Make Emergency Call**. The upper goals are very broad and will not be used for the workflow modeling. However, there existence is necessary to allow for the backward reasoning of leaf goals. The quality parameter **Credibility** expresses the level of credibility of the sensed data. If the health status is sensed by a smartwatch, it is assumed that the smartwatch is located at the wrist of the resident and therefore the credibility of the produced data is considered as high for detecting a critical health situation. This assumption is based on the modeled SWRL rules for creating the *canPlay* relation between the *SmartWatch* and the *HealthMonitor Role*. In contrast, if the data is generated by the smartphone, it is only guaranteed that the smartphone shares the same context as the resident. Therefore the credibility of the sensed health status is considered as low. In addition, the provided interaction via the smartwatch is considered a credible data source for detecting the responsiveness of a resident.

Mapping the Goals to SPARQL Queries

Listing 5.1 specifies the mapping of the goal **Get Health Data from SmartWatch** to the corresponding query. Here, capabilities of the *iot:HealthMonitor* role are selected which are able to retrieve data via an *iot:GetDataCommand*. Also, the *iot:SmartWatch* has to be able to play the *iot:HealthMonitor* role as defined by the goal. Listing 5.2 defines the mapping with a restriction to the *iot:SmartPhone*.

```
Get Health Data from SmartWatch ->
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX iot: <http://www.semanticweb.org/IoTUDresden#>
SELECT ?cap ?entity
 WHERE {
 ?entity rdf:type iot:PhysicalEntity .
 iot:SmartWatch iot:canPlay iot:HealthMonitor .
 iot:HealthMonitor iot:canBePlayedFor ?entity .
 iot:HealthMonitor iot:hasCapability ?cap .
 ?cap iot:hasCommand ?cmd .
 ?cmd rdf:type ?cmdType .
 ?cmdType rdfs:subClassOf iot:GetDataCommand .
}
```

Listing 5.1: Mapping for goal *Get Health Data from SmartWatch*

```
Get Health Data from SmartPhone ->
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX iot: <http://www.semanticweb.org/IoTUDresden#>
SELECT ?cap ?entity
 WHERE {
 ?entity rdf:type iot:PhysicalEntity .
 iot:SmartPhone iot:canPlay iot:HealthMonitor .
 iot:HealthMonitor iot:canBePlayedFor ?entity .
 iot:HealthMonitor iot:hasCapability ?cap .
 ?cap iot:hasCommand ?cmd .
 ?cmd rdf:type ?cmdType .
 ?cmdType rdfs:subClassOf iot:GetDataCommand .
}
```

Listing 5.2: Mapping for goal *Get Health Data from SmartPhone*

Listing 5.3 specifies the mapping of the goal **Get Interaction via SmartWatch** to the corresponding query. Here, capabilities of the *iot:InteractionDevice* role are selected which are able to retrieve data via an *iot:GetDataCommand*. Also, the *iot:SmartWatch* has to be able to play the *iot:InteractionDevice* role as defined by the goal. Listing 5.4 defines the mapping of the goal **Use Emergency Call App** to place an automatic emergency call via the *iot:OnCommand* of the *iot:AutomaticEmergencyCaller* role.

```
Get Interaction via SmartWatch ->
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX iot: <http://www.semanticweb.org/IoTUDresden#>
SELECT ?cap ?entity
 WHERE {
 ?entity rdf:type iot:PhysicalEntity .
 iot:SmartWatch iot:canPlay iot:InteractionDevice .
 iot:InteractionDevice iot:canBePlayedFor ?entity .
 iot:InteractionDevice iot:hasCapability ?cap .
 ?cap iot:hasCommand ?cmd .
 ?cmd rdf:type ?cmdType .
 ?cmdType rdfs:subClassOf iot:GetDataCommand .
}
```

Listing 5.3: Mapping for goal *Get Interaction via SmartWatch*

```
Use Emergency Call App ->
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX iot: <http://www.semanticweb.org/IoTUDresden#>
SELECT ?cap ?entity
 WHERE {
 ?entity rdf:type iot:PhysicalContext .
 iot:SmartPhone iot:canPlay iot:AutomaticEmergencyCaller .
 iot:AutomaticEmergencyCaller iot:canBePlayedFor ?entity .
 iot:AutomaticEmergencyCaller iot:hasCapability ?cap .
 ?cap iot:hasCommand ?cmd .
 ?cmd rdf:type ?cmdType .
 ?cmdType rdfs:subClassOf iot:OnCommand .
}
```

Listing 5.4: Mapping for goal *Use Emergency Call App*

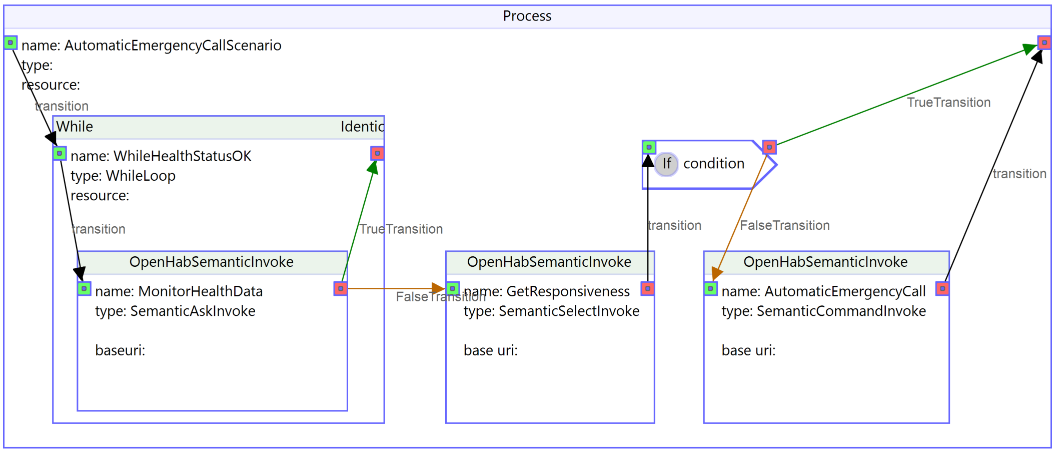


Figure 5.5: Semantic Query-based Workflow Model for Application Scenario.

5.3.3 Workflow Perspective

Modeling in the workflow perspective includes the workflow model itself which is either based on activities specified by semantic queries or goals from the goal model.

Semantic Query-based Workflow Model

Figure 5.5 shows the workflow model for the application scenario using semantic queries to specify the implicit goals of activities. The workflow model was created with the help of an extended process modeling IDE for CPS processes [SKNS14, HSS16, HSKS16]. The first activity of type *SemanticAskInvoke* is modeled inside of a while loop for continuously monitoring the health state of the smart home residents. As this activity is of type *SemanticAskInvoke*, it is directly evaluated to a boolean value depending on the semantic query. The corresponding semantic query is defined in Listing 5.5. If the query is evaluated to the boolean value *false*, the subsequent activity of type *SemanticSelectInvoke* is triggered. Here, the responsiveness of the resident is checked by requesting a user interaction. The corresponding semantic query defined in Listing 5.6. If the resident is responsive, there is no need for an automatic emergency call and the workflow is completed. Otherwise, the final activity of type *SemanticCommandInvoke* is triggered to place an emergency call via a smartphone app.

Listing 5.5 defines the SPARQL query for monitoring the health data of a resident. Here, the *Role iot:HealthMonitor* is searched which can be played for an entity, i. e., a resident and by the an *iot:Thing*. The *iot:HealthMonitor* role has to provide a *Capability* for retrieving data of a *PhysicalProperty* of the resident. Therefore, the retrieved data represents the health state of the resident. The value is filtered to be larger than zero. The value is considered to express the severity of a health problem and is defined as a xsd:float between [0..100]. If the query produces results, at least one of the residents may have a critical health condition.

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX iot: <http://www.semanticweb.org/IoTUDresden#>
ASK
WHERE {
?entity rdf:type iot:PhysicalEntity .
?thing rdf:type iot:Thing .
?thing iot:canPlay iot:HealthMonitor .
iot:HealthMonitor iot:canBePlayedFor ?entity .
iot:HealthMonitor iot:hasCapability ?cap .
?cap iot:hasCommand ?cmd .
?cmd rdf:type ?cmdType .
?cmdType rdfs:subClassOf iot:GetDataCommand .
?cmd iot:accessesDevice ?device .
?device iot:senses ?physProp .
?physProp iot:physicalPropertyData ?currentValue .
FILTER(?currentValue > 0.0)
}
```

Listing 5.5: Query for *MonitorHealthData* SemanticAskInvoke.

Listing 5.6 defines the SPARQL query for monitoring the responsiveness of a resident. Here, the boolean *physicalPropertyData* representing the responsiveness of a resident is retrieved. The data is generated by the execution of a *GetDataCommand* within the role *iot:InteractionDevice*. The role has to be playable for an *iot:PhysicalEntity*, i. e., a resident. Furthermore, the role *iot:InteractionDevice* has to be playable by an *iot:Thing*. As the query type is specified as *SemanticSelectInvoke*, the data is retrieved and made available at a data output port of the respective workflow activity.

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX iot: <http://www.semanticweb.org/IoTUDresden#>
SELECT ?responsive
WHERE {
?entity rdf:type iot:PhysicalEntity .
?thing rdf:type iot:Thing .
?thing iot:canPlay iot:InteractionDevice .
iot:InteractionDevice iot:canBePlayedFor ?entity .
iot:InteractionDevice iot:hasCapability ?cap .
?cap iot:hasCommand ?cmd .
?cmd rdf:type ?cmdType .
?cmdType rdfs:subClassOf iot:GetDataCommand .
?cmd iot:accessesDevice ?device .
?device iot:senses ?physProp .
?physProp iot:physicalPropertyData ?responsive .
}
```

Listing 5.6: Query for *GetResponsiveness* SemanticSelectInvoke.

Listing 5.7 defines the SPARQL query for triggering an *AutomaticEmergencyCall*. Here, the playable role *AutomaticEmergencyCaller* with a corresponding *Capability* and *OnCommand* is searched.

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX iot: <http://www.semanticweb.org/IoTUDresden#>
SELECT ?cmd ?thing
WHERE {
?entity rdf:type iot:PhysicalContext .
?thing iot:canPlay iot:AutomaticEmergencyCaller .
iot:AutomaticEmergencyCaller iot:canBePlayedFor ?entity .
iot:AutomaticEmergencyCaller iot:hasCapability ?cap .
?cap iot:hasCommand ?cmd .
?cmd rdf:type ?cmdType .
?cmdType rdfs:subClassOf iot:OnCommand .
}
```

Listing 5.7: Query for AutomaticEmergencyCall SemanticCommandInvoke.

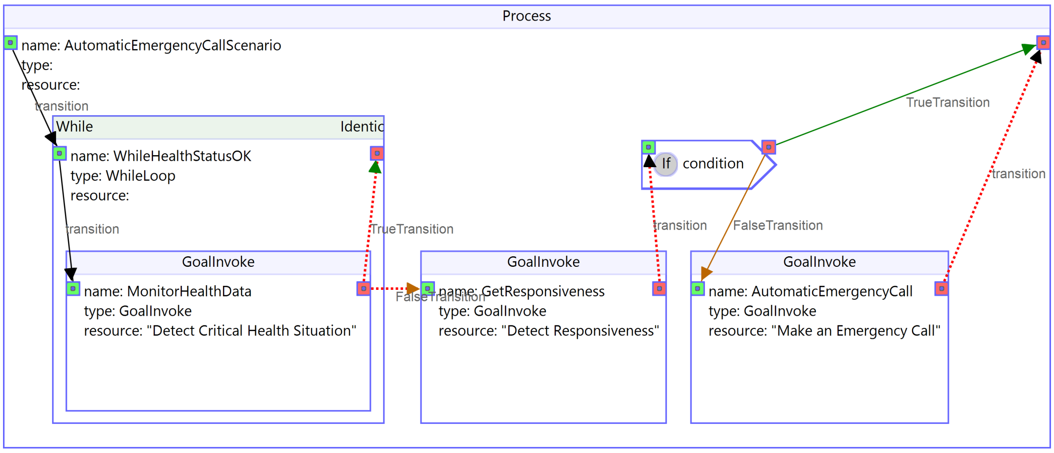


Figure 5.6: Goal-based Workflow Model for Application Scenario.

Goal-based Workflow Model

In contrast to the semantic query-based workflow model, the goal-based workflow model only includes activities of the type *GoalInvoke*. These specify their respective activity goals as Strings matching the names of the goals from the goal model. The first activity defines the goal **Detect Critical Health Situation**. This goal is evaluated at runtime, according to the available *Things* and playable *Roles*. Within the goal description of the activity, we also specified to consider the quality parameter **Credibility**. Therefore, the backward reasoning inside the goal model determines the leaf goal **Get Health Data from SmartWatch** to fulfill the upper goal of the activity. Then, the mapped SPARQL query is available for execution. The subsequent goals are defined respectively for detecting the responsiveness of the resident and for placing an emergency call. While reducing the expressiveness of the directly modeled semantic queries, the goal-based workflow model simplifies the workflow modeling process and allows for the reusability of activity goals. Furthermore, in a real-world scenario all of the activity types can be used within one workflow model to provide any degree of abstraction. In case of the goal modeling, a domain expert is necessary to provide both, the goal model itself and the mappings to semantic queries based on the domain-specific ABox of the created ontology.

5.4 Summary

In this chapter, we presented the models for adaptive workflow activities in the IoT. This includes the TBox of a novel ontology for the modeling of role-based *Things* with context-sensitive capabilities, which is designed to be applied in any IoT application scenario. The ontology represents the major contribution of this work to the resource perspective of IoT-aware workflow management. In addition, parts of the modeling within the resource perspective, the user perspective and the workflow perspective are domain-specific. Therefore, we revisited the application scenario introduced in Section 1.2 and provided the respective models for each perspective to enable context-sensitive activity-level adaptation of the scenario workflow.

6 Architecture for Adaptive Workflow Activities in the IoT

In this chapter[1], we introduce the overall system architecture and its main components for adaptive workflow activities in the IoT. This includes the Semantic Access Layer (SAL) as the major contribution to the workflow perspective of IoT-aware workflow management. The SAL serves as a middleware for the context-sensitive discovery and invocation of *Things* as workflow resources.

6.1 Overview of the System Architecture

The proposed system architecture for adaptive workflow activities in the IoT is structured into three-tier as depicted in Figure 6.1. The tiers are: 1) the workflow execution engine, 2) the *Semantic Access Layer* (SAL) and 3) IoT services existing in the cyber world. However, IoT services enable the integration of the physical world, too. Each *Thing* existing in the physical world provides sensing or actuating capabilities. The capabilities of these *Things* are accessed via IoT services as proposed in [BBDL$^+$13]. As the IoT services are considered as a given, we only describe the workflow execution engine and the Semantic Access Layer (SAL).

Workflow Execution Engine

The workflow engine executes model-based workflows that include specialized types of activities extended to include either goals or SPARQL queries. SPARQL queries are interpreted directly to discovery fitting *Things* considering the runtime context and capability constraints provided by the *Role* concept. In addition, the workflow modeler can use high-level goals to define the intention of activities. These goals are then translated into domain-specific SPARQL queries via a mapping description. A workflow modeler can use the goals originating from a Tropos goal model including quality parameters.

---

[1] This chapter is partially based on [HSK$^+$16], [HSS16], [HSKS16], [SHS15] and [SHS16]

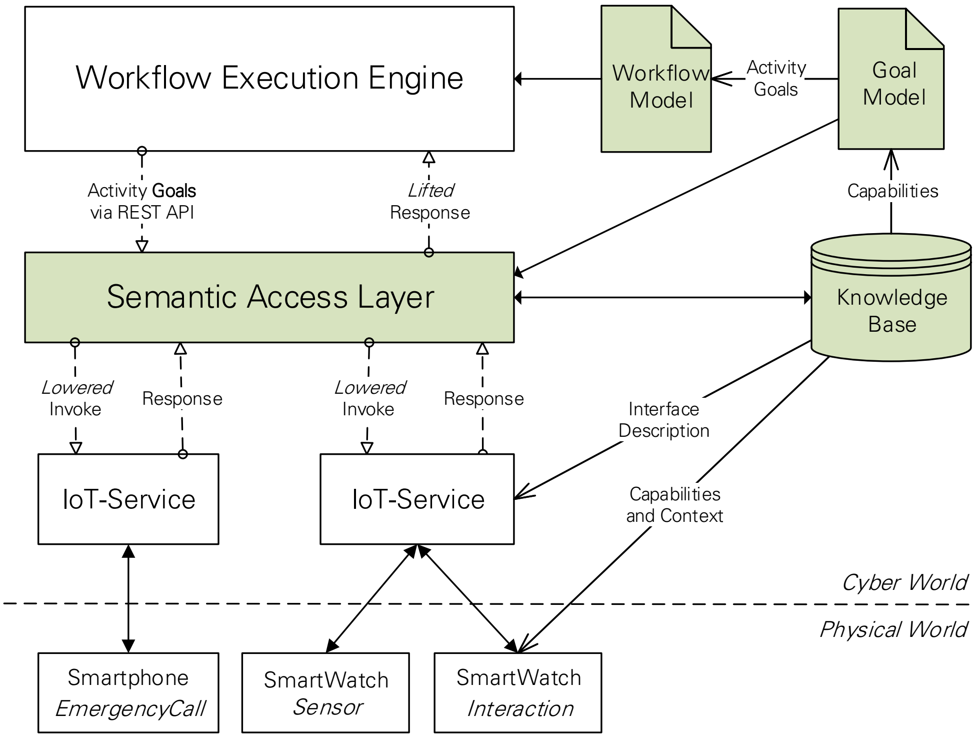


Figure 6.1: System Architecture for Adaptive Workflow Activities in the IoT.

Semantic Access Layer

The SAL has access to a knowledge base, which contains the proposed ontology for the role-based modeling of context-sensitive *Things* as described in Section 5.1.1. In addition, the knowledge base includes the domain-specific ontology ABox describing *Things*, *Roles*, their capabilities and context properties. The SAL is responsible for querying the knowledge base according to the SPARQL queries, which either were translated from activity goals or are directly included within the workflow activities. *Backward reasoning* is used by the SAL to determine which leaf goals need to be satisfied to reach the specified goal of the workflow activity as described in Section 2.2.2. Based on these leaf goals, the SAL then translated a domain specific SPARQL query using a *mapping description*. After executing a SPARQL query, the SAL transforms the result into an IoT service call based on the found *Capability* and the desired *Command*. The latter includes includes a reference to an IoT service method. The result of the service method invocation is fed back to the workflow engine. In this manner, the SAL implements the *Resource Discovery*, *Resource Estimation* and *Resource Allocation* phases of the *Resource Management* perspective within the IoT. Furthermore, the *Resource Monitoring* phase is implemented by updating the ontology through the execution of SWRL rules and resolving the possible inconsistencies.

Example Information Flow

In the following, we present an exemplary information flow: When a goal-based activity is executed by the process engine, the included goal and quality constraints are sent to the SAL via a REST service. The SAL uses the approach of *backward reasoning* [GMNS03] from the goal modeling research domain to select a set of leaf goals from the Tropos goal model. These leaf goals have been modeled to represent capabilities of *Things*. Using a *mapping description*, the leaf goals are translated into a SPARQL query. Subsequently, the SAL executes the SPARQL query on the knowledge base. The query result may contain a set of available *Role Capabilities* with an associated *Command* that fulfills the properties of the query. In this case, the IoT service method specified by the *Command* is invoked. Depending on the type of *Command*, either sensor data is retrieved or an actuating command is executed. Finally, the IoT service response is semantically annotated by the SAL and forwarded to the workflow engine. These transformation steps are called *lowering* and *lifting* [KNM10].

6.2 Specification of System Components

In the following, we describe the relevant components of the system architecture depicted in Figure 6.1. These are the *Knowledge Base* as part of the resource perspective, the *Goal* and *Workflow Model* in the context of the user perspective and the *Semantic Access Layer* as part of the workflow perspective. Even though the workflow execution engine is not a contribution in the context of this work, we shortly introduce the Process Execution System for Cyber-Physical Systems (PROtEUS) [SHS15] for executing the goal-based and semantic query-based workflows.

6.2.1 Resource Perspective

From an architecture point of view, the resource perspective includes the knowledge base for managing the ontological model. The knowledge base includes the TBox of the created ontology for modeling role-based *Things* in the IoT. In addition, the domain-specific ABox of the ontology is managed within the knowledge base to represent and update the physical and the cyber state of *PhysicalEntities*, *Things* and their playable *Roles*.

6.2.2 User Perspective

The user perspective includes the goal model and the mapping description. The modeling of the user perspective is introduced in Section 5.3.2, here we discuss their integration in the overall system architecture. The goal model includes all the high-level user goals within a specific application scenario as well as their mapping descriptions. As shown in Figure 6.1, the goal model uses the ontological model for the mapping descriptions and provides the goals for the modeling of workflow activities. With respect to the SAL, the goal model serves for the runtime reasoning from high-level goals to leaf goals. And for their mapping to SPARQL queries.

6.2.3 Workflow Perspective

From an architectural point of view, the workflow perspective includes the workflow engine, model and the Semantic Access Layer. The latter is the main contribution of this work within the field of IoT-aware workflow management.

Workflow Model

The workflow model either consists of goal-based or semantic query-based activities. It is executed by the workflow engine and not exposed to other system components. However, the execution of goal-based and semantic query-based activities invokes a service method call to the SAL for performing the resource discovery, estimation, allocation, and monitoring.

Workflow Engine

The process execution system for cyber-physical systems PROtEUS consists of several components designed to meet the specific challenges of CPS [SHS16]. It has

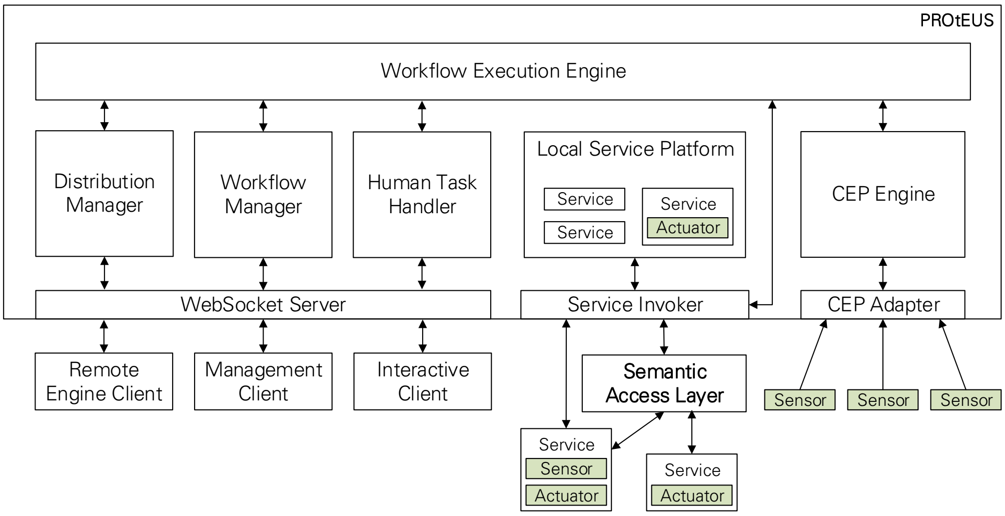


Figure 6.2: Overview of the PROtEUS workflow engine components.

been designed in alignment with the reference architecture for workflow management systems proposed by the WfMC [GdV98, SHS16]. The core of PROtEUS consists of a generic workflow meta-model describing the structure of workflow models and a workflow engine responsible for instantiating these models and executing actual workflow instances [SKNS14, SHS16]. The meta-model is designed with regard to the properties of CPS [SKNS14]. In general, it follows a component-oriented view of workflow elements. Workflows and activities can be composed hierarchically and ports describe their input and output with regard to control and typed data flow. Transitions between these ports allow for the modeling of concrete data and control flow between activities. Access to the workflow control functionality is achieved by the workflow manager. A complex event processing (CEP) engine is able to process large amounts of external sensor data and trigger high-level events within special activity types [SHS16]. PROtEUS also offers a service platform for deploying and calling internal services, e. g., Open Services Gateway initiative (OSGi) services, and an external service invoker. Human interactions are enabled by a human task handler sending interaction requests to interactive clients [SHS16]. The bi-directional communication with the workflow execution system from remote clients for monitoring, interaction and control purposes is enabled by a WebSocket server [SHS16]. The overall system architecture is illustrated in Figure 6.2. The interfaces between

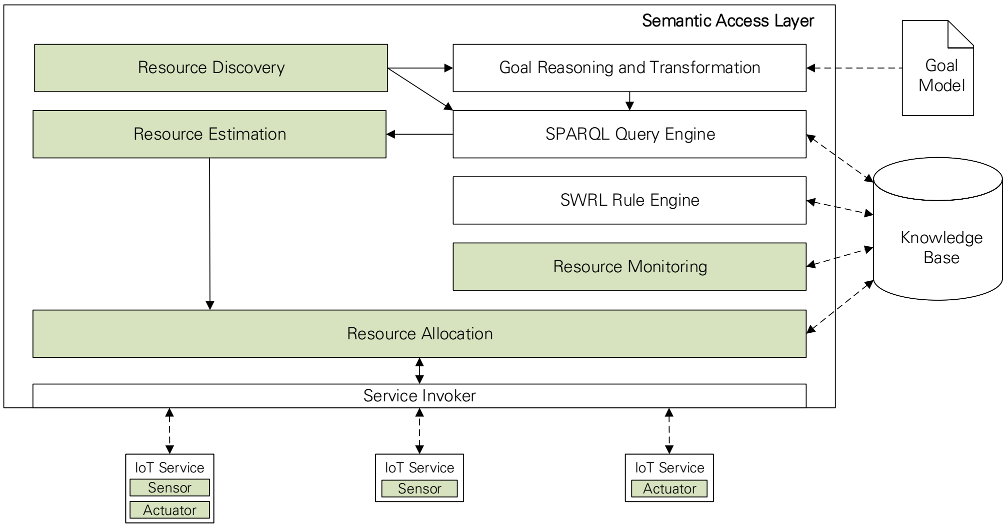


Figure 6.3: System Components of the Semantic Access Layer

the cyber world and the physical world are represented by the software controlled sensors and actuators of the CPS [SHS16].

At runtime, the engine instantiates a workflow model and executed activities according to the modeled control and data flow. Through subtype polymorphism, the workflow engine is able to call a specific execution method according to the type of the scheduled activity for execution. There exist special activity types for control flow logic, e. g., splits and loops, data flow logic and for the invocation of various types of services. Through specializations of an atomic or a composite activity type, the meta-model can easily be extended to support a wide range of workflow elements.

Semantic Access Layer Middleware

Figure 6.3 gives an overview of the SAL's system components. Conceptually, the SAL serves as a middleware mediating between the workflow engine and various IoT services. The SAL provides a REST service interface to receive *SemanticInvoke* queries and *GoalBasedInvoke* instances from a client, e. g., workflow execution system. Additionally, the SAL integrates a knowledge base for domain-specific ontologies that describe role-based *Things* in the IoT. At runtime, the ABox of the knowledge base includes all available *Things*, *Roles*, *PhysicalEntities* and their relations as well as context data. Among others, context information includes the absolute or relative location of a *Thing*. In the following, we introduce each component in a chronological order given a service call from the PROtEUS workflow engine for a goal-based activity.

Resource Discovery: At first, the resource discovery component receives the service call from the workflow engine. The request is evaluated for the inclusion of a SPARQL query or a goal. In the latter case, the goal is transferred to the Goal Reasoning and Transformation component.

Goal Reasoning and Transformation: In the event of a received *GoalBasedInvoke* instance, the SAL uses the goal model and the mapping description to generate a semantic query. As described in Section 4.2.2, the first step in generating a semantic query is the selection of a set of leaf goals contributing to the satisfaction of the user-defined activity goals. The SAL uses *backward reasoning*, cf. Section 2.2.2, to select the desired leaf goals in polynomial time. In case the workflow modeler provided quality parameters within the *GoalBasedInvoke* activity, these are included in the reasoning process to find the best fit set of leaf goals which satisfy the activity goal as well as contribute to the specified quality parameter. With the help of the mapping descriptions, the leaf goals are then substituted to a set of SPARQL queries. Subsequently, the semantic query is transferred to the SPARQL Query Engine.

SPARQL Query Engine: Upon receiving or generating a semantic query, the query engine executes the SPARQL query on the knowledge base. The query result is then transferred to the resource estimation component.

Resource Estimation: The resource estimation component defined an order of priority over the query results. As the query result does only include fitting resources in terms of an available *canPlay* relation between a *Thing* and a *Role* as well as the optional fulfillment of quality parameters, the resource estimation is based on a simple load balancing strategy. This counts the *plays* relations of the *Things* in the query result and orders them accordingly. Then, the orderd query result is transferred to the resource allocation component.

Resource Allocation: The resource allocation component is responsible for creating the *plays* relation between a *Thing* and a *Role* for representing the state of runtime allocation of *Things* as workflow resources. In general, the allocation of a resource is considered to be active until the workflow instance is completed, i. e., the control flow reached the end control flow port of the modeled workflow. Once the *plays* relation is created, the referred service method of the related *Command* is invoked to either gather sensor data or trigger an actuator. Even though, the knowledge base includes sensor data in the *physicalDataProperty* data property assertions, they are only eventually consistent as the resource monitoring component updates the data properties following a lazy fetch strategy. Therefore, in case of a *GetDataCommand* sensors also have to be allocated and the respective service method invoked to gather the data at runtime. This data is then fed back to the workflow engine, where it becomes available as a data object within the corresponding workflow instance. In case of a *OnCommand* or *OffCommand*, the provided service methods are invoked.

Resource Monitoring: The resource monitoring component has several functionalities. At first, it removes the created *plays* relation axioms of allocated resource when the corresponding workflow instance is finished or fails. Furthermore, all *GetDataCommands* of the ontology ABox are continuously executed to update the associated data properties of the *PhysicalEntities*. However, this update mechanism is following a lazy fetch strategy to mitigate the processing load of the knowledge base. The lazy fetch updates the *physicalDataProperties* whenever axioms are created either by the SWRL rule engine, or the resource allocation component. However, the resource monitoring can also choose a continuous fetch strategy where sensor data is retrieved and updated according to a discrete time interval. The eventual consitency of the *physicalDataProperties* is necessary for enabling SPARQL queries to use the restriction keyword *FILTER*. These are extensively used in the *SemanticAskInvoke* activities to filter query results according to some data threshold. In addition, the resource monitoring includes the runtime check for inconsistencies regarding the *canPlay* and *plays* as well as the *canBePlayedFor* and *isPlayedFor* relations. If such inconsistencies occur, the respective *plays* and *isPlayedFor* relations are removed for no longer available workflow resources.

SWRL Rule Engine: The SWRL rule engine synchronizes the knowledge base whenever there is an axiom added, updated or removed from the knowledge base. For the ontology to represent the runtime state of the physical world, either an external context service is required or extensive data-driven rules have to be modeled and the resource monitoring needs to use the continuous fetch strategy.

6.3 Summary

In this chapter, we introduced the overall system architecture and its main components for adaptive workflow activities in the IoT. We presented the functionality and general flow of data between all system components. Furthermore, we introduced the architeture of the Semantic Access Layer (SAL) as the contribution IoT-aware workflow management. The SAL is a middleware for the context-sensitive discovery and invocation of IoT service based on workflow activity goals or semantic queries. Withing our ontology, the service methods are included as part of the *Command* concept.

7 Implementation of Adaptive Workflow Activities in the IoT

In this chapter [1], we present the implementation of our proof-of-concept prototype of the Semantic Access Layer (SAL) as plugins to the OpenHAB platform. Furthermore, we briefly introduce some technical aspects of the related PROtEUS WfMS for modeling and executing IoT-aware workflows based on our extended workflow metamodel.

7.1 Resource Perspective

The resource perspective includes the knowledge base for accessing the ontological model and associated data. We use the OWL API [2] version 3.4.3 and the HermiT OWL Reasoner [3] in version 1.3.8 within our work to infer the context changes from SWRL rules and other object property assertions.

7.2 Workflow Perspective

Within the workflow perspective, we describe the PROtEUS WfMS as the execution engine for our IoT-aware workflows and the Semantic Access Layer (SAL) as the middleware for context-sensitive resource discovery and invocation.

7.2.1 PROtEUS

PROtEUS is implemented as a Java-based prototype and is currently employed within a Smart Home Lab [SHS16]. As the Eclipse Modeling Framework (EMF) [SBMP08] provides a large set of tools for creating metamodels, models, and implementations in an automated way, we based the workflow metamodel and implementation of the engine on Ecore [SHS16]. An integrated modeling environment

---

[1] This chapter is partially based on [HSK⁺16, HSS16, HSKS16, SHS16]

[2] The OWL Api is available at http://owlapi.sourceforge.net/.

[3] The HermiT OWL Reasoner is available at http://www.hermit-reasoner.com/.

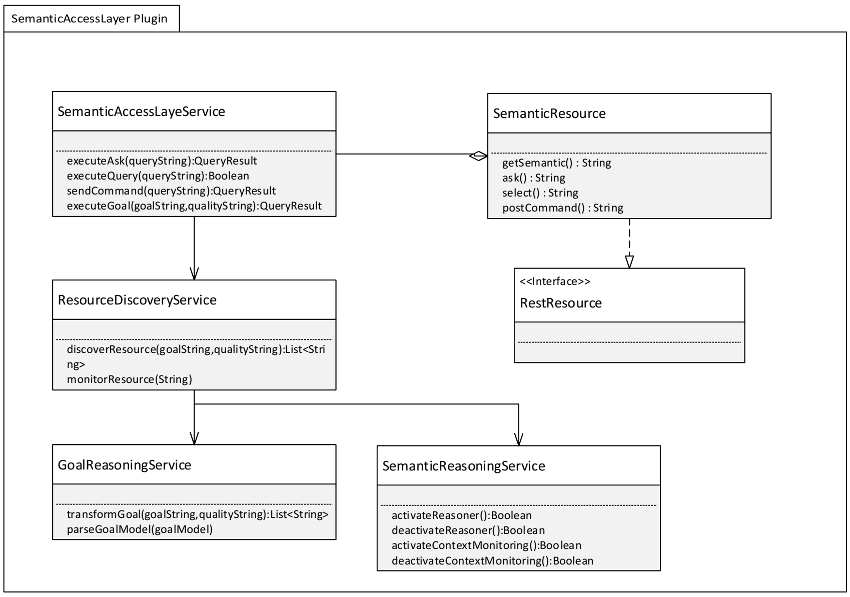
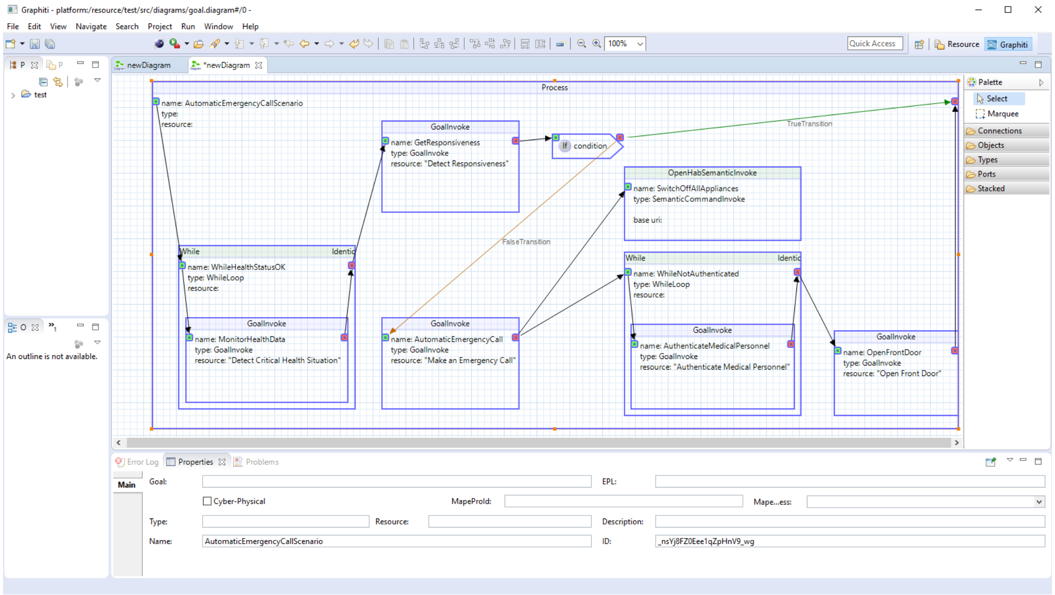

SemanticAccessLayer Plugin
SemanticAccessLayeService
executeAsk(queryString):QueryResult
executeQuery(queryString):Boolean
sendCommand(queryString):QueryResult
executeGoal(goalString,qualityString):QueryResult
SemanticResource
getSemantic() : String
ask() : String
select() : String
postCommand() : String
<<Interface>>
RestResource
ResourceDiscoveryService
discoverResource(goalString,qualityString):List<String>
monitorResource(String)
GoalReasoningService
transformGoal(goalString,qualityString):List<String>
parseGoalModel(goalModel)
SemanticReasoningService
activateReasoner():Boolean
deactivateReasoner():Boolean
activateContextMonitoring():Boolean
deactivateContextMonitoring():Boolean

Figure 7.1: Components of the Semantic Access Layer Core

realized as an Eclipse Plugin based on Graphiti [BGK^{+}11] supports the workflow designer with defining workflows for PROtEUS [SKNS14]. The event processing engine Esper [BV07] is used for implementing the CEP engine component [SHS16]. The model-based Smart Home middleware Open-HAB [SZZ14] is able to collect and unify sensor data from various sensors [SHS16]. It serves as the main source of event data for the CEP engine in PROtEUS and is used in combination with the ontology for role-based *Things* in the IoT by the SAL to find Smart Home services [HSS16]. PROtEUSs service platform is based on the OSGi component platform [All03]. Using protocol specific adapters, the service invoker is able to call OSGi services running on the local platform or services on remote servers [SHS16]. We implemented adapters for calling SOAP, REST and XMLRPC services, local Java classes, to invoke robot services on ROS [QCG^{+}09], and to call services on the OpenHAB middleware [SHS16]. To realize the WebSocket server, we use an implementation of the Web Application Messaging Protocol (WAMP) [FM11]. Clients supporting the processing of human tasks and monitoring of workflows are implemented on Android and tabletop devices [SSMS14].


Graphiti - platform:/resource/test/src/diagrams/goal.diagram#/0 -
File Edit View Navigate Search Project Run Window Help
100%
Quick Access
Resource
Graphiti
newDiagram
*newDiagram
test
An outline is not available.
Process
name: AutomaticEmergencyCallScenario
type:
resource:
GoalInvoke
name: GetResponsiveness
type: GoalInvoke
resource: "Detect Responsiveness"
If condition
TrueTransition
FalseTransition
OpenHabSemanticInvoke
name: SwitchOffAllAppliances
type: SemanticCommandInvoke
base uri:
While
Identic
name: WhileHealthStatusOK
type: WhileLoop
resource:
GoalInvoke
name: MonitorHealthData
type: GoalInvoke
resource: "Detect Critical Health Situation"
GoalInvoke
name: AutomaticEmergencyCall
type: GoalInvoke
resource: "Make an Emergency Call"
While
Identic
name: WhileNotAuthenticated
type: WhileLoop
resource:
GoalInvoke
name: AuthenticateMedicalPersonnel
type: GoalInvoke
resource: "Authenticate Medical Personnel"
GoalInvoke
name: OpenFrontDoor
type: GoalInvoke
resource: "Open Front Door"
Palette
Select
Marquee
Connections
Objects
Types
Ports
Stacked
Error Log
Properties
Problems
Main

Goal:
EPL:
☐ Cyber-Physical
MapeProId:
Mape...ess:
Type:
Resource:
Description:
Name: AutomaticEmergencyCallScenario
ID: _nsYj8FZ0Eee1qZpHnV9_wg

Figure 7.2: Eclipse-based workflow model editor

7.2.2 Semantic Access Layer

Figure 7.1 depicts the core components of the SAL in UML notation. We provide public service-based interfaces for accessing the SAL's functionality. All components are directly integrated as plugins into the OpenHAB Home project [SZZ14]. The *ResourceDiscoveryService* enables SPARQL and goal-based access to the ontological knowledge base. In correspondence with the introduced *SemanticInvoke* workflow activities, the *SemanticAccessLayerService* provides methods to either execute a select query, send a command, execute an *Ask* query, or to execute a goal. The implementation of the semantic service directly accesses available things, items, and events from the OpenHAB platform. In addition, it uses the semantic resources within the SAL and exposes their functionalities as REST services. In the case of a goal-based activity, the semantic access layer service delegates the goal to query transformation to the resource discovery service. Here, the goal reasoning service is used to transform a goal with optional quality parameter into a semantic query based on a mapping description and Tropos goals model. The transformation is performed by using backward reasoning to find fitting sets of leaf goals to satisfy the input goal. The ordering of the sets is performed by their contribution to the specified quality parameter. If no quality parameter is specified, the sets are ordered by size from smallest to biggest. The transformed set of queries is then used to execute the included queries in sequential order until a result is generated. In addition, the semantic reasoning service provides access to the ontology via the OWL API and uses the HermiT Reasoner to infer object property relations defined in SWRL rules. The semantic reasoning service implements the monitoring phase of workflow management and therefore detects and resolves context mismatches. This includes the monitoring of existence of *plays* relations without the corresponding *canPlay* relation.

7.3 User Perspective

In the user perspective, we extended the existing PROtEUS workflow model editor in alignment with the metamodel extension. Therefore, it is possible to include semantic query-based as well as goal-based workflow activities as part of a workflow model. Figure 7.2 shows the workflow model editor with the corresponding graphical user interface for drag and drop modeling of IoT-aware workflows. The property window at the bottom allows for the specification of goals, related quality parameters, as well as semantic queries.

7.4 Summary

In this chapter, we described the technical properties of the proof-of-concept implementation for the contributions of this work. This includes an extended workflow modeling editor and the Semantic Access Layer (SAL) middleware for the context-sensitive resource discovery and invocation. The prototype is implemented in conjunction with the existing PROtEUS workflow management system for cyber-physical systems. Furthermore, the SAL is implemented as an OpenHAB plugin to provide a homogeneous access to local sensors and actuators within our smart home test lab.

8 Evaluation

In this chapter [1], we evaluate our contributions regarding the fulfillment of the criteria for IoT-aware workflow management introduced in Section 3.2. First, we revisit the defined goals of this work to provide a foundation for the identification of individual features to test. Then, we introduce an extended version of the example application scenario introduced in Section 5.3. Furthermore, we provide the corresponding extended ABox of the ontology for role-based *Things* in the IoT. Based on the identified features to test, we develop several test cases within the application scenario and provide the entailed workflow models, goal models, mapping descriptions, SPARQL queries, and SWRL rules.

8.1 Goal and Evaluation Approach

Figure 3.5 shows the targeted goals of this work within the classification scheme developed in Chapter 3. The main contributions of this work target the requirements **Context-aware** (RQ_CA), **Query support** (RQ_QS), **Transparency** (RQ_Tra), **Resolution strategy** (RQ_RS), and **Modeling support** (RQ_MS). Secondary requirements are the **Dynamics** (RQ_Dyn), **Fault tolerance** (RQ_FT), **Flexibility support** (RQ_FS), **Heterogeneity** (RQ_Het), **Quality of Service** (RQ_QoS), and **Quality of Context** (RQ_QoC). The real-time processing as well as the processing of data streams are not included in the goals of this work. However, we also conduct a performance evaluation in Section 8.3. To generate the necessary test cases for evaluating the individual requirements, we revisit the relations of requirements to contributions as shown in the following Table 8.1 as an extension to Table 3.8 in Section 3.4.1.

---

[1] This chapter is partially based on [HSK$^+$16, HSS16]

Table 8.1: Revisited relations of contributions to requirements.

Requirement	Relation to Contribution				Test Case
	Middleware	Ontology	Workflow metamodel	Goal model	
RQ_FT	✓	-	-	-	1
RQ_CA	-	✓	-	-	2
RQ_Het	-	✓	-	-	
RQ_MS	-	-	✓	-	3
RQ_Dyn	✓	✓	-	-	4
RQ_FS	✓	✓	-	-	
RQ_QoC	✓	✓	-	-	
RQ_QS	✓	✓	✓	-	5
RQ_RS	✓	✓	-	✓	6
RQ_Tra	-	✓	✓	✓	7
RQ_QoS	-	✓	✓	✓	

We clustered the requirements for their relation to the four contributions of this work. In result, we generated seven groups with different subsets of relations to the contributions. We choose to separate the test cases in this way, to focus and limit their specification on the relevant parts of our contributions. In the following, we present the approach for each of the seven test cases.

8.1.1 Definition of Test Cases

To define the test cases according to their usage of specific system components and models, we revisit the Table 3.1 and Table 3.2 which define the relation between the requirements and the phases of resource and workflow in the IoT. From these, we identified the relevant phases in IoT-aware resource and workflow management for each test case according to their encapsulated requirements as defined in Table 8.1. The results are presented in the following Table 8.2 and provide the foundation for the specification of the test cases. In the following, we will define the approach for each test case.

Table 8.2: Relation of test cases to resource and workflow management.

Test Case	Resource Management					Workflow Management			
	Model.	Disc.	Est.	Alloc.	Mon.	Model.	Conf.	Exec.	Adapt.
1	-	-	-	-	-	-	-	-	✓
2	✓	✓	-	-	✓	-	-	-	-
3	-	-	-	-	-	✓	-	-	-
4	✓	✓	-	-	✓	✓	-	✓	✓
5	-	✓		-	-	-	-	-	-
6	-	-	-	-	-	-	-	✓	-
7	-	-	✓	-	✓	✓	-	-	-

Test Case 1

Test case 1 aims at evaluating the **fault tolerance** within the adaptation phase of workflow management and is related to the SAL middleware. As defined in the classification scheme in Table 3.4, fault tolerance can be provided by either re-allocating resources, supporting virtual consistency or by supporting cyber-physical consistency. These three levels are additive. However, in the definition of the goals of this work as depicted in Figure 3.5, we choose that the fault tolerance only has to be supported by re-allocation of resources. Therefore, test case 1 simulates a context change for an allocated resource such that the new context does not allow the *Thing* to play the *Role* in which it is currently allocated. In consequence, the resource monitoring component of the SAL should detect the context mismatch and trigger a re-allocation.

Test Case 2

Test case 2 aims at evaluating the **context-aware** modeling of **heterogeneous** *Things* within the modeling, discovery and monitoring phases of resource management and is related to the ontology for role-based *Things*. As defined in the classification scheme in Table 3.4, context-aware modeling can be provided by modeling location-based context, context-sensitive capabilities, and the context-sensitive relations between *Things*. In this test case, we address all of these aspects because we choose them as a goal of this work. In addition, heterogeneity can be bridged by modeling model-based input and output data, *Things*, and semantic *Things*. In this test case, we consider all these aspects. Therefore, test case 2 includes the modeling of similar *Things* able to play the same *Role* to bridge their heterogeneity. Furthermore, to demonstrate the context-awareness of our contribution, the application scenario includes the discovery and monitoring of *Things* based on their current location and capabilities. In addition, we provide an SWRL rule which implements a role implication. Thus, when a specific role 1 is played, the related *Thing* also has to play the specific role 2.

Test Case 3

Test case 3 aims at evaluating the **modeling support** within the modeling phase of workflow management and is related to the workflow metamodel. As defined in the classification scheme in Table 3.4, modeling support can be provided by modeling sensors, actuators, and context-aware *Things* with complex capabilities. Here, the modeling support is related to the workflow model. Therefore, it relates to the expressiveness of the workflow model regarding the specification of resource requirements. Test case 3 provides example semantic queries to demonstrate the specification of activity requirements related to sensors, actuators, and context-sensitive *Things*.

Test Case 4

Test case 4 aims at evaluating the **dynamics**, the **flexibility support**, and the **Quality of Context** requirements using the ontology and the SAL. More specifically, test case 4 is related to the modeling, discovery and monitoring phase in resource management as well as the modeling, execution and adaptation phase in workflow management. The *dynamics* have to be supported in syntactic, model-based, and semantic resource discovery. The *flexibility support* includes late binding, using context and resource constraints as well as the continuous monitoring and re-allocation of workflow resources. The *Quality of Context* can be expressed within context parameters, use an update mechanism, and provide a real-time consistency in terms of cyber-physical consistency. However, in this work we limited the goals to allow for the expression of context parameters and including an update mechanism. Therefore, test case 4 includes a late binding scenario based on semantic queries for defining several resource constraints. These include a syntactic definition, i. e., individual, and the definition of semantic constraints for the resource discovery. Moreover, the continuous monitoring and re-allocation is also included in test case 1. However, in test case 4 the monitoring and re-allocation is shown in an extended application scenario for resolving a context mismatch.

Test Case 5

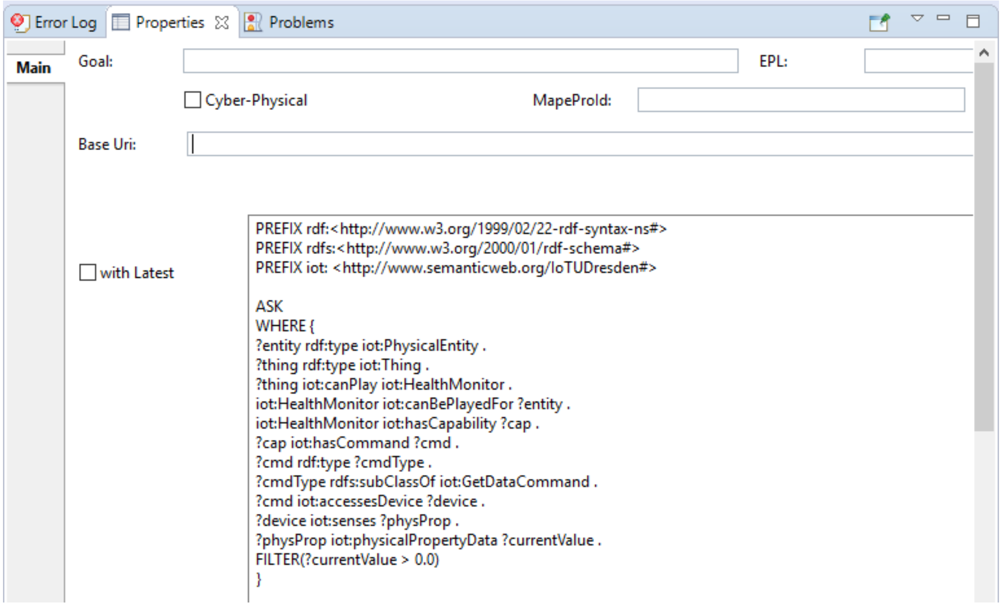
Test case 5 aims at evaluating the **query support** within the discovery phase of resource management and is related to the SAL middleware, the ontology, and the workflow metamodel. As defined in the classification scheme in Table 3.4, query support can be provided by exact match queries, range queries and by semantic queries. Our approach is designed to support semantic queries which entails the support for exact match queries and range queries. Test case 5 provides example semantic queries to demonstrate the specification of the required query types. Furthermore, we show their definition as part of a workflow model and execute the queries to provide the results within the workflow engine.

Test Case 6

Test case 6 aims at evaluating the **resolution strategy** within the execution phase of workflow management and is related to the SAL middleware, the ontology, and the goal model. As defined in the classification scheme in Table 3.4, a resolution strategy can use context parameters, QoC criteria, and QoS criteria to resolve the ambiguity in discovering resources. Test case 6 provides an example goal model and a goal-based workflow model. Based on these, we demonstrate the resolution within resource discovery as part of the SAL.

Test Case 7

Test case 7 aims at evaluating the **transparency** in workflow modeling and the specification of **Quality of Service** parameters for the estimation and monitoring of *Things*. Therefore, test case 7 includes the ontology, the workflow metamodel and the goal model. As defined in the classification scheme in Table 3.4, transparency for workflow modelers can be provided by allowing the modeling of resource requirements, resource constraints and intentions of activities. Furthermore, the *Quality of Service* can be provided by modeling support, QoS-aware resource discovery and continuous QoS monitoring. This work aims at the QoS-aware resource discovery. Test case 7 includes the modeling of QoS parameters as part of a goal model. Furthermore, we use these goals to specify the intention of a workflow activity in a workflow model and execute it to demonstrate the QoS-aware resource discovery.



```
PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs:<http://www.w3.org/2000/01/rdf-schema#>
PREFIX iot: <http://www.semanticweb.org/IoTUDresden#>

ASK
WHERE {
?entity rdf:type iot:PhysicalEntity .
?thing rdf:type iot:Thing .
?thing iot:canPlay iot:HealthMonitor .
iot:HealthMonitor iot:canBePlayedFor ?entity .
iot:HealthMonitor iot:hasCapability ?cap .
?cap iot:hasCommand ?cmd .
?cmd rdf:type ?cmdType .
?cmdType rdfs:subClassOf iot:GetDataCommand .
?cmd iot:accessesDevice ?device .
?device iot:senses ?physProp .
?physProp iot:physicalPropertyData ?currentValue .
FILTER(?currentValue > 0.0)
}
```

Figure 8.1: Extract of the PROtEUS workflow model editor’s property pane

8.2 Scenario Evaluation

In order to evaluate the test cases within our proof-of-concept implementation, we extend the smart home application scenario introduced in Section 5.3 towards an AAL scenario. We use the PROtEUS WfMS described in [SHS15] with an extended workflow meta-model according to the concepts presented in Section 4.3.1. Figure 8.1 shows an example of the semantic query annotation created with the workflow editor, which allows us to model semantically annotated activity. The SPARQL query is contained as a property in the respective activity. In the same way, we can annotate an activity with a goal from the Tropos goal model. The SAL is configured to integrate the ontology for role-based *Things* in the IoT. The OpenHAB (Open Home Automation Bus) [SZZ14] platform serves as an IoT service provider and low-level middleware supporting a unified view on device capabilities in the Smart Home domain. However, OpenHAB does not provide semantic descriptions for these functionalities, which is why we model the device functionality with the help of our ontology to enable dynamic IoT service discovery and invocation via the SAL.

8.2.1 Ambient Assisted Living Setting

Applications in the Ambient Assisted Living (AAL) domain employ sensors, actuators, and software components in order to support elderly people in living a self-determined life. In the following, we present a modified version of the automatic emergency call scenario from [SHS15]: Alice and Bob are living in an AAL-enabled home. Bob is in the kitchen, listening to his favorite music while cooking. Suddenly, his blood pressure drops and Bob faints. The desired reaction to this situation is as follows: When Bob faints, the Smart Home should detect his critical health situation. Upon this detection, Bob should be contacted and asked for his state of health. If he is unresponsive, the Smart Home should place an automatic emergency call. In the meantime, all critical appliances, e. g. the stove, in the apartment should be turned off to prevent further hazardous situations. When medical personnel arrives and is authenticated at the front door, the door should unlock automatically and Bob can be provided with medical attention.

8.2.2 Resource Perspective

First, we selected the devices to be integrated into OpenHAB for implementing the AAL scenario: the Kodi Open Source Home Theatre Software running on a Raspberry Pie computer for playing both Bob's favourite music and an audio message asking for Bob's health status; a digitalSTROM connector powering the kitchen stove; the Libelium eHealth kit for monitoring Bob's health status. The front door lock is controlled by a HomeMatic KeyMatic lock. In order to identify medical personnel at the front door, we apply an NFC card-based authentication mechanism. Each device is connected to a central OpenHAB server via Ethernet, Wifi, Bluetooth or USB. For every device, we created the appropriate ABox statements including the corresponding *Things*, *Roles*, capabilities, and their relative locations.

Role-based Resource Ontology

We added the individuals and property as well as data object assertions to the ABox of the ontology for role-based *Things*[2]. The following Figure 8.2 gives an overview of the individuals and their relations for the extended application scenario.

---

[2] The added axiomns for the extended application scenario are listed in the Appendix Section 10.2

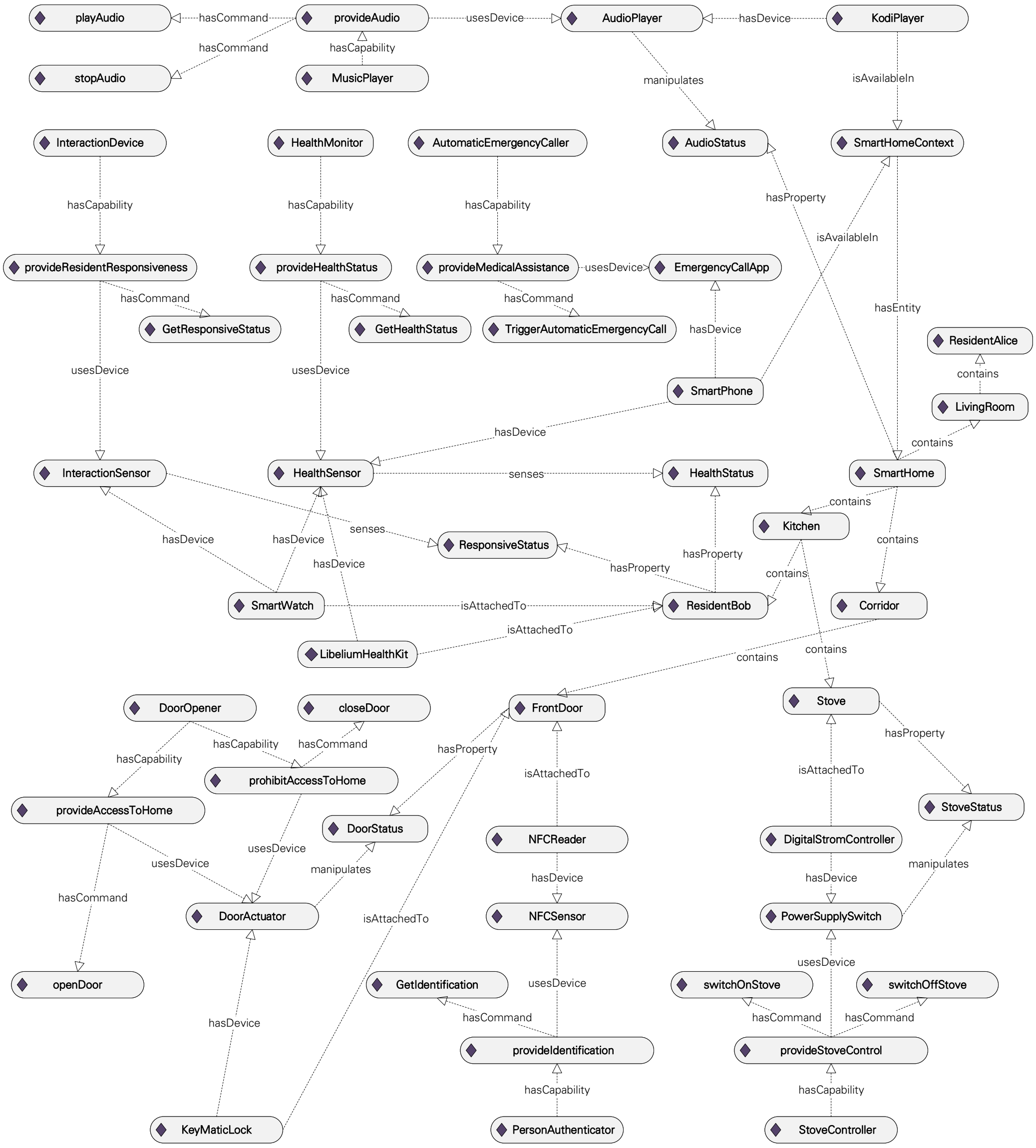


Figure 8.2: Extended ontology Abox for test case evaluation.

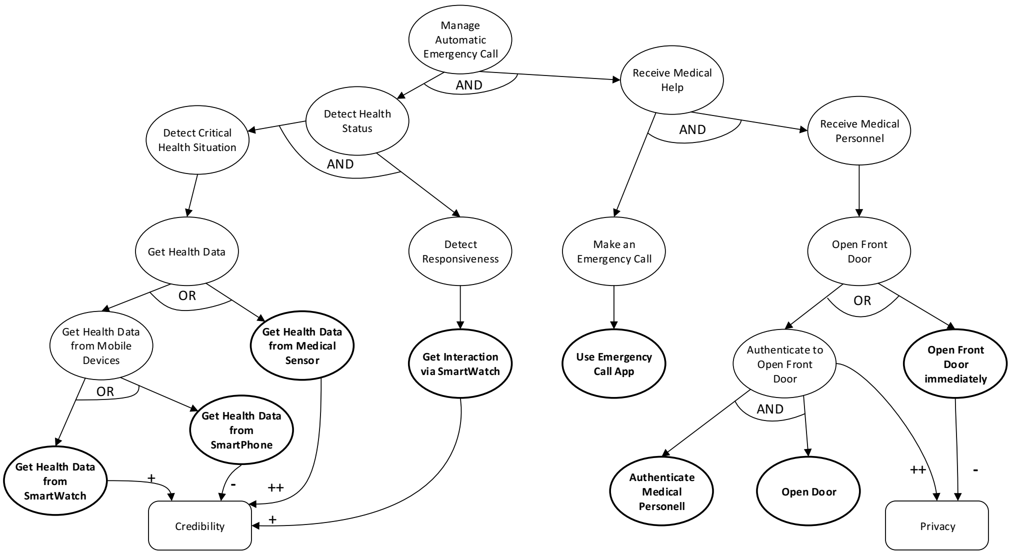

Manage Automatic Emergency Call
AND
Receive Medical Help
Detect Health Status
Detect Critical Health Situation
AND
AND
Receive Medical Personnel
Get Health Data
Detect Responsiveness
Make an Emergency Call
Open Front Door
OR
OR
Get Health Data from Mobile Devices
Get Health Data from Medical Sensor
Get Interaction via SmartWatch
Use Emergency Call App
Authenticate to Open Front Door
Open Front Door immediately
OR
Get Health Data from SmartPhone
AND
Get Health Data from SmartWatch
+
-
++
+
Credibility
Authenticate Medical Personell
Open Door
++
-
Privacy

Figure 8.3: Tropos Goal Model for test case evaluation.

8.2.3 User Perspective

In the user perspective, we provide a basic goal model for evaluating the test cases. The included goals do not cover all of the capabilities within the smart home scenario. Within each test case, we discuss the application of the goals for modeling and provide the respective mapping description.

Goal Model

Goal modeling for the scenario process was conducted within the Tropos goal modeling language both for the test cases. The model is illustrated in Figure 8.3. It includes all functional goals and their relations that lead to workflow-based health assistance to the Smart Home residents. The leaf nodes of the goal model represent capabilities, i. e., role-based capabilities, that contribute to the satisfaction of the respective upper goals. From this model follows, that in order to detect a critical health situation, the Smart Home needs to include at least one device that is able to produce health data, e. g., smartwatch, smartphone, or dedicated medical sensor. The goal model enables backward reasoning as described in Section 2.2.2. The leaf goals are marked with a bold edge and represent the capabilities provided by

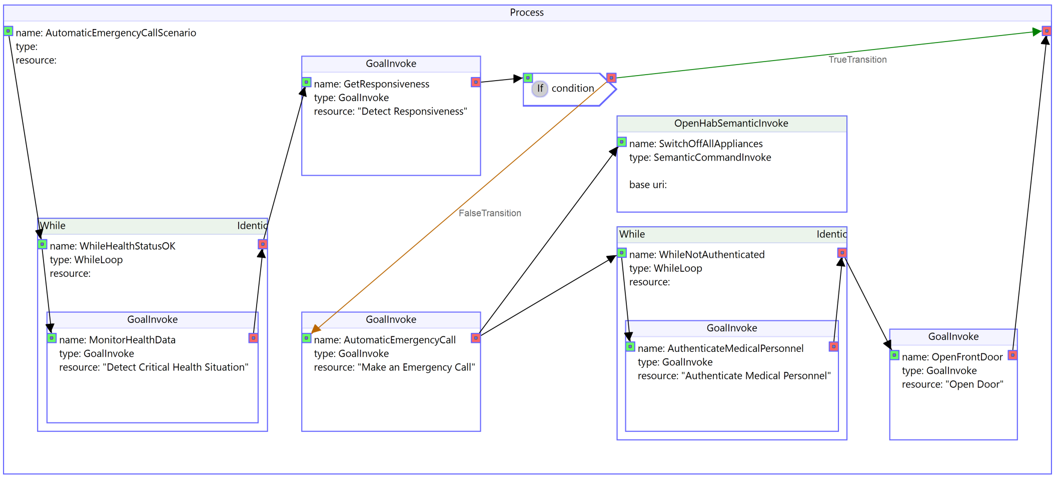


Figure 8.4: Extended Automatic emergency call scenario.

the *Things* and encapsulated within *Roles*. For each leaf goal, there exists a mapping to a SPARQL query in the mapping description. Non-functional requirements, i. e., Quality of Service parameters, are illustrated as rectangles with round edges. Each goal may contribute positively (+) or negatively (-) to any number of non-functional requirements. For example, in order to open the front door to receive medical personnel, the workflow can either open the door immediately, possibly violating the privacy of the residents, or use authentication.

8.2.4 Workflow Perspective

In the workflow perspective, we provide an extended workflow model for the application scenario shown in Figure 8.4. The first activity is modeled inside a loop and executed every 10 seconds. We use the first activity of type *GoalBasedInvoke* to illustrate the concept of the additional goal-based abstraction layer. The intended behavior of the process step is to retrieve sensor data from devices that provide the capability to measure health-related data, e. g., blood pressure, and are related to a resident. Therefore, we add the goal *Detect Critical Health Situation* and the non-functional requirement of *credibility* from the goal model depicted in Figure 8.3 to the properties of the activity. In the case of a critical health state, the loop is discontinued and the subsequent activities are executed. The following activity detects if the resident is in fact unresponsive. In this case, an automatic emergency call is place. Then, the next two activities are executed in parallel to switch off all appliances, e. g., stove, in the smart home to prevent further safety hazards. In addition, the other activity tries to get the information indicating an authentication of medical personnel. Once the medical personnel arrived at the front door and used their NFC card, the final activity opens the front door such that the medical personnel can enter the smart home and treat the resident.

8.2.5 Execution of Test Cases

In the following, we conduct the execution of the seven test cases defined in Section 8.1.1 to evaluate the fulfillment of the targeted requirements with this work.

Execution of Test Case 1

In this test case we evaluate the fault tolerance within the SAL. In the scenario, we assume that the *iot:SmartWatch* individual of type *iot:Thing* is related to the *iot:HealthMonitor* individual of type *Role* via the *plays* and the *canPlay* relation:

$$plays(SmartWatch, HealthMonitor)$$
$$canPlay(SmartWatch, HealthMonitor)$$

The associated rule, defining the constraints for the *canPlay* relation is as follows:

$$\begin{gathered} hasDevice(SmartWatch, HealthSensor) \land \\ usesDevice(provideHealthStatus, HealthSensor) \land \\ hasCapability(HealthMonitor, provideHealthStatus) \land \\ hasCapability(InteractionDevice, provideResidentResponsiveness) \land \\ isAttachedTo(SmartWatch, ?x) \rightarrow \\ canPlay(SmartWatch, HealthMonitor) \land canBePlayedFor(HealthMonitor, ?x) \end{gathered}$$

When removing the smartwatch from the arm of a resident and updating this context change in the ontology by removing the axiom isAttachedTo(SmartWatch, Bob), the swrl rule inference also excludes the axiom canPlay(SmartWatch, HealthMonitor). The SAL now only detects the *plays* relation representing a context mismatch:

$$plays(SmartWatch, HealthMonitor)$$

The SAL then removes this axiom and reruns the associated SPARQL query to find another fitting resource. As allocated resources are directly invoked for the retrieval of data or the triggering of an actuating task, the re-allocation will often be of no effect to the executed workflow. However, in the case of blocking *GetDataCommands*, e. g., using an interaction device for user feedback, the re-allocation will produce a noticeable effect.

Execution of Test Case 2

In this test case we evaluate the modeling of context-aware and heterogeneous *Things* and provide a role implication rule within the ontology. In the scenario, we assume that the *iot:SmartHome* individual of type *iot:PhysicalEntity* contains a *iot:Corridor* of type *iot:PhysicalEntity* which in turn contains a *iot:FrontDoor.* Through the transitive *contains* relation, the reasoner infers that the front door is contained in the Smart Home:

$$contains(SmartHome, FrontDoor)$$

Furthermore, we introduce the role *iot:ResponsiveResident* to represent typical state of a resident. We define the restriction to play this role as follows:

$$\begin{gathered} hasProperty(?x, \mathit{ResponsiveStatus}) \wedge \\ physicalPropertyData(\mathit{ResponsiveStatus}, true) \rightarrow \\ plays(?x, \mathit{ResponsiveResident}) \wedge canPlay(?x, \mathit{ResponsiveResident}) \end{gathered}$$

Even though, we did not specify PhysicalEntities to be able to play roles. The open-world assumption[3] in OWL allows for such axioms. Based on this, we define the following role implication such that a resident who is responsive also open doors, i. e., play the role of a door opener, if he is in the same location as the door:

$$\begin{gathered} plays(?x, \mathit{ResponsiveResident}) \wedge contains(?y, ?x) \wedge contains(?y, FrontDoor) \rightarrow \\ canPlay(?x, \mathit{DoorOpener}) \end{gathered}$$

However, such an implication is only possible after the inferred *contains* relation is transferred into an explicit axiom of the ontology. To illustrate how the heterogeneity is bridged via the role concept, we provide the following SPARQL query shown in Listing 8.1:

---

[3] The open-world asssumption implies that axioms which are not part of the ontology are considered to be unknown as the closed world assumption implies that everything that is not stated in a model is false.

```
PREFIX iot: <http://www.semanticweb.org/IoTUDresden#>
SELECT ?x
WHERE {
       ?x iot:canPlay iot:DoorOpener.
}
```

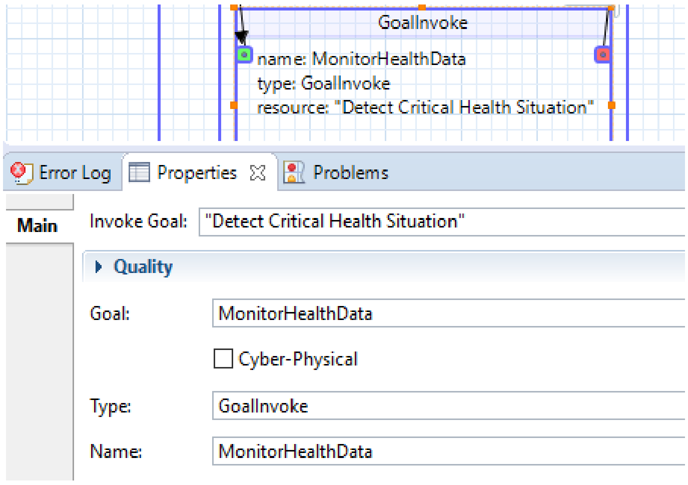
Listing 8.1: Get all individuals which can play the role of door opener

the query result is presented in the following:

$$iot : KeyMaticLock$$
$$iot : ResidentBob$$

Execution of Test Case 3

In test case 3, we show the modeling support for the semantic queries and the specification of goals in the workflow model editor. Figure 8.5 shows the modeling of a *GoalInvoke* activity. In the property window, the goal of the activity is specified under *Invoke Goal*. Here, the **Detect Critical Health Situation** goal from the Tropos goal model is referenced. Therefore, the SAL can translate the goal into a semantic query at runtime. In addition, Quality of Service priorities can be specified under *Quality*.



name: MonitorHealthData
type: GoalInvoke
resource: "Detect Critical Health Situation"

Invoke Goal: "Detect Critical Health Situation"

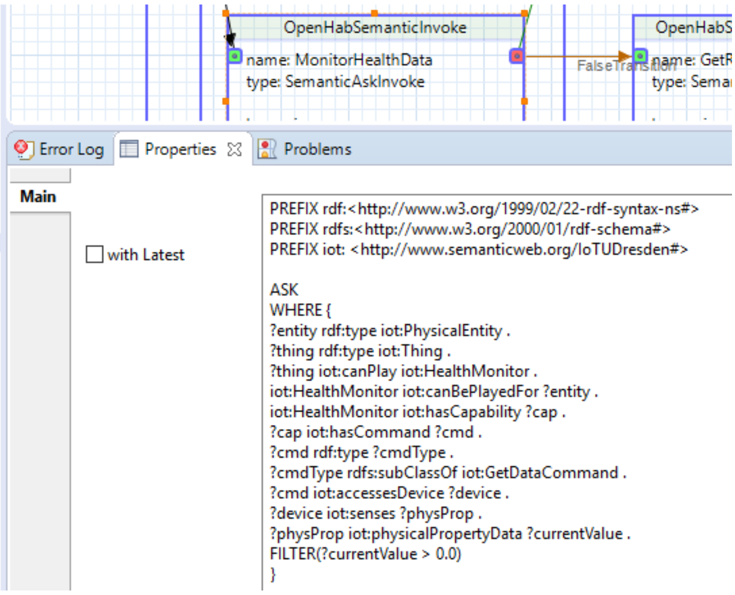
Goal: MonitorHealthData

☐ Cyber-Physical

Type: GoalInvoke

Name: MonitorHealthData

Figure 8.5: Workflow model editor view for GoalInvoke.



```
PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs:<http://www.w3.org/2000/01/rdf-schema#>
PREFIX iot: <http://www.semanticweb.org/IoTUDresden#>

ASK
WHERE {
?entity rdf:type iot:PhysicalEntity .
?thing rdf:type iot:Thing .
?thing iot:canPlay iot:HealthMonitor .
iot:HealthMonitor iot:canBePlayedFor ?entity .
iot:HealthMonitor iot:hasCapability ?cap .
?cap iot:hasCommand ?cmd .
?cmd rdf:type ?cmdType .
?cmdType rdfs:subClassOf iot:GetDataCommand .
?cmd iot:accessesDevice ?device .
?device iot:senses ?physProp .
?physProp iot:physicalPropertyData ?currentValue .
FILTER(?currentValue > 0.0)
}
```

Figure 8.6: Workflow model editor view for SemanticAskInvoke.

Figure 8.6 shows the modeling of a *SemanticAskInvoke* activity. In the property window, the query is directly specified including the prefixes. Here, the is specified to detect a critical health situation of a resident. The query only returns a result if a HealthMonitor detects a sensor value larger than 0.0. In the ontology we use the PhysicalProperty to express the severity of a critical health situation between the float values 0.0 and 1.0.

Execution of Test Case 4

In this test case, we evaluate the *dynamics*, *flexibility support*, and *Quality of Context* requirements. We use the query-based activity for continuously monitoring the health state of a resident shown in Figure 8.6. At first, the query does not provide results. However, when setting the *iot:physicalPropertyData* of the *iot:HealthStatus* of *iot:ResidentBob* to 1.0, the query result includes the *PhysicaEntity iot:ResidentBob*. To illustrate the update mechanism for the context ontology, we add the axiom:

$$hasDevice(SmartPhone, InteractionSensor)$$

Furthermore, we specify the following SWRL rule to implement the monitoring of the *canPlay* relation for smartphones to play the role of an *InteractionDevice*:

$$\begin{gathered} hasDevice(?x, InteractionSensor) \wedge \\ usesDevice(provideResidentResponsiveness, InteractionSensor) \wedge \\ isAvailableIn(?x, ?y) \wedge hasEntity(?y, ?z) \wedge contains(?z, ResidentBob) \wedge \\ hasCapability(InteractionDevice, provideResidentResponsiveness) \rightarrow \\ canPlay(?x, InteractionDevice) \wedge \\ canBePlayedFor(InteractionDevice, ResidentBob) \end{gathered}$$

It specifies, that the smartphone must have an *InteractionSensor*, i. e., typically a smartphone app with the ability to detect a touch event. Furthermore, the role *InteractionDevice* needs to provide the capability *provideResidentResponsiveness* which in turn has to be linked to the *InteractionSensor*. As a location constraint, we specify that the smartphone need to be available in the context which contains Bob. At runtime, the rule inferred the following relations:

$$\begin{gathered} canPlay(SmartPhone, InteractionDevice) \\ canBePlayFor(InteractionDevice, ResidentBob) \end{gathered}$$

Execution of Test Case 5

In test case 5 we provide several example queries for the specification of workflow activities. At first, exact match queries can be used to retrieve a specific resource as shown in Listing 8.2:

```
PREFIX iot: <http://www.semanticweb.org/IoTUDresden#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
SELECT *
WHERE {
?thing owl:sameAs iot:HealthMonitor .
}
```

Listing 8.2: Exact match query example.

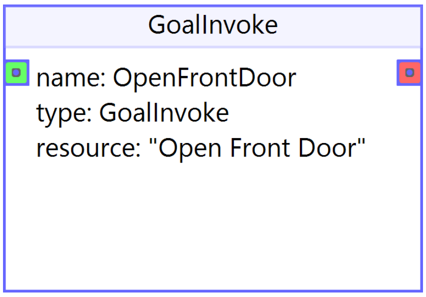
In itself, the query does not provide a useful result, as it merely returns *iot: HealthMonitor*. However, it is useful when extended for the retrieval of a specific role or its referenced commands. In addition, we provide the following range query to retrieve all *PhysicalEntities* with a critical health situation as shown in Listing 8.3:

```
PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX iot: <http://www.semanticweb.org/IoTUDresden#>
SELECT ?entity WHERE {
?entity rdf:type iot:PhysicalEntity .
?thing rdf:type iot:Thing .
?thing iot:canPlay iot:HealthMonitor .
iot:HealthMonitor iot:canBePlayedFor ?entity .
iot:HealthMonitor iot:hasCapability ?cap .
?cap iot:hasCommand ?cmd .
?cmd rdf:type ?cmdType .
?cmdType rdfs:subClassOf iot:GetDataCommand .
?cmd iot:accessesDevice ?device .
?device iot:senses ?physProp .
?physProp iot:physicalPropertyData ?currentValue .
FILTER(?currentValue > 0.0)
}
```

Listing 8.3: Range query example.

Execution of Test Case 6

In test case 6, we consider the workflow activity to specify the goal **Open Front Door** as shown in Figure 8.7. At runtime, the SAL finds the fitting leaf goal sets {Open Front Door immediately} and {Authenticate Medical Personel, Open Door}. If not further specified, the SAL will order the sets by size and choose the first set of leaf goals. In this case, the includes leaf goal **Open Front Door immediately** is chosen and translated into a SPARQL query using the mapping description.



name: OpenFrontDoor
type: GoalInvoke
resource: "Open Front Door"

Figure 8.7: Goal-based activity to open the front door.

```
Open Front Door immediately ->
PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX iot: <http://www.semanticweb.org/IoTUDresden#>
SELECT ?cmd ?entity WHERE {
?entity rdf:type iot:PhysicalEntity.
?thing rdf:type iot:Thing .
?thing iot:canPlay iot:DoorOpener .
iot:DoorOpener iot:hasCapability ?cap .
iot:DoorOpener iot:canBePlayedFor ?entity .
?cap iot:hasCommand ?cmd .
?cmd rdf:type ?cmdType .
?cmdType rdfs:subClassOf iot:OnCommand .
}
```

Listing 8.4: Query Mapping for *Open Front Door immediately.*

The query result includes the desired command and entity *iot:openDoor* and *iot:FrontDoor.* The command does include a reference to the service method for the door actuator.

Execution of Test Case 7

In test case 7, we demonstrate the QoS-aware resource discovery for the activity to detect a critical health situation. Here, the goal **Detect Critical Health Situation** is defined alongside the quality parameter *Credibility.* As depicted in Figure 8.8, this goal can be fulfilled by each of the three leaf goals **Get Health Data from SmartWatch**, **Get Health Data from SmartPhone**, and **Get Health Data from Medical Sensor**. However, they each contribute differently to the quality parameter *credibility.* At runtime, the SAL finds the three sets {Get Health Data from Medical Sensor, ++}, {Get Health Data from SmartWatch, +} and {Get Health Data from SmartPhone, -}. They are ordered for their contribution to the specified quality parameter. Then, the SAL translates the first leaf goal to a SPARQL query and executed it on the knowledge base. However, in the current context, we modeled the *LibeliumHealthKit* to be unavailable, i. e., there exists no *canPlay* relation to the role *HealthMonitor.* In this case, the SAL takes the next leaf goal, translates it and executes it on the knowledge base. As this process is iterated by the while loop in the workflow model, the health state is monitored dynamically with any available *Thing* fitting the current runtime context.

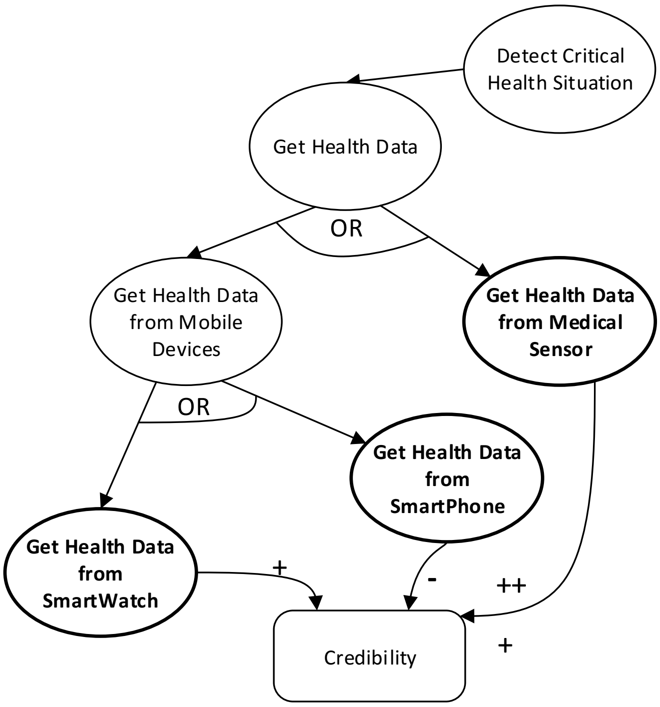

Detect Critical Health Situation
Get Health Data
OR
Get Health Data from Mobile Devices
Get Health Data from Medical Sensor
OR
Get Health Data from SmartPhone
Get Health Data from SmartWatch
+
-
++
+
Credibility

Figure 8.8: Excerpt of goal model for detecting a critical health situation.

8.2.6 Discussion of Results

Within the seven test cases we demonstrated the fulfillment of each of the targeted goals in this work. Even though, the fulfillment of these criteria was targeted by design, the criteria themselves are often abstract and can be fulfilled by a range of concepts. For example, the DL-safe SWRL rules are restricted to the specification of constraints of individuals and do not extend to classes. Therefore, the rules can become large, if they are targeted to specify relations for a type of individual over its relations to other types of individuals. In addition, due to the open-world assumption there can be ambiguity and inferred relations that hinder an isolated specification of rules for a specific type of individual. Therefore, the definition of SWRL rules and object property assertions are prone to failures such that unwanted relations are generated by the rules. Furthermore, it is best to keep the inferred

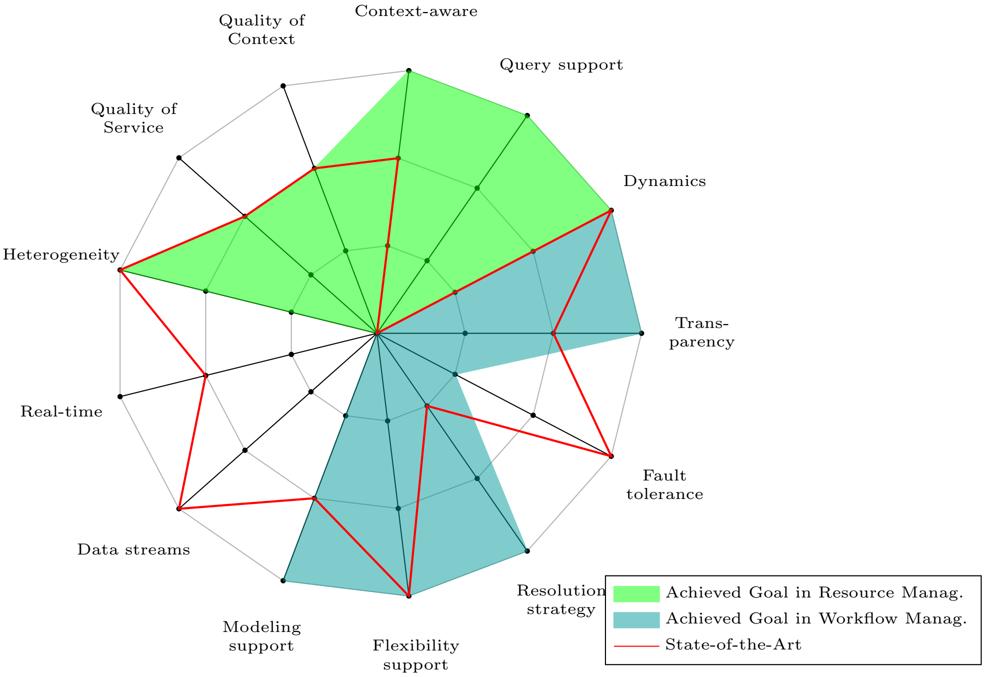


Figure 8.9: Revisiting the criteria and goals within this work.

axioms separated from the modeled axioms to allow for their quick update. Figure 8.9 shows the achieved goals within this work in relation to the state-of-the-art approaches. The metrics of the ontology for role-based *Things* in the IoT as it is used within the application scenario are listed in the following Table 8.3.

Table 8.3: Metrics of the ontology within the application scenario.

Axiom count	667
Logical axiom count	575
Declaration axioms count	92
Class count	14
Object property count	22
Data property count	2
Individual count	54
Annotation Property count	3
DL expressivity	SHOIQ(D)

8.3 Performance Evaluation

Even though, we did not choose the real-time processing as a requirement for the SAL, the ontology reasoning or the communication with the WfMS; we want to conduct a performance measurement due to the possible large-scale of IoT applications. Although we do not reach levels of real-time processing, our approach can fulfill near real-time resource discovery and invocation in some applications. In the following, we present the conducted performance measurements for each of the three query types available to the workflow activity modeling. It is important to state that these measurements were conducted using the DogOnt ontology for the modeling of domotic environments [BC08]. We choose the ontology as it was easily integrable into the SAL and the OpenHab runtime environment to retrieve real-world sensor data. Furthermore, we wanted to demonstrate the exchangeability of the underlying ontology within the SAL. Our example scenarios cover only a small-scale system with only a few available *Things* and *Roles* to demonstrate the concepts of workflow-based dynamic service invocation. In the following, we will verify our approach with respect to scaling up also to large-scale, heterogeneous distributed systems. We present several conducted experiments to measure the performance overhead introduced by using the SAL for semantic service discovery and invocation. The experiments are designed to measure query performance of all three semantic query types, based on the introduced *SemanticInvoke* process step types.

8.3.1 Experimental Setup

In the experiment, SPARQL test queries were utilized to check and retrieve the current luminance level inside a room from sensor data and to switch on a dimmer actuator. For the *SemanticAskInvoke* we utilized the SPARQL query depicted in Listing 8.5, which checks whether the current luminance level is lower than the provided lumen value of 290.1 or not. The evaluation of this query results in a Boolean value. The *SemanticCommandInvoke* was tested with the SPARQL query shown in Listing 8.6. It retrieves all available dimmer switches for a given location. For each resulting device, a service call is generated by the SAL to switch it on. However, as we are investigating the overhead of the SAL with respect to execution time, the actual service invocation is not taken into account. The *SemanticSelectInvoke* was tested with the SPARQL query shown in Listing 8.7. This query retrieves sensor values from the knowledge base, which represent the current luminance level in a given location. We utilized the DogOnt ontology as knowledge base of the SAL. As a baseline setting, the knowledge base included 100 OWL classes and 480 triples. Also counting OWL imports and inference, the base included a total of 1026 OWL classes and 16002 triples.

```
ASK WHERE {
   instance:State_Current_Location
              dogont:hasStateValue ?stateValue .
   ?stateValue dogont:realStateValue ?curRealLoc .
   ?thing dogont:hasState ?lightState .
   ?lightState dogont:hasStateValue ?lightValue .
   ?lightValue rdf:type ?type.
   ?type rdfs:subClassOf* dogont:BrightnessStateValue .
   ?lightValue dogont:realStateValue ?currentValue .
   ?thing dogont:isIn ?loc .
   ?loc rdfs:label ?realLoc .
FILTER(?curRealLoc = ?realLoc &&
              xsd:double(?currentValue) < 290.1)
}
```

Listing 8.5: SPARQL ask query for checking current illuminance level

```
SELECT ?func WHERE {
   instance:State_Current_Location
              dogont:hasStateValue ?stateValue .
   ?stateValue dogont:realStateValue ?curRealLoc .
   ?thing dogont:hasFunctionality ?func .
   ?thing rdf:type ?thingType .
   ?thing dogont:isIn ?loc .
   ?thingType rdfs:subClassOf* dogont_DimmerSwitch .
   ?func dogont:hasCommand ?cmd .
   ?cmd rdf:type ?cmdType .
   ?cmdType rdfs:subClassOf* dogont:OnCommand .
   ?loc rdfs:label ?realLoc .
FILTER(?realLoc = ?curRealLoc)
}
```

Listing 8.6: SPARQL command query for activating a dimmer actuator

```
SELECT ?realLoc ?curRealLoc ?lightState ?currentValue
WHERE {
   instance:State_Current_Location
              dogont:hasStateValue ?stateValue .
   ?stateValue dogont:realStateValue ?curRealLoc .
   ?thing dogont:hasState ?lightState .
   ?lightState dogont:hasStateValue ?lightValue .
   ?lightValue rdf:type ?type.
   ?type rdfs:subClassOf* dogont:BrightnessStateValue .
   ?lightValue dogont:realStateValue ?currentValue .
   ?thing dogont:isIn ?loc .
   ?loc rdfs:label ?realLoc .
FILTER(?curRealLoc = ?realLoc)
}
```

Listing 8.7: SPARQL select query for retrieving current luminance level

The first set of experiments aimed at measuring execution times for each semantic activity type for an increasing number of model instances. Therefore, we generated $(10^x|x = 1..6)$ additional instances in the DogOnt ontology by extending an existing luminance sensor *owl:class*. We modeled three test workflows each consisting of only one activity for each semantic activity type. We then executed the test workflows for each case on a logarithmic scale, i. e., $(10^x|x = 1..6)$. For the second set of experiments, we used the same setup but generated $(10^x|x = 1..6)$ additional classes instead of instances. Each measurement was conducted 10 times. The measurement times shown in Figures 8.10 are mean values with standard deviations ranging from 19.188 % to 54.786 %. The experiments were conducted on an Intel Core-i7-3770S computer equipped with 8192 MB RAM running a Ubuntu Linux distribution.

Performance Impact of Dynamic Service Discovery

Figure 8.10 shows the query performance for each semantic process step type for the case of additionally generated ontology instances and classes. While the *SemantiCommandInvoke* shows a linear scaling even for up to 10^6 instances, the *SemanticAskInvoke* and *SemanticSelectInvoke* queries slow down the system significantly if there are more than 10^3 instances existing. The evaluation of the *SemanticAskInvoke* took an average of 366.216 ms for 10^5 and 68.044 s for 10^6 instances. The *SemanticSelectInvoke* took an average of 850.784 ms for 10^5 and 108.657 s for 10^6 instances. In contrast to the *SemantiCommandInvoke*, the *SemanticSe-*

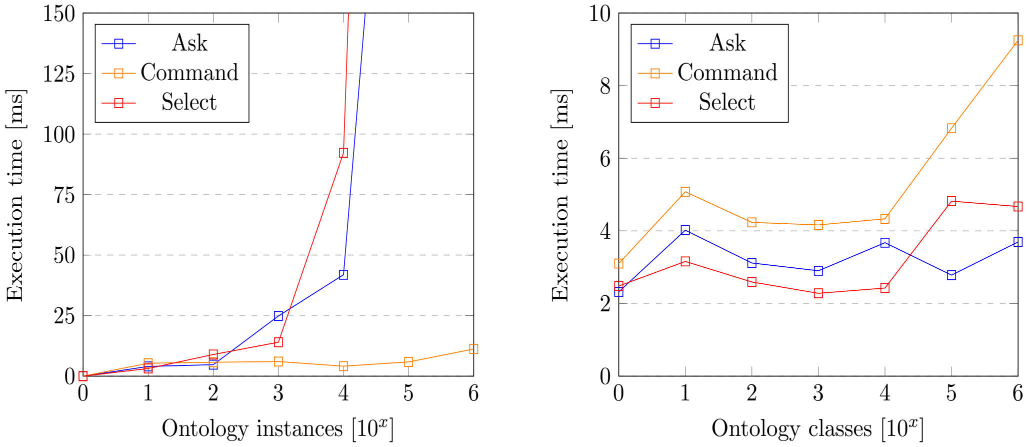


Figure 8.10: Query performance scaling for ontology instances and classes

lectInvoke and the *SemanticAskInvoke* retrieve sensor values from the knowledge base. This indicates that the retrieval of sensor values is the main reason for longer execution times and the limited overall scalability with respect to the number of instances. While an increase inthe number of instances slows down the performance of *SemanticAskInvoke* and *SemanticSelectInvoke* queries, an increased number of classes without associated instances has virtually no impact on query performance as shown in Figure 8.10. This result meets the expectations as the test queries are evaluated on the instance level.

8.3.2 Discussion of Results

From the experiments, we draw the following conclusion for the execution of IoT-aware workflows in large-scale systems: 1) with respect to query performance, the number of non-instantiated ontology classes is not significant. 2) A domain-specific approach including required devices should be preferred–even in large-scale scenarios. 3) There are certain limitations for executing time-sensitive workflows with our approach. In large-scale systems containing more than 10^6 IoT services, service discovery and invocation will most likely take minutes. This can be remedied to a certain degree by increasing and distributing the processing power of the SAL, e. g., by implementing the SAL as a cloud service. However, in this setup it is still feasible to execute *SemanticInvokeCommand* queries as there is no sensor data retrieved from the SAL's knowledge base.

From a workflow modeling view, the semantic queries introduce an additional overhead. Workflow modelers are not familiar with semantic technologies [TMS$^+$12]. However, there exist several approaches from the workflow and the semantic modeling communities, to generate executable models from natural language descriptions [BHK$^+$08, MPPM15]. The automatic provision and suggestions of vocabulary from the domain-specific ontology can simplify the modeling of the SPARQL query and facilitate end-user programming. Furthermore, workflow modelers can be supported by transforming natural language query descriptions into executable SPARQL queries. To simplify workflow specification and increase the usability of our approach [HSS16], we use a Tropos goal model. The goal-based modeling approach provides an additional abstraction layer for workflow modelers. It reduces the overhead of modeling detailed semantic queries while retaining the concept of dynamic service discovery and invocation for IoT-aware workflows. Our approach is suitable to be applied in very dynamic environments consisting of mobile and resource-constraint IoT devices as well as digitally enhanced things and objects. In these cyber-physical systems, the SAL knowledge base contains the real world state of *Things* and *Roles*, a description of their capabilities and additional runtime context information. For stationary and known devices, direct service allocation should be preferred as their performance is independent of the size of the knowledge base. As our approach is proposed as an extension to existing workflow meta-models, workflow modelers can combine static binding of resources with dynamic IoT-service discovery and invocation. Additionally, our approach allows for context-sensitive resource allocation by including context parameters in the SAL knowledge base and specifying restrictions based on the current context in the goal-based activities or directly in the SPARQL queries.

8.4 Summary

In this chapter, we provided a qualitative and quantitative evaluation of our approach. First, we deducted the evaluation approach from the targeted requirements presented in Chapter 3. Then, we defined seven test cases to demonstrate and evaluate if and how these requirements are fulfilled by our approach. We described the execution of each test case in detail and concluded that the targeted requirements are fulfilled by design. Furthermore, we conducted a performance evaluation of the semantic query-based resource discovery and invocation. In conclusion, our approach can support near real-time resource discovery in some application scenarios with lesser than 10^6 *Things*.

9 Discussion

In this chapter, we discuss the contributions and results of the thesis considering the research question RQ1-RQ5. In addition, we illustrate the extendability and limitations of the proposed solution within the field of IoT-aware workflow management.

9.1 Comparison of Solution to Research Questions

In the following, we revisit the research questions to provide the respective answers regarding our developed contributions:

- **RQ1** What are the differences between workflow resources in the cyber and the physical world from the perspective of a WfMS?

  Answer: In Section 2.1.1 we presented the general features of *Things*. The distinguishing feature of a *Thing* in the IoT is the addition of sensors, actuators, and tags to real-world objects as well as their identification by means of a unique identifier (UID). In the context of IoT-aware workflow management, the main difference are the context-sensitive capabilities of *Things* as well as their ability to sense and influence the real-world context in contrast to static service methods and virtual capabilities of regular services.

- **RQ2** How to model context-sensitive capabilities of workflow resources in the IoT?

  Answer: Within this thesis, we provide means to model and facilitate the context-sensitive capabilities of *Things* within a role-based ontology. Here, roles represent non-exclusive subsets of capabilities provided by a *Thing*. In addition, we modeled the physical context and physical entities in relation to these roles, *Things*, and capabilities. Therefore, we can express the context-sensitive relations for the activation and deactivation of roles. The context restriction can include physical context parameters as well as constraints on relations within the ontology, e. g., role implication.

- **RQ3** How can a WfMS cope with the uncertainty of IoT resource availability during workflow execution?

  Answer: In general, a model-based approach coupled with a late binding mechanism can implement a dynamic resource discovery to bridge the uncertainty in availability by the ad-hoc discovery of *Things*. We propose a solution to use an ontology for bridging the heterogeneity of *Things* and providing a foundation for a model-based ad-hoc resource discovery in the IoT. Furthermore, on the workflow modeling perspective, we provide an extended workflow metamodel to allow for specialized activity types. These provide means for the specification of resource requirements and constraints based on semantic queries or goals from a Tropos goal model. Therefore, we decouple the resource from the workflow model and allow a middleware to cope with the uncertainty of resource availability at runtime.

- **RQ4** How to minimize failures during workflow execution caused by unavailable resources?

  Answer: In this work, we use SWRL rules to specify an update mechanism to reflect the real-world state as well as the virtual system state within a coherent model. Therefore, we allow the proposed middleware to detect context mismatches in the ontology and trigger a re-allocation of the according to resources. This leads to an eventually consistent state of context-representation and resource allocation during workflow execution. However, it is the responsibility of the WfMS to cope with a changing resource in the ongoing execution of a workflow.

- **RQ5** How to allow for flexible as well as correct workflow execution in the IoT and what are the tradeoffs?

  Answer: With our work, we allow for a mixed specification of highly flexible activities as well as activities using static resources inside a workflow model. The most flexibility is allowed by the specification of a goal-based activity. Here, the goal reasoning usually provides several sets of leaf goals for fulfilling a high-level user goal. In addition, QoS parameters can contribute to the prioritization of these leaf goals. The ordered set of leaf goals each represent a feasible solution for the resource discovery problem. In sequentially translating and executing the goals into semantic queries and their execution on the knowledge base, the SAL is likely to find a fitting resource for the activity. If the workflow modeler wants to limit the scope of the resource discovery, either a specification of a leaf goal or the direct definition of SPARQL queries within activities is supported. In general, when increasing the flexibility for activity goals and goal to resource selection, the intended system behaviour is challenged by ambiguity of discovered resources and goals. Therefore, the correct or intended behavior of the workflow and each activity highly depends on a careful modeling of the Tropos goal model and the respective mapping descriptions.

9.2 Extendability of the Solutions

In the following, we discuss the extension and replacement of solutions presented in this thesis.

- **Workflow Modeling Language:** Even though, we proposed a workflow metamodel extension the extension of our approach to other workflow modeling languages and WfMS is supported to a certain degree. In general, any workflow modeling language able to specify REST service calls within activities can be coupled with the SAL to provide a context-sensitive resource discovery and invocation. However, the goal or semantic query has to be included in a service call parameter. Furthermore, the invocation of actuators is possible, while the retrieval of sensor data has to be supported by the WfMS as it is an asynchronous data fetch. Even though, we only considered imperative workflow models in this work; our approach is compatible with declarative workflow models as well. As our approach is activity-centric and independent of control-flow adaptation, a declarative workflow model can include goal-based activities.

- **Ontology:** As we already showed in the performance evaluation in Section 8.3, the SAL can easily integrate any OWL ontology as knowledge base. However, this entails the respective re-modeling of the mapping descriptions and semantic query-based activities. Furthermore, the ontology itself can be extended, e. g., with subtypes for roles, *Things* and *PhysicalEntities*. For example, it is possible to differentiate between regular objects and persons as subtypes of the class *PhysicalEntity*. These subtypes can be used in the semantic queries to provide further query specializations.

- **IoT services:** Even though, we used local services connected via the OpenHAB platform for the evaluation scenarios, any IoT service can be integrated with our approach by referencing the respective service methods within the *Commands* of the ontology.

9.3 Limitations

In the following, we present some of the limitations of this work regarding the fulfillment of the targeted requirements.

- **Large-scale:** As mentioned in the performance evaluation in Section 8.3, our approach can provide near real-time performance of resource discovery within limited scale application scenarios.
- **Quality of Context:** The update mechanism for the context-sensitive parts in the ontology is realized by continuously evaluating SWRL rules. However, the context changes themselves can only be implemented by these rules to a certain degree. For example, it is possible to use the up-to-date sensor data of *Things* and specify rules evaluating the states to trigger context changes. By evaluating data properties, rules can specify to detect the taking off of a smartwatch based on some accelerometer data. However, this approach puts a high processing demand on the knowledge base. Therefore, we recommend to use an external context service for the continuous evaluation of such data and the updating of ontology axioms via an interface.
- **Flexibility support:** By using a late binding mechanism based on resource requirements and constraints, we used an abstraction that introduces several problems within the data-flow perspective of workflow modeling. First, the input and output data ports need be filled with data from the activity. In the case of a *GetDataCommand*, this is possible and also implemented in the PROtEUS WfMS, but depending on the selected resource, the produced data may not fit the assumed data type of the subsequent consuming data port. Therefore, our approach is limited considering the data-flow within workflows.

10 Summary and Future Work

In the final chapter [1], we present a summary of this thesis and provide an outlook on future work in the field of IoT-aware workflow management.

10.1 Summary of the Thesis

Workflow resource allocation in the Internet of Things raises new challenges with respect to workflow modeling and execution due to the dynamic nature and varying availability of devices and services as well as the context-sensitive capabilities of *Things*. In this thesis, we aimed at implementing Mark Weiser's vision of Ubiquitous Computing using IoT technology to specify dynamic and context-sensitive workflow. Therefore, we targeted the fulfillment of user goals on the activity level of a workflow while adapting to context-sensitive IoT resources at runtime. We conducted our research along the outline presented in the following:

- **Requirements Analysis:** We conducted a requirements analysis to extract requirements for enabling IoT-aware workflow management from related work. Based on the requirements, we developed a classification scheme for evaluating existing approaches for their quantified fulfillment of the requirements.
- **Approach:** We used the identified gaps in research to define the aims of this thesis. The main goals of this work are the support for the **context-aware** modeling of *Things* as workflow resources, the **modeling support** within workflows to specify the requirements and constraints for resources on a per activity basis, the **query support** within these workflow activities, a **resolution strategy** to mitigate the ambiguity in resource discovery, and the improvement of the **transparency** for the workflow modeler. Even though it is already supported by related work, we aimed to fulfill the requirement for the support of the **dynamics** and general **flexibility** in the IoT. Based on the targeted requirements and their level of fulfillment, we developed our

---

[1] This chapter is partially based on [HSK$^+$16].

approach for enabling the integration of context-sensitive *Things* as workflow resources.

- **Contributions:** Within this work, we provided the following contributions: A **role-based** resource and context **ontology** for *Things* in the IoT. A workflow **metamodel** extension to enable the modeling of resource requirements and constraints. A **goal model** for high-level user goals and a **middleware** for dynamic resource discovery and invocation in the IoT. The role-based ontology for the modeling of *Things* includes SWRL rules as an update mechanism to represent the current state of real-world and virtual context in one coherent model. In the ontology, *Things* can play *Roles* with capabilities wich in turn can be activated or deactivated with the help of context constraints. The workflow metamodel extension enables the modeling of semantic query-based as well as goal-based activities in special activity types. The Tropos goal model provides means to specify the relation between high-level user goals and executable leaf goals, annotated with mapping descriptions for their translation into semantic queries. Furthermore, the goal model can express additional QoS parameters and the contribution of goals towards these parameters. Finally, the Semantic Access Layer (SAL) middleware is designed to mediate between a WfMS and highly dynamic resources in the IoT using the ontology and the goal model.

- **Evaluation:** Based on the targeted requirements defined in the requirements analysis of this work, we developed seven test cases to demonstrate and evaluate their fulfillment. Using a proof-of-concept implementation in the Smart Home domain, we conducted the execution of these test cases in addition to performance tests regarding the query execution times. The results show that we reached the targeted requirements by design and the resource discovery and invocation scales nearly linear up 10^6 instances representing *Things* in the ontology. Furthermore, we identified the problem that there is a significant decrease of query performance when sensor values have to be retrieved from a vast amount of IoT services.

In conclusion, this thesis contributes to enabling the vision of Mark Weiser by allowing the specification of Ubiquitous System behavior in the form of repeatable workflows that are able to cope with the dynamics and context-sensitive capabilities of *Things* in the IoT.

10.2 Future Work

Regarding future work, we will further investigate the data-flow perspective such that dynamically adapted activities can still produce and consume data both syntactically and semantically correct. The data-flow also has an impact on the definition of the goal model and the command types in the ontology. In addition, to reduce the modeling overhead of our approach, we will investigate the partial generation of the different models. We see an opportunity to generate the goal model and mapping descriptions from the domain-specific part of the ontology such that the modeling effort is significantly reduced. With respect to the query performance, we will extend the SAL to store the sensor data in separation to the ontology and to execute the sensor data retrieval on the separate database. Therefore, the *FILTER* part of SPARQL queries need to be extracted and translated into another query such that both queries can be executed within their respective databases and the results from the semantic query can be with the results of the external query. This will dramatically increase the retrieval of sensor data, as ontological knowledge bases are neither designed to store large amounts of data nor to update such data within frequent write operations.

Appendix

List of Figures

List of Tables

List of Listings

List of Abbreviations

6LoWPAN IPv6 over Low power Wireless Personal Area Network.

AAL Ambient Assisted Living.

ABox Assertion component.

BPM Business Process Management.

BPMN Business Process Model and Notation.

CoAP Constrained Application Protocol.

CpC Cyber-physical Consistency.

CPS Cyber-physical Systems.

ECA Event-Condition-Action.

EPC Event-driven Process Chain.

IoT Internet of Things.

IPSO Internet Protocol for Smart Objects.

IPv6 Internet Protocol Version 6.

MDA Model-driven Architecture.

MOF Meta Object Facility.

NFC Near Field Communication.

OSGi Open Services Gateway initiative.

OSI model Open Systems Interconnection model.

OWL Web Ontology Langauge.

PROtEUS Process Execution System for Cyber-Physical Systems.

QoC Quality of Context.

QoS Quality of Service.

RDF Resource Description Framework.

RDFS Resource Description Framework Schema.

RFID Radio Frequency Identification.

SAL Semantic Access Layer.

SOA Service-oriented Architecture.

SPARQL SPARQL Protocol and RDF Query Language.

SWRL Semantic Web Rule Language.

TBox Terminological component.

TCPN Timed Colored Petri Nets.

UID Unique Identifier.

UML Unified Modeling Language.

WADL Web Application Description Language.

WfMC Workflow Management Coalition.

WfMS Workflow Management System.

WS-BPEL Web Services Business Process Execution Language.

WSDL Web Services Description Language.

WSN Wireless Sensor Network.

XML Extensible Markup Language.

YAWL Yet Another Workflow Language.

Example Semantic Context Model for IoT-Things

```
<?xml version=”1.0”?>
<!DOCTYPE Ontology [
    <!ENTITY xsd ”http://www.w3.org/2001/XMLSchema#” >
    <!ENTITY xml ”http://www.w3.org/XML/1998/namespace” >
    <!ENTITY rdfs ”http://www.w3.org/2000/01/rdf−schema#” >
    <!ENTITY rdf ”http://www.w3.org/1999/02/22−rdf−syntax−ns#” >
]>
<Ontology xmlns=”http://www.w3.org/2002/07/owl#”
     xml:base=”http://www.semanticweb.org/role_based_iot_ontology_inferred”
     xmlns:rdf=”http://www.w3.org/1999/02/22−rdf−syntax−ns#”
     xmlns:xml=”http://www.w3.org/XML/1998/namespace”
     xmlns:xsd=”http://www.w3.org/2001/XMLSchema#”
     xmlns:rdfs=”http://www.w3.org/2000/01/rdf−schema#”
     ontologyIRI=”http://www.semanticweb.org/role_based_iot_ontology_inferred”>
    <Prefix name=”rdf” IRI=”http://www.w3.org/1999/02/22−rdf−syntax−ns#”/>
    <Prefix name=”rdfs” IRI=”http://www.w3.org/2000/01/rdf−schema#”/>
    <Prefix name=”xsd” IRI=”http://www.w3.org/2001/XMLSchema#”/>
    <Prefix name=”owl” IRI=”http://www.w3.org/2002/07/owl#”/>
    <SubClassOf>
        <Class IRI=”http://www.semanticweb.org/role_based_iot_ontology#Actuator”/
            >
        <ObjectSomeValuesFrom>
            <ObjectProperty IRI=”http://www.semanticweb.org/role_based_iot_ontology
                #manipulate”/>
            <Class IRI=”http://www.semanticweb.org/role_based_iot_ontology#
                PhysicalObject”/>
        </ObjectSomeValuesFrom>
    </SubClassOf>
    <SubClassOf>
        <Class IRI=”http://www.semanticweb.org/role_based_iot_ontology#Actuator”/
            >
        <ObjectSomeValuesFrom>
            <ObjectProperty IRI=”http://www.semanticweb.org/role_based_iot_ontology
                #manipulate”/>
            <Class IRI=”http://www.semanticweb.org/role_based_iot_ontology#
                PhysicalProperty”/>
        </ObjectSomeValuesFrom>
    </SubClassOf>
    <SubClassOf>
        <Class IRI=”http://www.semanticweb.org/role_based_iot_ontology#IoT_Thing”/
            >
        <ObjectSomeValuesFrom>
            <ObjectProperty IRI=”http://www.semanticweb.org/role_based_iot_ontology
                #has”/>
            <Class IRI=”http://www.semanticweb.org/role_based_iot_ontology#
                Actuator”/>
        </ObjectSomeValuesFrom>
    </SubClassOf>
    <SubClassOf>
        <Class IRI=”http://www.semanticweb.org/role_based_iot_ontology#IoT_Thing”/
            >
        <ObjectSomeValuesFrom>
            <ObjectProperty IRI=”http://www.semanticweb.org/role_based_iot_ontology
                #has”/>
            <Class IRI=”http://www.semanticweb.org/role_based_iot_ontology#Sensor”
                />
        </ObjectSomeValuesFrom>
    </SubClassOf>
    <SubClassOf>
        <Class IRI=”http://www.semanticweb.org/role_based_iot_ontology#
            PhysicalContext”/>
        <ObjectSomeValuesFrom>
            <ObjectProperty IRI=”http://www.semanticweb.org/role_based_iot_ontology
                #has”/>
            <Class IRI=”http://www.semanticweb.org/role_based_iot_ontology#
                PhysicalObject”/>
        </ObjectSomeValuesFrom>
    </SubClassOf>
    <SubClassOf>
        <Class IRI=”http://www.semanticweb.org/role_based_iot_ontology#
            PhysicalObject”/>
        <ObjectSomeValuesFrom>
            <ObjectProperty IRI=”http://www.semanticweb.org/role_based_iot_ontology
                #contain”/>
            <Class IRI=”http://www.semanticweb.org/role_based_iot_ontology#
                PhysicalObject”/>
        </ObjectSomeValuesFrom>
    </SubClassOf>
    <SubClassOf>
        <Class IRI=”http://www.semanticweb.org/role_based_iot_ontology#
            PhysicalObject”/>
        <ObjectSomeValuesFrom>
            <ObjectProperty IRI=”http://www.semanticweb.org/role_based_iot_ontology
            #manipulate”/>
        <Class IRI=”http://www.semanticweb.org/role_based_iot_ontology#
            PhysicalObject”/>
    </ObjectSomeValuesFrom>
</SubClassOf>
<SubClassOf>
    <Class IRI=”http://www.semanticweb.org/role_based_iot_ontology#
        PhysicalObject”/>
    <ObjectSomeValuesFrom>
        <ObjectProperty IRI=”http://www.semanticweb.org/role_based_iot_ontology
            #manipulate”/>
        <Class IRI=”http://www.semanticweb.org/role_based_iot_ontology#
            PhysicalProperty”/>
    </ObjectSomeValuesFrom>
</SubClassOf>
<SubClassOf>
    <Class IRI=”http://www.semanticweb.org/role_based_iot_ontology#
        PhysicalObject”/>
    <ObjectMinCardinality cardinality=”1”>
        <ObjectProperty IRI=”http://www.semanticweb.org/role_based_iot_ontology
            #has”/>
        <Class IRI=”http://www.semanticweb.org/role_based_iot_ontology#
            PhysicalProperty”/>
    </ObjectMinCardinality>
</SubClassOf>
<SubClassOf>
    <Class IRI=”http://www.semanticweb.org/role_based_iot_ontology#Sensor”/>
    <ObjectSomeValuesFrom>
        <ObjectProperty IRI=”http://www.semanticweb.org/role_based_iot_ontology
            #sense”/>
        <Class IRI=”http://www.semanticweb.org/role_based_iot_ontology#
            PhysicalProperty”/>
    </ObjectSomeValuesFrom>
</SubClassOf>
<DisjointClasses>
    <Class IRI=”http://www.semanticweb.org/role_based_iot_ontology#Actuator”/
        >
    <Class IRI=”http://www.semanticweb.org/role_based_iot_ontology#Sensor”/>
</DisjointClasses>
<ClassAssertion>
    <Class IRI=”http://www.semanticweb.org/role_based_iot_ontology#Sensor”/>
    <NamedIndividual IRI=”http://www.semanticweb.org/role_based_iot_ontology#
```

```
            AmbientLightSensor”/>
    </ClassAssertion>
    <ClassAssertion>
        <Class IRI=”http://www.semanticweb.org/role_based_iot_ontology#
            PhysicalObject”/>
        <NamedIndividual IRI=”http://www.semanticweb.org/role_based_iot_ontology#
            Curtain”/>
    </ClassAssertion>
    <ClassAssertion>
        <Class IRI=”http://www.semanticweb.org/role_based_iot_ontology#Actuator”/
            >
        <NamedIndividual IRI=”http://www.semanticweb.org/role_based_iot_ontology#
            CurtainActuator”/>
    </ClassAssertion>
    <ClassAssertion>
        <Class IRI=”http://www.semanticweb.org/role_based_iot_ontology#
            PhysicalProperty”/>
        <NamedIndividual IRI=”http://www.semanticweb.org/role_based_iot_ontology#
            Illumination”/>
    </ClassAssertion>
    <ClassAssertion>
        <Class IRI=”http://www.semanticweb.org/role_based_iot_ontology#
            PhysicalObject”/>
        <NamedIndividual IRI=”http://www.semanticweb.org/role_based_iot_ontology#
            Room”/>
    </ClassAssertion>
    <ClassAssertion>
        <Class IRI=”http://www.semanticweb.org/role_based_iot_ontology#IoT_Thing”/
            >
        <NamedIndividual IRI=”http://www.semanticweb.org/role_based_iot_ontology#
            SmartCurtain”/>
    </ClassAssertion>
    <ClassAssertion>
        <Class IRI=”http://www.semanticweb.org/role_based_iot_ontology#
            PhysicalContext”/>
        <NamedIndividual IRI=”http://www.semanticweb.org/role_based_iot_ontology#
            SmartHome”/>
    </ClassAssertion>
    <ObjectPropertyAssertion>
        <ObjectProperty IRI=”http://www.semanticweb.org/role_based_iot_ontology#
            sense”/>
        <NamedIndividual IRI=”http://www.semanticweb.org/role_based_iot_ontology#
        AmbientLightSensor”/>
    <NamedIndividual IRI=”http://www.semanticweb.org/role_based_iot_ontology#
        Illumination”/>
</ObjectPropertyAssertion>
<ObjectPropertyAssertion>
    <ObjectProperty IRI=”http://www.semanticweb.org/role_based_iot_ontology#
        manipulate”/>
    <NamedIndividual IRI=”http://www.semanticweb.org/role_based_iot_ontology#
        Curtain”/>
    <NamedIndividual IRI=”http://www.semanticweb.org/role_based_iot_ontology#
        Illumination”/>
</ObjectPropertyAssertion>
<ObjectPropertyAssertion>
    <ObjectProperty IRI=”http://www.semanticweb.org/role_based_iot_ontology#
        manipulate”/>
    <NamedIndividual IRI=”http://www.semanticweb.org/role_based_iot_ontology#
        CurtainActuator”/>
    <NamedIndividual IRI=”http://www.semanticweb.org/role_based_iot_ontology#
        Illumination”/>
</ObjectPropertyAssertion>
<ObjectPropertyAssertion>
    <ObjectProperty IRI=”http://www.semanticweb.org/role_based_iot_ontology#
        manipulate”/>
    <NamedIndividual IRI=”http://www.semanticweb.org/role_based_iot_ontology#
        CurtainActuator”/>
    <NamedIndividual IRI=”http://www.semanticweb.org/role_based_iot_ontology#
        Curtain”/>
</ObjectPropertyAssertion>
<ObjectPropertyAssertion>
    <ObjectProperty IRI=”http://www.semanticweb.org/role_based_iot_ontology#
        contain”/>
    <NamedIndividual IRI=”http://www.semanticweb.org/role_based_iot_ontology#
        Room”/>
    <NamedIndividual IRI=”http://www.semanticweb.org/role_based_iot_ontology#
        Curtain”/>
</ObjectPropertyAssertion>
<ObjectPropertyAssertion>
    <ObjectProperty IRI=”http://www.semanticweb.org/role_based_iot_ontology#
        has”/>
    <NamedIndividual IRI=”http://www.semanticweb.org/role_based_iot_ontology#
        Room”/>
    <NamedIndividual IRI=”http://www.semanticweb.org/role_based_iot_ontology#
```

```
          Illumination”/>
  </ObjectPropertyAssertion>
  <ObjectPropertyAssertion>
      <ObjectProperty IRI=”http://www.semanticweb.org/role_based_iot_ontology#
          has”/>
      <NamedIndividual IRI=”http://www.semanticweb.org/role_based_iot_ontology#
          SmartCurtain”/>
      <NamedIndividual IRI=”http://www.semanticweb.org/role_based_iot_ontology#
          AmbientLightSensor”/>
  </ObjectPropertyAssertion>
  <ObjectPropertyAssertion>
      <ObjectProperty IRI=”http://www.semanticweb.org/role_based_iot_ontology#
          has”/>
      <NamedIndividual IRI=”http://www.semanticweb.org/role_based_iot_ontology#
          SmartCurtain”/>
      <NamedIndividual IRI=”http://www.semanticweb.org/role_based_iot_ontology#
          CurtainActuator”/>
  </ObjectPropertyAssertion>
  <ObjectPropertyAssertion>
      <ObjectProperty IRI=”http://www.semanticweb.org/role_based_iot_ontology#
          has”/>
      <NamedIndividual IRI=”http://www.semanticweb.org/role_based_iot_ontology#
          SmartHome”/>
      <NamedIndividual IRI=”http://www.semanticweb.org/role_based_iot_ontology#
          Room”/>
  </ObjectPropertyAssertion>
  <EquivalentObjectProperties>
      <ObjectProperty IRI=”http://www.semanticweb.org/role_based_iot_ontology#
          contain”/>
  </EquivalentObjectProperties>
  <EquivalentObjectProperties>
      <ObjectProperty IRI=”http://www.semanticweb.org/role_based_iot_ontology#
          has”/>
  </EquivalentObjectProperties>
  <EquivalentObjectProperties>
      <ObjectProperty IRI=”http://www.semanticweb.org/role_based_iot_ontology#
          manipulate”/>
  </EquivalentObjectProperties>
  <EquivalentObjectProperties>
      <ObjectProperty IRI=”http://www.semanticweb.org/role_based_iot_ontology#
          sense”/>
  </EquivalentObjectProperties>
```

```
    <EquivalentObjectProperties>
        <ObjectProperty abbreviatedIRI=”owl:topObjectProperty”/>
    </EquivalentObjectProperties>
    <SubObjectPropertyOf>
        <ObjectProperty IRI=”http://www.semanticweb.org/role_based_iot_ontology#
            manipulate”/>
        <ObjectProperty abbreviatedIRI=”owl:topObjectProperty”/>
    </SubObjectPropertyOf>
    <SubObjectPropertyOf>
        <ObjectProperty IRI=”http://www.semanticweb.org/role_based_iot_ontology#
            sense”/>
        <ObjectProperty abbreviatedIRI=”owl:topObjectProperty”/>
    </SubObjectPropertyOf>
    <InverseObjectProperties>
        <ObjectProperty abbreviatedIRI=”owl:topObjectProperty”/>
        <ObjectProperty abbreviatedIRI=”owl:topObjectProperty”/>
    </InverseObjectProperties>
    <SymmetricObjectProperty>
        <ObjectProperty abbreviatedIRI=”owl:topObjectProperty”/>
    </SymmetricObjectProperty>
    <TransitiveObjectProperty>
        <ObjectProperty IRI=”http://www.semanticweb.org/role_based_iot_ontology#
            contain”/>
    </TransitiveObjectProperty>
    <TransitiveObjectProperty>
        <ObjectProperty IRI=”http://www.semanticweb.org/role_based_iot_ontology#
            manipulate”/>
    </TransitiveObjectProperty>
</Ontology>
```

T-Box of Ontology for Role-based Things in the IoT

```
<?xml version=”1.0”?>
<Ontology xmlns=”http://www.w3.org/2002/07/owl#”
      xml:base=”http://www.semanticweb.org/IoTUDresden”
      xmlns:rdf=”http://www.w3.org/1999/02/22−rdf−syntax−ns#”
      xmlns:xml=”http://www.w3.org/XML/1998/namespace”
      xmlns:xsd=”http://www.w3.org/2001/XMLSchema#”
      xmlns:rdfs=”http://www.w3.org/2000/01/rdf−schema#”
      ontologyIRI=”http://www.semanticweb.org/IoTUDresden”>
     <Prefix name=”owl” IRI=”http://www.w3.org/2002/07/owl#”/>
     <Prefix name=”rdf” IRI=”http://www.w3.org/1999/02/22−rdf−syntax−ns#”/>
     <Prefix name=”xml” IRI=”http://www.w3.org/XML/1998/namespace”/>
     <Prefix name=”xsd” IRI=”http://www.w3.org/2001/XMLSchema#”/>
     <Prefix name=”rdfs” IRI=”http://www.w3.org/2000/01/rdf−schema#”/>
     <Declaration>
          <ObjectProperty IRI=”#canBePlayedFor”/>
     </Declaration>
     <Declaration>
          <Class IRI=”#Tag”/>
     </Declaration>
     <Declaration>
          <Class IRI=”#Device”/>
     </Declaration>
     <Declaration>
          <Class IRI=”#Sensor”/>
     </Declaration>
     <Declaration>
          <Class IRI=”#PhysicalContext”/>
     </Declaration>
     <Declaration>
          <ObjectProperty IRI=”#canPlay”/>
     </Declaration>
     <Declaration>
          <ObjectProperty IRI=”#hasCapability”/>
     </Declaration>
     <Declaration>
          <Class IRI=”#PhysicalEntity”/>
     </Declaration>
     <Declaration>
          <ObjectProperty IRI=”#contains”/>
     </Declaration>
     <Declaration>
```

```
    <ObjectProperty IRI=”#plays”/>
</Declaration>
<Declaration>
    <ObjectProperty IRI=”#isAttachedTo”/>
</Declaration>
<Declaration>
    <ObjectProperty IRI=”#hasEntity”/>
</Declaration>
<Declaration>
    <ObjectProperty IRI=”#affects”/>
</Declaration>
<Declaration>
    <ObjectProperty IRI=”#reads”/>
</Declaration>
<Declaration>
    <ObjectProperty IRI=”#usesDevice”/>
</Declaration>
<Declaration>
    <Class IRI=”#Capability”/>
</Declaration>
<Declaration>
    <Class IRI=”#PhysicalProperty”/>
</Declaration>
<Declaration>
    <Class IRI=”#OnCommand”/>
</Declaration>
<Declaration>
    <Class IRI=”#Thing”/>
</Declaration>
<Declaration>
    <ObjectProperty IRI=”#isAvailableIn”/>
</Declaration>
<Declaration>
    <ObjectProperty IRI=”#hasDevice”/>
</Declaration>
<Declaration>
    <Class IRI=”#GetDataCommand”/>
</Declaration>
<Declaration>
    <ObjectProperty IRI=”#senses”/>
</Declaration>
<Declaration>
```

```
    <ObjectProperty IRI=”#monitors”/>
</Declaration>
<Declaration>
    <ObjectProperty IRI=”#isAllocatedTo”/>
</Declaration>
<Declaration>
    <Class IRI=”#Command”/>
</Declaration>
<Declaration>
    <ObjectProperty IRI=”#isPlayedFor”/>
</Declaration>
<Declaration>
    <Class IRI=”#Role”/>
</Declaration>
<Declaration>
    <ObjectProperty IRI=”#identifies”/>
</Declaration>
<Declaration>
    <ObjectProperty IRI=”#hasProperty”/>
</Declaration>
<Declaration>
    <Class IRI=”#Actuator”/>
</Declaration>
<Declaration>
    <DataProperty IRI=”#serviceMethodURL”/>
</Declaration>
<Declaration>
    <AnnotationProperty IRI=”http://swrl.stanford.edu/ontologies/3.3/swrla.owl#
        isRuleEnabled”/>
</Declaration>
<Declaration>
    <ObjectProperty IRI=”#manipulates”/>
</Declaration>
<Declaration>
    <ObjectProperty IRI=”#hasCommand”/>
</Declaration>
<Declaration>
    <ObjectProperty IRI=”#accessesDevice”/>
</Declaration>
<Declaration>
    <DataProperty IRI=”#physicalPropertyData”/>
</Declaration>
```

```
<Declaration>
    <Class IRI=”#OffCommand”/>
</Declaration>
<SubClassOf>
    <Class IRI=”#Actuator”/>
    <Class IRI=”#Device”/>
</SubClassOf>
<SubClassOf>
    <Class IRI=”#Actuator”/>
    <ObjectSomeValuesFrom>
        <ObjectProperty IRI=”#affects”/>
        <Class IRI=”#Sensor”/>
    </ObjectSomeValuesFrom>
</SubClassOf>
<SubClassOf>
    <Class IRI=”#Actuator”/>
    <ObjectSomeValuesFrom>
        <ObjectProperty IRI=”#isAttachedTo”/>
        <Class IRI=”#PhysicalEntity”/>
    </ObjectSomeValuesFrom>
</SubClassOf>
<SubClassOf>
    <Class IRI=”#Actuator”/>
    <ObjectSomeValuesFrom>
        <ObjectProperty IRI=”#manipulates”/>
        <Class IRI=”#PhysicalEntity”/>
    </ObjectSomeValuesFrom>
</SubClassOf>
<SubClassOf>
    <Class IRI=”#Actuator”/>
    <ObjectSomeValuesFrom>
        <ObjectProperty IRI=”#manipulates”/>
        <Class IRI=”#PhysicalProperty”/>
    </ObjectSomeValuesFrom>
</SubClassOf>
<SubClassOf>
    <Class IRI=”#Capability”/>
    <ObjectSomeValuesFrom>
        <ObjectProperty IRI=”#hasCommand”/>
        <Class IRI=”#Command”/>
    </ObjectSomeValuesFrom>
</SubClassOf>
```

```
<SubClassOf>
    <Class IRI=”#Capability”/>
    <ObjectSomeValuesFrom>
        <ObjectProperty IRI=”#usesDevice”/>
        <Class IRI=”#Device”/>
    </ObjectSomeValuesFrom>
</SubClassOf>
<SubClassOf>
    <Class IRI=”#Command”/>
    <ObjectSomeValuesFrom>
        <ObjectProperty IRI=”#accessesDevice”/>
        <Class IRI=”#Device”/>
    </ObjectSomeValuesFrom>
</SubClassOf>
<SubClassOf>
    <Class IRI=”#Command”/>
    <DataSomeValuesFrom>
        <DataProperty IRI=”#serviceMethodURL”/>
        <Datatype abbreviatedIRI=”xsd:anyURI”/>
    </DataSomeValuesFrom>
</SubClassOf>
<SubClassOf>
    <Class IRI=”#GetDataCommand”/>
    <Class IRI=”#Command”/>
</SubClassOf>
<SubClassOf>
    <Class IRI=”#GetDataCommand”/>
    <ObjectSomeValuesFrom>
        <ObjectProperty IRI=”#accessesDevice”/>
        <Class IRI=”#Device”/>
    </ObjectSomeValuesFrom>
</SubClassOf>
<SubClassOf>
    <Class IRI=”#OffCommand”/>
    <Class IRI=”#Command”/>
</SubClassOf>
<SubClassOf>
    <Class IRI=”#OffCommand”/>
    <ObjectSomeValuesFrom>
        <ObjectProperty IRI=”#accessesDevice”/>
        <Class IRI=”#Device”/>
    </ObjectSomeValuesFrom>
```

```
</SubClassOf>
<SubClassOf>
    <Class IRI=”#OnCommand”/>
    <Class IRI=”#Command”/>
</SubClassOf>
<SubClassOf>
    <Class IRI=”#OnCommand”/>
    <ObjectSomeValuesFrom>
        <ObjectProperty IRI=”#accessesDevice”/>
        <Class IRI=”#Device”/>
    </ObjectSomeValuesFrom>
</SubClassOf>
<SubClassOf>
    <Class IRI=”#PhysicalContext”/>
    <ObjectSomeValuesFrom>
        <ObjectProperty IRI=”#hasEntity”/>
        <Class IRI=”#PhysicalEntity”/>
    </ObjectSomeValuesFrom>
</SubClassOf>
<SubClassOf>
    <Class IRI=”#PhysicalEntity”/>
    <ObjectUnionOf>
        <Class IRI=”#Actuator”/>
        <Class IRI=”#PhysicalEntity”/>
    </ObjectUnionOf>
</SubClassOf>
<SubClassOf>
    <Class IRI=”#PhysicalEntity”/>
    <ObjectSomeValuesFrom>
        <ObjectProperty IRI=”#contains”/>
        <Class IRI=”#PhysicalEntity”/>
    </ObjectSomeValuesFrom>
</SubClassOf>
<SubClassOf>
    <Class IRI=”#PhysicalEntity”/>
    <ObjectSomeValuesFrom>
        <ObjectProperty IRI=”#hasProperty”/>
        <Class IRI=”#PhysicalProperty”/>
    </ObjectSomeValuesFrom>
</SubClassOf>
<SubClassOf>
    <Class IRI=”#PhysicalEntity”/>
    <ObjectSomeValuesFrom>
        <ObjectProperty IRI=”#manipulates”/>
        <Class IRI=”#PhysicalProperty”/>
    </ObjectSomeValuesFrom>
</SubClassOf>
<SubClassOf>
    <Class IRI=”#PhysicalProperty”/>
    <ObjectUnionOf>
        <Class IRI=”#PhysicalEntity”/>
        <Class IRI=”#PhysicalProperty”/>
    </ObjectUnionOf>
</SubClassOf>
<SubClassOf>
    <Class IRI=”#Role”/>
    <ObjectSomeValuesFrom>
        <ObjectProperty IRI=”#hasCapability”/>
        <Class IRI=”#Capability”/>
    </ObjectSomeValuesFrom>
</SubClassOf>
<SubClassOf>
    <Class IRI=”#Sensor”/>
    <Class IRI=”#Device”/>
</SubClassOf>
<SubClassOf>
    <Class IRI=”#Sensor”/>
    <ObjectSomeValuesFrom>
        <ObjectProperty IRI=”#isAttachedTo”/>
        <Class IRI=”#PhysicalEntity”/>
    </ObjectSomeValuesFrom>
</SubClassOf>
<SubClassOf>
    <Class IRI=”#Sensor”/>
    <ObjectSomeValuesFrom>
        <ObjectProperty IRI=”#monitors”/>
        <Class IRI=”#Actuator”/>
    </ObjectSomeValuesFrom>
</SubClassOf>
<SubClassOf>
    <Class IRI=”#Sensor”/>
    <ObjectSomeValuesFrom>
        <ObjectProperty IRI=”#reads”/>
        <Class IRI=”#Tag”/>
    </ObjectSomeValuesFrom>
</SubClassOf>
<SubClassOf>
    <Class IRI=”#Sensor”/>
    <ObjectSomeValuesFrom>
        <ObjectProperty IRI=”#senses”/>
        <Class IRI=”#PhysicalProperty”/>
    </ObjectSomeValuesFrom>
</SubClassOf>
<SubClassOf>
    <Class IRI=”#Tag”/>
    <Class IRI=”#Device”/>
</SubClassOf>
<SubClassOf>
    <Class IRI=”#Tag”/>
    <ObjectSomeValuesFrom>
        <ObjectProperty IRI=”#identifies”/>
        <Class IRI=”#PhysicalEntity”/>
    </ObjectSomeValuesFrom>
</SubClassOf>
<SubClassOf>
    <Class IRI=”#Tag”/>
    <ObjectSomeValuesFrom>
        <ObjectProperty IRI=”#isAttachedTo”/>
        <Class IRI=”#PhysicalEntity”/>
    </ObjectSomeValuesFrom>
</SubClassOf>
<SubClassOf>
    <Class IRI=”#Thing”/>
    <ObjectSomeValuesFrom>
        <ObjectProperty IRI=”#canPlay”/>
        <Class IRI=”#Role”/>
    </ObjectSomeValuesFrom>
</SubClassOf>
<SubClassOf>
    <Class IRI=”#Thing”/>
    <ObjectSomeValuesFrom>
        <ObjectProperty IRI=”#hasDevice”/>
        <Class IRI=”#Device”/>
    </ObjectSomeValuesFrom>
</SubClassOf>
<SubClassOf>
```

```
    <Class IRI=”#Thing”/>
    <ObjectSomeValuesFrom>
        <ObjectProperty IRI=”#isAttachedTo”/>
        <Class IRI=”#PhysicalEntity”/>
    </ObjectSomeValuesFrom>
</SubClassOf>
<SubClassOf>
    <Class IRI=”#Thing”/>
    <ObjectSomeValuesFrom>
        <ObjectProperty IRI=”#isAvailableIn”/>
        <Class IRI=”#PhysicalContext”/>
    </ObjectSomeValuesFrom>
</SubClassOf>
<SubClassOf>
    <Class IRI=”#Thing”/>
    <ObjectSomeValuesFrom>
        <ObjectProperty IRI=”#plays”/>
        <Class IRI=”#Role”/>
    </ObjectSomeValuesFrom>
</SubClassOf>
<SubObjectPropertyOf>
    <ObjectProperty IRI=”#accessesDevice”/>
    <ObjectProperty abbreviatedIRI=”owl:topObjectProperty”/>
</SubObjectPropertyOf>
<SubObjectPropertyOf>
    <ObjectProperty IRI=”#affects”/>
    <ObjectProperty abbreviatedIRI=”owl:topObjectProperty”/>
</SubObjectPropertyOf>
<SubObjectPropertyOf>
    <ObjectProperty IRI=”#canBePlayedFor”/>
    <ObjectProperty abbreviatedIRI=”owl:topObjectProperty”/>
</SubObjectPropertyOf>
<SubObjectPropertyOf>
    <ObjectProperty IRI=”#canPlay”/>
    <ObjectProperty abbreviatedIRI=”owl:topObjectProperty”/>
</SubObjectPropertyOf>
<SubObjectPropertyOf>
    <ObjectProperty IRI=”#contains”/>
    <ObjectProperty abbreviatedIRI=”owl:topObjectProperty”/>
</SubObjectPropertyOf>
<SubObjectPropertyOf>
    <ObjectProperty IRI=”#hasCapability”/>
    <ObjectProperty abbreviatedIRI=”owl:topObjectProperty”/>
</SubObjectPropertyOf>
<SubObjectPropertyOf>
    <ObjectProperty IRI=”#hasCommand”/>
    <ObjectProperty abbreviatedIRI=”owl:topObjectProperty”/>
</SubObjectPropertyOf>
<SubObjectPropertyOf>
    <ObjectProperty IRI=”#hasDevice”/>
    <ObjectProperty abbreviatedIRI=”owl:topObjectProperty”/>
</SubObjectPropertyOf>
<SubObjectPropertyOf>
    <ObjectProperty IRI=”#hasEntity”/>
    <ObjectProperty abbreviatedIRI=”owl:topObjectProperty”/>
</SubObjectPropertyOf>
<SubObjectPropertyOf>
    <ObjectProperty IRI=”#hasProperty”/>
    <ObjectProperty abbreviatedIRI=”owl:topObjectProperty”/>
</SubObjectPropertyOf>
<SubObjectPropertyOf>
    <ObjectProperty IRI=”#isAttachedTo”/>
    <ObjectProperty abbreviatedIRI=”owl:topObjectProperty”/>
</SubObjectPropertyOf>
<SubObjectPropertyOf>
    <ObjectProperty IRI=”#isAvailableIn”/>
    <ObjectProperty abbreviatedIRI=”owl:topObjectProperty”/>
</SubObjectPropertyOf>
<SubObjectPropertyOf>
    <ObjectProperty IRI=”#manipulates”/>
    <ObjectProperty abbreviatedIRI=”owl:topObjectProperty”/>
</SubObjectPropertyOf>
<SubObjectPropertyOf>
    <ObjectProperty IRI=”#monitors”/>
    <ObjectProperty abbreviatedIRI=”owl:topObjectProperty”/>
</SubObjectPropertyOf>
<SubObjectPropertyOf>
    <ObjectProperty IRI=”#plays”/>
    <ObjectProperty abbreviatedIRI=”owl:topObjectProperty”/>
</SubObjectPropertyOf>
<SubObjectPropertyOf>
    <ObjectProperty IRI=”#reads”/>
    <ObjectProperty abbreviatedIRI=”owl:topObjectProperty”/>
</SubObjectPropertyOf>
```

```
<SubObjectPropertyOf>
    <ObjectProperty IRI=”#senses”/>
    <ObjectProperty abbreviatedIRI=”owl:topObjectProperty”/>
</SubObjectPropertyOf>
<SubObjectPropertyOf>
    <ObjectProperty IRI=”#usesDevice”/>
    <ObjectProperty abbreviatedIRI=”owl:topObjectProperty”/>
</SubObjectPropertyOf>
<InverseObjectProperties>
    <ObjectProperty IRI=”#affects”/>
    <ObjectProperty IRI=”#monitors”/>
</InverseObjectProperties>
<TransitiveObjectProperty>
    <ObjectProperty IRI=”#contains”/>
</TransitiveObjectProperty>
<TransitiveObjectProperty>
    <ObjectProperty IRI=”#manipulates”/>
</TransitiveObjectProperty>
<ObjectPropertyDomain>
    <ObjectProperty IRI=”#accessesDevice”/>
    <Class IRI=”#Command”/>
</ObjectPropertyDomain>
<ObjectPropertyDomain>
    <ObjectProperty IRI=”#affects”/>
    <Class IRI=”#Actuator”/>
</ObjectPropertyDomain>
<ObjectPropertyDomain>
    <ObjectProperty IRI=”#canBePlayedFor”/>
    <Class IRI=”#Role”/>
</ObjectPropertyDomain>
<ObjectPropertyDomain>
    <ObjectProperty IRI=”#canPlay”/>
    <Class IRI=”#Thing”/>
</ObjectPropertyDomain>
<ObjectPropertyDomain>
    <ObjectProperty IRI=”#contains”/>
    <Class IRI=”#PhysicalEntity”/>
</ObjectPropertyDomain>
<ObjectPropertyDomain>
    <ObjectProperty IRI=”#hasCapability”/>
    <Class IRI=”#Role”/>
</ObjectPropertyDomain>
```

```
<ObjectPropertyDomain>
    <ObjectProperty IRI=”#hasCommand”/>
    <Class IRI=”#Capability”/>
</ObjectPropertyDomain>
<ObjectPropertyDomain>
    <ObjectProperty IRI=”#hasDevice”/>
    <Class IRI=”#Thing”/>
</ObjectPropertyDomain>
<ObjectPropertyDomain>
    <ObjectProperty IRI=”#hasEntity”/>
    <Class IRI=”#PhysicalContext”/>
</ObjectPropertyDomain>
<ObjectPropertyDomain>
    <ObjectProperty IRI=”#hasProperty”/>
    <Class IRI=”#PhysicalEntity”/>
</ObjectPropertyDomain>
<ObjectPropertyDomain>
    <ObjectProperty IRI=”#identifies”/>
    <Class IRI=”#Tag”/>
</ObjectPropertyDomain>
<ObjectPropertyDomain>
    <ObjectProperty IRI=”#isAllocatedTo”/>
    <Class IRI=”#Thing”/>
</ObjectPropertyDomain>
<ObjectPropertyDomain>
    <ObjectProperty IRI=”#isAttachedTo”/>
    <Class IRI=”#Thing”/>
</ObjectPropertyDomain>
<ObjectPropertyDomain>
    <ObjectProperty IRI=”#isAvailableIn”/>
    <Class IRI=”#Thing”/>
</ObjectPropertyDomain>
<ObjectPropertyDomain>
    <ObjectProperty IRI=”#isPlayedFor”/>
    <Class IRI=”#Role”/>
</ObjectPropertyDomain>
<ObjectPropertyDomain>
    <ObjectProperty IRI=”#manipulates”/>
    <ObjectUnionOf>
        <Class IRI=”#Actuator”/>
        <Class IRI=”#PhysicalEntity”/>
    </ObjectUnionOf>
```

```
</ObjectPropertyDomain>
<ObjectPropertyDomain>
    <ObjectProperty IRI=”#monitors”/>
    <Class IRI=”#Sensor”/>
</ObjectPropertyDomain>
<ObjectPropertyDomain>
    <ObjectProperty IRI=”#plays”/>
    <Class IRI=”#Thing”/>
</ObjectPropertyDomain>
<ObjectPropertyDomain>
    <ObjectProperty IRI=”#reads”/>
    <Class IRI=”#Sensor”/>
</ObjectPropertyDomain>
<ObjectPropertyDomain>
    <ObjectProperty IRI=”#senses”/>
    <Class IRI=”#Sensor”/>
</ObjectPropertyDomain>
<ObjectPropertyDomain>
    <ObjectProperty IRI=”#usesDevice”/>
    <Class IRI=”#Capability”/>
</ObjectPropertyDomain>
<ObjectPropertyRange>
    <ObjectProperty IRI=”#accessesDevice”/>
    <Class IRI=”#Device”/>
</ObjectPropertyRange>
<ObjectPropertyRange>
    <ObjectProperty IRI=”#affects”/>
    <Class IRI=”#Sensor”/>
</ObjectPropertyRange>
<ObjectPropertyRange>
    <ObjectProperty IRI=”#canBePlayedFor”/>
    <Class IRI=”#PhysicalContext”/>
</ObjectPropertyRange>
<ObjectPropertyRange>
    <ObjectProperty IRI=”#canBePlayedFor”/>
    <Class IRI=”#PhysicalEntity”/>
</ObjectPropertyRange>
<ObjectPropertyRange>
    <ObjectProperty IRI=”#canPlay”/>
    <Class IRI=”#Role”/>
</ObjectPropertyRange>
<ObjectPropertyRange>
```

```
    <ObjectProperty IRI=”#contains”/>
    <Class IRI=”#PhysicalEntity”/>
</ObjectPropertyRange>
<ObjectPropertyRange>
    <ObjectProperty IRI=”#hasCapability”/>
    <Class IRI=”#Capability”/>
</ObjectPropertyRange>
<ObjectPropertyRange>
    <ObjectProperty IRI=”#hasCommand”/>
    <Class IRI=”#Command”/>
</ObjectPropertyRange>
<ObjectPropertyRange>
    <ObjectProperty IRI=”#hasDevice”/>
    <Class IRI=”#Device”/>
</ObjectPropertyRange>
<ObjectPropertyRange>
    <ObjectProperty IRI=”#hasEntity”/>
    <Class IRI=”#PhysicalEntity”/>
</ObjectPropertyRange>
<ObjectPropertyRange>
    <ObjectProperty IRI=”#hasProperty”/>
    <Class IRI=”#PhysicalProperty”/>
</ObjectPropertyRange>
<ObjectPropertyRange>
    <ObjectProperty IRI=”#identifies”/>
    <Class IRI=”#PhysicalEntity”/>
</ObjectPropertyRange>
<ObjectPropertyRange>
    <ObjectProperty IRI=”#isAllocatedTo”/>
    <Class IRI=”#PhysicalContext”/>
</ObjectPropertyRange>
<ObjectPropertyRange>
    <ObjectProperty IRI=”#isAllocatedTo”/>
    <Class IRI=”#PhysicalEntity”/>
</ObjectPropertyRange>
<ObjectPropertyRange>
    <ObjectProperty IRI=”#isAttachedTo”/>
    <Class IRI=”#PhysicalEntity”/>
</ObjectPropertyRange>
<ObjectPropertyRange>
    <ObjectProperty IRI=”#isAvailableIn”/>
    <Class IRI=”#PhysicalContext”/>
```

```
</ObjectPropertyRange>
<ObjectPropertyRange>
    <ObjectProperty IRI=”#isPlayedFor”/>
    <Class IRI=”#PhysicalContext”/>
</ObjectPropertyRange>
<ObjectPropertyRange>
    <ObjectProperty IRI=”#isPlayedFor”/>
    <Class IRI=”#PhysicalEntity”/>
</ObjectPropertyRange>
<ObjectPropertyRange>
    <ObjectProperty IRI=”#manipulates”/>
    <ObjectUnionOf>
        <Class IRI=”#PhysicalEntity”/>
        <Class IRI=”#PhysicalProperty”/>
    </ObjectUnionOf>
</ObjectPropertyRange>
<ObjectPropertyRange>
    <ObjectProperty IRI=”#monitors”/>
    <Class IRI=”#Actuator”/>
</ObjectPropertyRange>
<ObjectPropertyRange>
    <ObjectProperty IRI=”#plays”/>
    <Class IRI=”#Role”/>
</ObjectPropertyRange>
<ObjectPropertyRange>
    <ObjectProperty IRI=”#reads”/>
    <Class IRI=”#Tag”/>
</ObjectPropertyRange>
<ObjectPropertyRange>
    <ObjectProperty IRI=”#senses”/>
    <Class IRI=”#PhysicalProperty”/>
</ObjectPropertyRange>
<ObjectPropertyRange>
    <ObjectProperty IRI=”#usesDevice”/>
    <Class IRI=”#Device”/>
</ObjectPropertyRange>
<DataPropertyDomain>
    <DataProperty IRI=”#physicalPropertyData”/>
    <Class IRI=”#PhysicalProperty”/>
</DataPropertyDomain>
<DataPropertyDomain>
    <DataProperty IRI=”#physicalPropertyData”/>
    <ObjectUnionOf>
        <Class IRI=”#PhysicalEntity”/>
        <Class IRI=”#PhysicalProperty”/>
    </ObjectUnionOf>
</DataPropertyDomain>
<DataPropertyDomain>
    <DataProperty IRI=”#serviceMethodURL”/>
    <Class IRI=”#Command”/>
</DataPropertyDomain>
<DataPropertyRange>
    <DataProperty IRI=”#serviceMethodURL”/>
    <Datatype abbreviatedIRI=”xsd:anyURI”/>
</DataPropertyRange>
<DLSafeRule>
    <Annotation>
        <AnnotationProperty IRI=”http://swrl.stanford.edu/ontologies/3.3/swrla.
            owl#isRuleEnabled”/>
        <Literal datatypeIRI=”http://www.w3.org/2001/XMLSchema#boolean”>
            true</Literal>
    </Annotation>
    <Annotation>
        <AnnotationProperty abbreviatedIRI=”rdfs:comment”/>
        <Literal datatypeIRI=”http://www.w3.org/2001/XMLSchema#string”></
            Literal>
    </Annotation>
    <Annotation>
        <AnnotationProperty abbreviatedIRI=”rdfs:label”/>
        <Literal datatypeIRI=”http://www.w3.org/2001/XMLSchema#string”>
            canBePlayedForPhysicalContext</Literal>
    </Annotation>
    <Body>
        <ObjectPropertyAtom>
            <ObjectProperty IRI=”#canPlay”/>
            <Variable IRI=”urn:swrl#IoTUDresdenx”/>
            <Variable IRI=”urn:swrl#IoTUDresdeny”/>
        </ObjectPropertyAtom>
        <ObjectPropertyAtom>
            <ObjectProperty IRI=”#isAvailableIn”/>
            <Variable IRI=”urn:swrl#IoTUDresdenx”/>
            <Variable IRI=”urn:swrl#IoTUDresdenz”/>
        </ObjectPropertyAtom>
    </Body>
```

```
        <Head>
            <ObjectPropertyAtom>
                <ObjectProperty IRI=”#canBePlayedFor”/>
                <Variable IRI=”urn:swrl#IoTUDresdeny”/>
                <Variable IRI=”urn:swrl#IoTUDresdenz”/>
            </ObjectPropertyAtom>
        </Head>
    </DLSafeRule>
    <DLSafeRule>
        <Annotation>
            <AnnotationProperty IRI=”http://swrl.stanford.edu/ontologies/3.3/swrla.
                owl#isRuleEnabled”/>
            <Literal datatypeIRI=”http://www.w3.org/2001/XMLSchema#boolean”>
                true</Literal>
        </Annotation>
        <Annotation>
            <AnnotationProperty abbreviatedIRI=”rdfs:comment”/>
            <Literal datatypeIRI=”http://www.w3.org/2001/XMLSchema#string”></
                Literal>
        </Annotation>
        <Annotation>
            <AnnotationProperty abbreviatedIRI=”rdfs:label”/>
            <Literal datatypeIRI=”http://www.w3.org/2001/XMLSchema#string”>
                canBePlayedForPhysicalEntity</Literal>
        </Annotation>
        <Body>
            <ObjectPropertyAtom>
                <ObjectProperty IRI=”#canPlay”/>
                <Variable IRI=”urn:swrl#x”/>
                <Variable IRI=”urn:swrl#y”/>
            </ObjectPropertyAtom>
            <ObjectPropertyAtom>
                <ObjectProperty IRI=”#isAttachedTo”/>
                <Variable IRI=”urn:swrl#x”/>
                <Variable IRI=”urn:swrl#z”/>
            </ObjectPropertyAtom>
        </Body>
        <Head>
            <ObjectPropertyAtom>
                <ObjectProperty IRI=”#canBePlayedFor”/>
                <Variable IRI=”urn:swrl#y”/>
                <Variable IRI=”urn:swrl#z”/>
        </ObjectPropertyAtom>
    </Head>
</DLSafeRule>
<DLSafeRule>
    <Annotation>
        <AnnotationProperty IRI=”http://swrl.stanford.edu/ontologies/3.3/swrla.
            owl#isRuleEnabled”/>
        <Literal datatypeIRI=”http://www.w3.org/2001/XMLSchema#boolean”>
            true</Literal>
    </Annotation>
    <Annotation>
        <AnnotationProperty abbreviatedIRI=”rdfs:comment”/>
        <Literal datatypeIRI=”http://www.w3.org/2001/XMLSchema#string”></
            Literal>
    </Annotation>
    <Annotation>
        <AnnotationProperty abbreviatedIRI=”rdfs:label”/>
        <Literal datatypeIRI=”http://www.w3.org/2001/XMLSchema#string”>
            accessesDeviceRule</Literal>
    </Annotation>
    <Body>
        <ObjectPropertyAtom>
            <ObjectProperty IRI=”#usesDevice”/>
            <Variable IRI=”urn:swrl#x”/>
            <Variable IRI=”urn:swrl#y”/>
        </ObjectPropertyAtom>
        <ObjectPropertyAtom>
            <ObjectProperty IRI=”#hasCommand”/>
            <Variable IRI=”urn:swrl#x”/>
            <Variable IRI=”urn:swrl#z”/>
        </ObjectPropertyAtom>
    </Body>
    <Head>
        <ObjectPropertyAtom>
            <ObjectProperty IRI=”#accessesDevice”/>
            <Variable IRI=”urn:swrl#z”/>
            <Variable IRI=”urn:swrl#y”/>
        </ObjectPropertyAtom>
    </Head>
</DLSafeRule>
<DLSafeRule>
    <Annotation>
```

```
            <AnnotationProperty IRI=”http://swrl.stanford.edu/ontologies/3.3/swrla.
                owl#isRuleEnabled”/>
            <Literal datatypeIRI=”http://www.w3.org/2001/XMLSchema#boolean”>
                true</Literal>
        </Annotation>
        <Annotation>
            <AnnotationProperty abbreviatedIRI=”rdfs:comment”/>
            <Literal datatypeIRI=”http://www.w3.org/2001/XMLSchema#string”></
                Literal>
        </Annotation>
        <Annotation>
            <AnnotationProperty abbreviatedIRI=”rdfs:label”/>
            <Literal datatypeIRI=”http://www.w3.org/2001/XMLSchema#string”>
                isAvailableInContextRule</Literal>
        </Annotation>
        <Body>
            <ObjectPropertyAtom>
                <ObjectProperty IRI=”#isAttachedTo”/>
                <Variable IRI=”urn:swrl#x”/>
                <Variable IRI=”urn:swrl#y”/>
            </ObjectPropertyAtom>
            <ObjectPropertyAtom>
                <ObjectProperty IRI=”#hasEntity”/>
                <Variable IRI=”urn:swrl#z”/>
                <Variable IRI=”urn:swrl#y”/>
            </ObjectPropertyAtom>
        </Body>
        <Head>
            <ObjectPropertyAtom>
                <ObjectProperty IRI=”#isAvailableIn”/>
                <Variable IRI=”urn:swrl#x”/>
                <Variable IRI=”urn:swrl#z”/>
            </ObjectPropertyAtom>
        </Head>
    </DLSafeRule>
    <DLSafeRule>
        <Annotation>
            <AnnotationProperty IRI=”http://swrl.stanford.edu/ontologies/3.3/swrla.
                owl#isRuleEnabled”/>
            <Literal datatypeIRI=”http://www.w3.org/2001/XMLSchema#boolean”>
                true</Literal>
        </Annotation>
    <Annotation>
        <AnnotationProperty abbreviatedIRI=”rdfs:comment”/>
        <Literal datatypeIRI=”http://www.w3.org/2001/XMLSchema#string”></
            Literal>
    </Annotation>
    <Annotation>
        <AnnotationProperty abbreviatedIRI=”rdfs:label”/>
        <Literal datatypeIRI=”http://www.w3.org/2001/XMLSchema#string”>
            isPlayedForEntity</Literal>
    </Annotation>
    <Body>
        <ObjectPropertyAtom>
            <ObjectProperty IRI=”#plays”/>
            <Variable IRI=”urn:swrl#x”/>
            <Variable IRI=”urn:swrl#y”/>
        </ObjectPropertyAtom>
        <ObjectPropertyAtom>
            <ObjectProperty IRI=”#isAttachedTo”/>
            <Variable IRI=”urn:swrl#x”/>
            <Variable IRI=”urn:swrl#z”/>
        </ObjectPropertyAtom>
    </Body>
    <Head>
        <ObjectPropertyAtom>
            <ObjectProperty IRI=”#isPlayedFor”/>
            <Variable IRI=”urn:swrl#y”/>
            <Variable IRI=”urn:swrl#z”/>
        </ObjectPropertyAtom>
    </Head>
</DLSafeRule>
<DLSafeRule>
    <Annotation>
        <AnnotationProperty IRI=”http://swrl.stanford.edu/ontologies/3.3/swrla.
            owl#isRuleEnabled”/>
        <Literal datatypeIRI=”http://www.w3.org/2001/XMLSchema#boolean”>
            true</Literal>
    </Annotation>
    <Annotation>
        <AnnotationProperty abbreviatedIRI=”rdfs:comment”/>
        <Literal datatypeIRI=”http://www.w3.org/2001/XMLSchema#string”></
            Literal>
    </Annotation>
```

```
        <Annotation>
            <AnnotationProperty abbreviatedIRI=”rdfs:label”/>
            <Literal datatypeIRI=”http://www.w3.org/2001/XMLSchema#string”>
                isPlayedForContext</Literal>
        </Annotation>
        <Body>
            <ObjectPropertyAtom>
                <ObjectProperty IRI=”#plays”/>
                <Variable IRI=”urn:swrl#x”/>
                <Variable IRI=”urn:swrl#y”/>
            </ObjectPropertyAtom>
            <ObjectPropertyAtom>
                <ObjectProperty IRI=”#isAvailableIn”/>
                <Variable IRI=”urn:swrl#x”/>
                <Variable IRI=”urn:swrl#z”/>
            </ObjectPropertyAtom>
        </Body>
        <Head>
            <ObjectPropertyAtom>
                <ObjectProperty IRI=”#isPlayedFor”/>
                <Variable IRI=”urn:swrl#y”/>
                <Variable IRI=”urn:swrl#z”/>
            </ObjectPropertyAtom>
        </Head>
    </DLSafeRule>
    <DLSafeRule>
        <Annotation>
            <AnnotationProperty IRI=”http://swrl.stanford.edu/ontologies/3.3/swrla.
                owl#isRuleEnabled”/>
            <Literal datatypeIRI=”http://www.w3.org/2001/XMLSchema#boolean”>
                true</Literal>
        </Annotation>
        <Annotation>
            <AnnotationProperty abbreviatedIRI=”rdfs:comment”/>
            <Literal datatypeIRI=”http://www.w3.org/2001/XMLSchema#string”></
                Literal>
        </Annotation>
        <Annotation>
            <AnnotationProperty abbreviatedIRI=”rdfs:label”/>
            <Literal datatypeIRI=”http://www.w3.org/2001/XMLSchema#string”>
                isAllocatedToRule</Literal>
        </Annotation>
    <Body>
        <ObjectPropertyAtom>
            <ObjectProperty IRI=”#plays”/>
            <Variable IRI=”urn:swrl#x”/>
            <Variable IRI=”urn:swrl#y”/>
        </ObjectPropertyAtom>
        <ObjectPropertyAtom>
            <ObjectProperty IRI=”#isPlayedFor”/>
            <Variable IRI=”urn:swrl#y”/>
            <Variable IRI=”urn:swrl#z”/>
        </ObjectPropertyAtom>
    </Body>
    <Head>
        <ObjectPropertyAtom>
            <ObjectProperty IRI=”#isAllocatedTo”/>
            <Variable IRI=”urn:swrl#x”/>
            <Variable IRI=”urn:swrl#z”/>
        </ObjectPropertyAtom>
    </Head>
</DLSafeRule>
<DLSafeRule>
    <Annotation>
        <AnnotationProperty IRI=”http://swrl.stanford.edu/ontologies/3.3/swrla.
            owl#isRuleEnabled”/>
        <Literal datatypeIRI=”http://www.w3.org/2001/XMLSchema#boolean”>
            true</Literal>
    </Annotation>
    <Annotation>
        <AnnotationProperty abbreviatedIRI=”rdfs:comment”/>
        <Literal datatypeIRI=”http://www.w3.org/2001/XMLSchema#string”></
            Literal>
    </Annotation>
    <Annotation>
        <AnnotationProperty abbreviatedIRI=”rdfs:label”/>
        <Literal datatypeIRI=”http://www.w3.org/2001/XMLSchema#string”>
            monitorAndAffect</Literal>
    </Annotation>
    <Body>
        <ObjectPropertyAtom>
            <ObjectProperty IRI=”#manipulates”/>
            <Variable IRI=”urn:swrl#x”/>
            <Variable IRI=”urn:swrl#y”/>
            </ObjectPropertyAtom>
            <ObjectPropertyAtom>
                <ObjectProperty IRI=”#senses”/>
                <Variable IRI=”urn:swrl#z”/>
                <Variable IRI=”urn:swrl#y”/>
            </ObjectPropertyAtom>
        </Body>
        <Head>
            <ObjectPropertyAtom>
                <ObjectProperty IRI=”#affects”/>
                <Variable IRI=”urn:swrl#x”/>
                <Variable IRI=”urn:swrl#z”/>
            </ObjectPropertyAtom>
        </Head>
    </DLSafeRule>
</Ontology>
```

A-Box for Example Scenario Model

```
<?xml version=”1.0”?>
<Ontology xmlns=”http://www.w3.org/2002/07/owl#”
     xml:base=”http://www.semanticweb.org/IoTUDresden/ABox”
     xmlns:rdf=”http://www.w3.org/1999/02/22−rdf−syntax−ns#”
     xmlns:xml=”http://www.w3.org/XML/1998/namespace”
     xmlns:xsd=”http://www.w3.org/2001/XMLSchema#”
     xmlns:rdfs=”http://www.w3.org/2000/01/rdf−schema#”
     ontologyIRI=”http://www.semanticweb.org/IoTUDresden/ABox”>
    <Prefix name=”owl” IRI=”http://www.w3.org/2002/07/owl#”/>
    <Prefix name=”rdf” IRI=”http://www.w3.org/1999/02/22−rdf−syntax−ns#”/>
    <Prefix name=”xml” IRI=”http://www.w3.org/XML/1998/namespace”/>
    <Prefix name=”xsd” IRI=”http://www.w3.org/2001/XMLSchema#”/>
    <Prefix name=”rdfs” IRI=”http://www.w3.org/2000/01/rdf−schema#”/>
    <Declaration>
        <ObjectProperty IRI=”http://www.semanticweb.org/IoTUDresden#affects”/>
    </Declaration>
    <Declaration>
        <Class IRI=”http://www.semanticweb.org/IoTUDresden#Device”/>
    </Declaration>
    <Declaration>
        <Class IRI=”http://www.semanticweb.org/IoTUDresden#Capability”/>
    </Declaration>
    <Declaration>
        <NamedIndividual IRI=”http://www.semanticweb.org/IoTUDresden#Kitchen”/
            >
    </Declaration>
    <Declaration>
        <Class IRI=”http://www.semanticweb.org/IoTUDresden#PhysicalProperty”/>
    </Declaration>
    <Declaration>
        <NamedIndividual IRI=”http://www.semanticweb.org/IoTUDresden#
            ResidentBob”/>
    </Declaration>
    <Declaration>
        <Class IRI=”http://www.semanticweb.org/IoTUDresden#Role”/>
    </Declaration>
    <Declaration>
        <ObjectProperty IRI=”http://www.semanticweb.org/IoTUDresden#monitors”/
            >
    </Declaration>
    <Declaration>
    <NamedIndividual IRI=”http://www.semanticweb.org/IoTUDresden#
        provideHealthStatus”/>
</Declaration>
<Declaration>
    <NamedIndividual IRI=”http://www.semanticweb.org/IoTUDresden#
        HealthMonitor”/>
</Declaration>
<Declaration>
    <Class IRI=”http://www.semanticweb.org/IoTUDresden#Sensor”/>
</Declaration>
<Declaration>
    <Class IRI=”http://www.semanticweb.org/IoTUDresden#PhysicalEntity”/>
</Declaration>
<Declaration>
    <NamedIndividual IRI=”http://www.semanticweb.org/IoTUDresden#
        HealthStatus”/>
</Declaration>
<Declaration>
    <NamedIndividual IRI=”http://www.semanticweb.org/IoTUDresden#
        InteractionDevice”/>
</Declaration>
<Declaration>
    <Class IRI=”http://www.semanticweb.org/IoTUDresden#Command”/>
</Declaration>
<Declaration>
    <ObjectProperty IRI=”http://www.semanticweb.org/IoTUDresden#contains”/
        >
</Declaration>
<Declaration>
    <NamedIndividual IRI=”http://www.semanticweb.org/IoTUDresden#
        SmartHome”/>
</Declaration>
<Declaration>
    <Class IRI=”http://www.semanticweb.org/IoTUDresden#OnCommand”/>
</Declaration>
<Declaration>
    <NamedIndividual IRI=”http://www.semanticweb.org/IoTUDresden#
        provideResidentResponsiveness”/>
</Declaration>
<Declaration>
    <NamedIndividual IRI=”http://www.semanticweb.org/IoTUDresden#
        InteractionSensor”/>
```

```
</Declaration>
<Declaration>
    <Class IRI=”http://www.semanticweb.org/IoTUDresden#PhysicalContext”/>
</Declaration>
<Declaration>
    <NamedIndividual IRI=”http://www.semanticweb.org/IoTUDresden#
        SmartHomeContext”/>
</Declaration>
<Declaration>
    <NamedIndividual IRI=”http://www.semanticweb.org/IoTUDresden#
        provideMedicalAssistance”/>
</Declaration>
<Declaration>
    <NamedIndividual IRI=”http://www.semanticweb.org/IoTUDresden#
        HealthSensor”/>
</Declaration>
<Declaration>
    <Class IRI=”http://www.semanticweb.org/IoTUDresden#OffCommand”/>
</Declaration>
<Declaration>
    <NamedIndividual IRI=”http://www.semanticweb.org/IoTUDresden#
        ResponsiveStatus”/>
</Declaration>
<Declaration>
    <Class IRI=”http://www.semanticweb.org/IoTUDresden#Thing”/>
</Declaration>
<Declaration>
    <NamedIndividual IRI=”http://www.semanticweb.org/IoTUDresden#
        GetResponsiveStatus”/>
</Declaration>
<Declaration>
    <Class IRI=”http://www.semanticweb.org/IoTUDresden#Tag”/>
</Declaration>
<Declaration>
    <NamedIndividual IRI=”http://www.semanticweb.org/IoTUDresden#
        TriggerAutomaticEmergencyCall”/>
</Declaration>
<Declaration>
    <NamedIndividual IRI=”http://www.semanticweb.org/IoTUDresden#
        AutomaticEmergencyCaller”/>
</Declaration>
<Declaration>
```

```
    <Class IRI=”http://www.semanticweb.org/IoTUDresden#Actuator”/>
</Declaration>
<Declaration>
    <NamedIndividual IRI=”http://www.semanticweb.org/IoTUDresden#
        GetHealthStatus”/>
</Declaration>
<Declaration>
    <Class IRI=”http://www.semanticweb.org/IoTUDresden#GetDataCommand”/
        >
</Declaration>
<Declaration>
    <NamedIndividual IRI=”http://www.semanticweb.org/IoTUDresden#
        SmartPhone”/>
</Declaration>
<Declaration>
    <NamedIndividual IRI=”http://www.semanticweb.org/IoTUDresden#
        EmergencyCallApp”/>
</Declaration>
<Declaration>
    <NamedIndividual IRI=”http://www.semanticweb.org/IoTUDresden#
        SmartWatch”/>
</Declaration>
<Declaration>
    <ObjectProperty IRI=”http://www.semanticweb.org/IoTUDresden#
        manipulates”/>
</Declaration>
<SubClassOf>
    <Class IRI=”http://www.semanticweb.org/IoTUDresden#Actuator”/>
    <Class IRI=”http://www.semanticweb.org/IoTUDresden#Device”/>
</SubClassOf>
<SubClassOf>
    <Class IRI=”http://www.semanticweb.org/IoTUDresden#Actuator”/>
    <Class IRI=”http://www.semanticweb.org/IoTUDresden#Thing”/>
</SubClassOf>
<SubClassOf>
    <Class IRI=”http://www.semanticweb.org/IoTUDresden#GetDataCommand”/
        >
    <Class IRI=”http://www.semanticweb.org/IoTUDresden#Command”/>
</SubClassOf>
<SubClassOf>
    <Class IRI=”http://www.semanticweb.org/IoTUDresden#OffCommand”/>
    <Class IRI=”http://www.semanticweb.org/IoTUDresden#Command”/>
```

```
</SubClassOf>
<SubClassOf>
    <Class IRI=”http://www.semanticweb.org/IoTUDresden#OnCommand”/>
    <Class IRI=”http://www.semanticweb.org/IoTUDresden#Command”/>
</SubClassOf>
<SubClassOf>
    <Class IRI=”http://www.semanticweb.org/IoTUDresden#Sensor”/>
    <Class IRI=”http://www.semanticweb.org/IoTUDresden#Device”/>
</SubClassOf>
<SubClassOf>
    <Class IRI=”http://www.semanticweb.org/IoTUDresden#Sensor”/>
    <Class IRI=”http://www.semanticweb.org/IoTUDresden#Thing”/>
</SubClassOf>
<SubClassOf>
    <Class IRI=”http://www.semanticweb.org/IoTUDresden#Tag”/>
    <Class IRI=”http://www.semanticweb.org/IoTUDresden#Device”/>
</SubClassOf>
<SubClassOf>
    <Class IRI=”http://www.semanticweb.org/IoTUDresden#Tag”/>
    <Class IRI=”http://www.semanticweb.org/IoTUDresden#Thing”/>
</SubClassOf>
<ClassAssertion>
    <Class IRI=”http://www.semanticweb.org/IoTUDresden#Role”/>
    <NamedIndividual IRI=”http://www.semanticweb.org/IoTUDresden#
        AutomaticEmergencyCaller”/>
</ClassAssertion>
<ClassAssertion>
    <Class IRI=”http://www.semanticweb.org/IoTUDresden#Actuator”/>
    <NamedIndividual IRI=”http://www.semanticweb.org/IoTUDresden#
        EmergencyCallApp”/>
</ClassAssertion>
<ClassAssertion>
    <Class IRI=”http://www.semanticweb.org/IoTUDresden#Device”/>
    <NamedIndividual IRI=”http://www.semanticweb.org/IoTUDresden#
        EmergencyCallApp”/>
</ClassAssertion>
<ClassAssertion>
    <Class IRI=”http://www.semanticweb.org/IoTUDresden#Thing”/>
    <NamedIndividual IRI=”http://www.semanticweb.org/IoTUDresden#
        EmergencyCallApp”/>
</ClassAssertion>
<ClassAssertion>
```

```
        <Class IRI=”http://www.semanticweb.org/IoTUDresden#Command”/>
        <NamedIndividual IRI=”http://www.semanticweb.org/IoTUDresden#
            GetHealthStatus”/>
    </ClassAssertion>
    <ClassAssertion>
        <Class IRI=”http://www.semanticweb.org/IoTUDresden#GetDataCommand”/
            >
        <NamedIndividual IRI=”http://www.semanticweb.org/IoTUDresden#
            GetHealthStatus”/>
    </ClassAssertion>
    <ClassAssertion>
        <Class IRI=”http://www.semanticweb.org/IoTUDresden#Command”/>
        <NamedIndividual IRI=”http://www.semanticweb.org/IoTUDresden#
            GetResponsiveStatus”/>
    </ClassAssertion>
    <ClassAssertion>
        <Class IRI=”http://www.semanticweb.org/IoTUDresden#GetDataCommand”/
            >
        <NamedIndividual IRI=”http://www.semanticweb.org/IoTUDresden#
            GetResponsiveStatus”/>
    </ClassAssertion>
    <ClassAssertion>
        <Class IRI=”http://www.semanticweb.org/IoTUDresden#Role”/>
        <NamedIndividual IRI=”http://www.semanticweb.org/IoTUDresden#
            HealthMonitor”/>
    </ClassAssertion>
    <ClassAssertion>
        <Class IRI=”http://www.semanticweb.org/IoTUDresden#Device”/>
        <NamedIndividual IRI=”http://www.semanticweb.org/IoTUDresden#
            HealthSensor”/>
    </ClassAssertion>
    <ClassAssertion>
        <Class IRI=”http://www.semanticweb.org/IoTUDresden#Sensor”/>
        <NamedIndividual IRI=”http://www.semanticweb.org/IoTUDresden#
            HealthSensor”/>
    </ClassAssertion>
    <ClassAssertion>
        <Class IRI=”http://www.semanticweb.org/IoTUDresden#Thing”/>
        <NamedIndividual IRI=”http://www.semanticweb.org/IoTUDresden#
            HealthSensor”/>
    </ClassAssertion>
    <ClassAssertion>
    <Class IRI=”http://www.semanticweb.org/IoTUDresden#PhysicalProperty”/>
    <NamedIndividual IRI=”http://www.semanticweb.org/IoTUDresden#
        HealthStatus”/>
</ClassAssertion>
<ClassAssertion>
    <Class IRI=”http://www.semanticweb.org/IoTUDresden#Role”/>
    <NamedIndividual IRI=”http://www.semanticweb.org/IoTUDresden#
        InteractionDevice”/>
</ClassAssertion>
<ClassAssertion>
    <Class IRI=”http://www.semanticweb.org/IoTUDresden#Device”/>
    <NamedIndividual IRI=”http://www.semanticweb.org/IoTUDresden#
        InteractionSensor”/>
</ClassAssertion>
<ClassAssertion>
    <Class IRI=”http://www.semanticweb.org/IoTUDresden#Sensor”/>
    <NamedIndividual IRI=”http://www.semanticweb.org/IoTUDresden#
        InteractionSensor”/>
</ClassAssertion>
<ClassAssertion>
    <Class IRI=”http://www.semanticweb.org/IoTUDresden#Thing”/>
    <NamedIndividual IRI=”http://www.semanticweb.org/IoTUDresden#
        InteractionSensor”/>
</ClassAssertion>
<ClassAssertion>
    <Class IRI=”http://www.semanticweb.org/IoTUDresden#PhysicalEntity”/>
    <NamedIndividual IRI=”http://www.semanticweb.org/IoTUDresden#Kitchen”/
        >
</ClassAssertion>
<ClassAssertion>
    <Class IRI=”http://www.semanticweb.org/IoTUDresden#PhysicalContext”/>
    <NamedIndividual IRI=”http://www.semanticweb.org/IoTUDresden#
        ResidentBob”/>
</ClassAssertion>
<ClassAssertion>
    <Class IRI=”http://www.semanticweb.org/IoTUDresden#PhysicalEntity”/>
    <NamedIndividual IRI=”http://www.semanticweb.org/IoTUDresden#
        ResidentBob”/>
</ClassAssertion>
<ClassAssertion>
    <Class IRI=”http://www.semanticweb.org/IoTUDresden#PhysicalProperty”/>
    <NamedIndividual IRI=”http://www.semanticweb.org/IoTUDresden#
```

```
            ResponsiveStatus”/>
    </ClassAssertion>
    <ClassAssertion>
        <Class IRI=”http://www.semanticweb.org/IoTUDresden#PhysicalEntity”/>
        <NamedIndividual IRI=”http://www.semanticweb.org/IoTUDresden#
            SmartHome”/>
    </ClassAssertion>
    <ClassAssertion>
        <Class IRI=”http://www.semanticweb.org/IoTUDresden#PhysicalContext”/>
        <NamedIndividual IRI=”http://www.semanticweb.org/IoTUDresden#
            SmartHomeContext”/>
    </ClassAssertion>
    <ClassAssertion>
        <Class IRI=”http://www.semanticweb.org/IoTUDresden#PhysicalEntity”/>
        <NamedIndividual IRI=”http://www.semanticweb.org/IoTUDresden#
            SmartHomeContext”/>
    </ClassAssertion>
    <ClassAssertion>
        <Class IRI=”http://www.semanticweb.org/IoTUDresden#Thing”/>
        <NamedIndividual IRI=”http://www.semanticweb.org/IoTUDresden#
            SmartPhone”/>
    </ClassAssertion>
    <ClassAssertion>
        <Class IRI=”http://www.semanticweb.org/IoTUDresden#Thing”/>
        <NamedIndividual IRI=”http://www.semanticweb.org/IoTUDresden#
            SmartWatch”/>
    </ClassAssertion>
    <ClassAssertion>
        <Class IRI=”http://www.semanticweb.org/IoTUDresden#Command”/>
        <NamedIndividual IRI=”http://www.semanticweb.org/IoTUDresden#
            TriggerAutomaticEmergencyCall”/>
    </ClassAssertion>
    <ClassAssertion>
        <Class IRI=”http://www.semanticweb.org/IoTUDresden#OnCommand”/>
        <NamedIndividual IRI=”http://www.semanticweb.org/IoTUDresden#
            TriggerAutomaticEmergencyCall”/>
    </ClassAssertion>
    <ClassAssertion>
        <Class IRI=”http://www.semanticweb.org/IoTUDresden#Capability”/>
        <NamedIndividual IRI=”http://www.semanticweb.org/IoTUDresden#
            provideHealthStatus”/>
    </ClassAssertion>
    <ClassAssertion>
        <Class IRI=”http://www.semanticweb.org/IoTUDresden#Capability”/>
        <NamedIndividual IRI=”http://www.semanticweb.org/IoTUDresden#
            provideMedicalAssistance”/>
    </ClassAssertion>
    <ClassAssertion>
        <Class IRI=”http://www.semanticweb.org/IoTUDresden#Capability”/>
        <NamedIndividual IRI=”http://www.semanticweb.org/IoTUDresden#
            provideResidentResponsiveness”/>
    </ClassAssertion>
    <InverseObjectProperties>
        <ObjectProperty IRI=”http://www.semanticweb.org/IoTUDresden#affects”/>
        <ObjectProperty IRI=”http://www.semanticweb.org/IoTUDresden#monitors”/
            >
    </InverseObjectProperties>
    <InverseObjectProperties>
        <ObjectProperty abbreviatedIRI=”owl:topObjectProperty”/>
        <ObjectProperty abbreviatedIRI=”owl:topObjectProperty”/>
    </InverseObjectProperties>
    <SymmetricObjectProperty>
        <ObjectProperty abbreviatedIRI=”owl:topObjectProperty”/>
    </SymmetricObjectProperty>
    <TransitiveObjectProperty>
        <ObjectProperty IRI=”http://www.semanticweb.org/IoTUDresden#contains”/
            >
    </TransitiveObjectProperty>
    <TransitiveObjectProperty>
        <ObjectProperty IRI=”http://www.semanticweb.org/IoTUDresden#
            manipulates”/>
    </TransitiveObjectProperty>
    <TransitiveObjectProperty>
        <ObjectProperty abbreviatedIRI=”owl:topObjectProperty”/>
    </TransitiveObjectProperty>
</Ontology>
```

A-Box for Extended Example Scenario Model

```
<?xml version=”1.0”?>
<Ontology xmlns=”http://www.w3.org/2002/07/owl#”
     xml:base=”http://www.semanticweb.org/IoTUDresden/Inferred/ABox”
     xmlns:rdf=”http://www.w3.org/1999/02/22−rdf−syntax−ns#”
     xmlns:xml=”http://www.w3.org/XML/1998/namespace”
     xmlns:xsd=”http://www.w3.org/2001/XMLSchema#”
     xmlns:rdfs=”http://www.w3.org/2000/01/rdf−schema#”
     ontologyIRI=”http://www.semanticweb.org/IoTUDresden/Inferred/ABox”>
    <Prefix name=”owl” IRI=”http://www.w3.org/2002/07/owl#”/>
    <Prefix name=”rdf” IRI=”http://www.w3.org/1999/02/22−rdf−syntax−ns#”/>
    <Prefix name=”xml” IRI=”http://www.w3.org/XML/1998/namespace”/>
    <Prefix name=”xsd” IRI=”http://www.w3.org/2001/XMLSchema#”/>
    <Prefix name=”rdfs” IRI=”http://www.w3.org/2000/01/rdf−schema#”/>
    <Declaration>
        <ObjectProperty IRI=”http://www.semanticweb.org/IoTUDresden#affects”/>
    </Declaration>
    <Declaration>
        <NamedIndividual IRI=”http://www.semanticweb.org/IoTUDresden#
            DoorStatus”/>
    </Declaration>
    <Declaration>
        <Class IRI=”http://www.semanticweb.org/IoTUDresden#Device”/>
    </Declaration>
    <Declaration>
        <NamedIndividual IRI=”http://www.semanticweb.org/IoTUDresden#
            StoveController”/>
    </Declaration>
    <Declaration>
        <Class IRI=”http://www.semanticweb.org/IoTUDresden#Capability”/>
    </Declaration>
    <Declaration>
        <NamedIndividual IRI=”http://www.semanticweb.org/IoTUDresden#
            provideAudio”/>
    </Declaration>
    <Declaration>
        <NamedIndividual IRI=”http://www.semanticweb.org/IoTUDresden#
            NFCReader”/>
    </Declaration>
    <Declaration>
        <NamedIndividual IRI=”http://www.semanticweb.org/IoTUDresden#Kitchen”/
            >
```

```
</Declaration>
<Declaration>
    <NamedIndividual IRI=”http://www.semanticweb.org/IoTUDresden#
        ResidentAlice”/>
</Declaration>
<Declaration>
    <NamedIndividual IRI=”http://www.semanticweb.org/IoTUDresden#Corridor”
        />
</Declaration>
<Declaration>
    <ObjectProperty IRI=”http://www.semanticweb.org/IoTUDresden#monitors”/
        >
</Declaration>
<Declaration>
    <NamedIndividual IRI=”http://www.semanticweb.org/IoTUDresden#
        prohibitAccessToHome”/>
</Declaration>
<Declaration>
    <NamedIndividual IRI=”http://www.semanticweb.org/IoTUDresden#
        PersonAuthenticator”/>
</Declaration>
<Declaration>
    <NamedIndividual IRI=”http://www.semanticweb.org/IoTUDresden#
        ResidentBob”/>
</Declaration>
<Declaration>
    <ObjectProperty IRI=”http://www.semanticweb.org/IoTUDresden#contains”/
        >
</Declaration>
<Declaration>
    <NamedIndividual IRI=”http://www.semanticweb.org/IoTUDresden#
        KeyMaticLock”/>
</Declaration>
<Declaration>
    <NamedIndividual IRI=”http://www.semanticweb.org/IoTUDresden#
        SmartHome”/>
</Declaration>
<Declaration>
    <Class IRI=”http://www.semanticweb.org/IoTUDresden#OnCommand”/>
</Declaration>
<Declaration>
    <NamedIndividual IRI=”http://www.semanticweb.org/IoTUDresden#
```

```
            DoorActuator”/>
</Declaration>
<Declaration>
    <Class IRI=”http://www.semanticweb.org/IoTUDresden#PhysicalContext”/>
</Declaration>
<Declaration>
    <NamedIndividual IRI=”http://www.semanticweb.org/IoTUDresden#stopAudio
            ”/>
</Declaration>
<Declaration>
    <NamedIndividual IRI=”http://www.semanticweb.org/IoTUDresden#
            switchOnStove”/>
</Declaration>
<Declaration>
    <Class IRI=”http://www.semanticweb.org/IoTUDresden#Thing”/>
</Declaration>
<Declaration>
    <NamedIndividual IRI=”http://www.semanticweb.org/IoTUDresden#
            switchOffStove”/>
</Declaration>
<Declaration>
    <NamedIndividual IRI=”http://www.semanticweb.org/IoTUDresden#
            DoorOpener”/>
</Declaration>
<Declaration>
    <NamedIndividual IRI=”http://www.semanticweb.org/IoTUDresden#
            LivingRoom”/>
</Declaration>
<Declaration>
    <NamedIndividual IRI=”http://www.semanticweb.org/IoTUDresden#
            AudioPlayer”/>
</Declaration>
<Declaration>
    <Class IRI=”http://www.semanticweb.org/IoTUDresden#GetDataCommand”/
            >
</Declaration>
<Declaration>
    <NamedIndividual IRI=”http://www.semanticweb.org/IoTUDresden#
            MusicPlayer”/>
</Declaration>
<Declaration>
    <NamedIndividual IRI=”http://www.semanticweb.org/IoTUDresden#
```

```
        FrontDoor”/>
</Declaration>
<Declaration>
    <NamedIndividual IRI=”http://www.semanticweb.org/IoTUDresden#
        SmartWatch”/>
</Declaration>
<Declaration>
    <ObjectProperty IRI=”http://www.semanticweb.org/IoTUDresden#
        manipulates”/>
</Declaration>
<Declaration>
    <NamedIndividual IRI=”http://www.semanticweb.org/IoTUDresden#
        KodiPlayer”/>
</Declaration>
<Declaration>
    <NamedIndividual IRI=”http://www.semanticweb.org/IoTUDresden#playAudio
        ”/>
</Declaration>
<Declaration>
    <NamedIndividual IRI=”http://www.semanticweb.org/IoTUDresden#
        NFCSensor”/>
</Declaration>
<Declaration>
    <Class IRI=”http://www.semanticweb.org/IoTUDresden#PhysicalProperty”/>
</Declaration>
<Declaration>
    <Class IRI=”http://www.semanticweb.org/IoTUDresden#Role”/>
</Declaration>
<Declaration>
    <Class IRI=”http://www.semanticweb.org/IoTUDresden#Sensor”/>
</Declaration>
<Declaration>
    <NamedIndividual IRI=”http://www.semanticweb.org/IoTUDresden#
        StoveStatus”/>
</Declaration>
<Declaration>
    <NamedIndividual IRI=”http://www.semanticweb.org/IoTUDresden#
        AudioStatus”/>
</Declaration>
<Declaration>
    <Class IRI=”http://www.semanticweb.org/IoTUDresden#PhysicalEntity”/>
</Declaration>
```

```
<Declaration>
    <NamedIndividual IRI=”http://www.semanticweb.org/IoTUDresden#
        LibeliumHealthKit”/>
</Declaration>
<Declaration>
    <Class IRI=”http://www.semanticweb.org/IoTUDresden#Command”/>
</Declaration>
<Declaration>
    <NamedIndividual IRI=”http://www.semanticweb.org/IoTUDresden#closeDoor
        ”/>
</Declaration>
<Declaration>
    <NamedIndividual IRI=”http://www.semanticweb.org/IoTUDresden#openDoor
        ”/>
</Declaration>
<Declaration>
    <NamedIndividual IRI=”http://www.semanticweb.org/IoTUDresden#Stove”/>
</Declaration>
<Declaration>
    <NamedIndividual IRI=”http://www.semanticweb.org/IoTUDresden#
        PowerSupplySwitch”/>
</Declaration>
<Declaration>
    <NamedIndividual IRI=”http://www.semanticweb.org/IoTUDresden#
        SmartHomeContext”/>
</Declaration>
<Declaration>
    <NamedIndividual IRI=”http://www.semanticweb.org/IoTUDresden#
        DigitalStromController”/>
</Declaration>
<Declaration>
    <NamedIndividual IRI=”http://www.semanticweb.org/IoTUDresden#
        provideAccessToHome”/>
</Declaration>
<Declaration>
    <Class IRI=”http://www.semanticweb.org/IoTUDresden#OffCommand”/>
</Declaration>
<Declaration>
    <Class IRI=”http://www.semanticweb.org/IoTUDresden#Tag”/>
</Declaration>
<Declaration>
    <NamedIndividual IRI=”http://www.semanticweb.org/IoTUDresden#
```

```
        provideStoveControl”/>
</Declaration>
<Declaration>
    <Class IRI=”http://www.semanticweb.org/IoTUDresden#Actuator”/>
</Declaration>
<Declaration>
    <NamedIndividual IRI=”http://www.semanticweb.org/IoTUDresden#
        SmartPhone”/>
</Declaration>
<Declaration>
    <NamedIndividual IRI=”http://www.semanticweb.org/IoTUDresden#
        GetIdentification”/>
</Declaration>
<SubClassOf>
    <Class IRI=”http://www.semanticweb.org/IoTUDresden#Actuator”/>
    <Class IRI=”http://www.semanticweb.org/IoTUDresden#Device”/>
</SubClassOf>
<SubClassOf>
    <Class IRI=”http://www.semanticweb.org/IoTUDresden#Actuator”/>
    <Class IRI=”http://www.semanticweb.org/IoTUDresden#Thing”/>
</SubClassOf>
<SubClassOf>
    <Class IRI=”http://www.semanticweb.org/IoTUDresden#GetDataCommand”/
        >
    <Class IRI=”http://www.semanticweb.org/IoTUDresden#Command”/>
</SubClassOf>
<SubClassOf>
    <Class IRI=”http://www.semanticweb.org/IoTUDresden#OffCommand”/>
    <Class IRI=”http://www.semanticweb.org/IoTUDresden#Command”/>
</SubClassOf>
<SubClassOf>
    <Class IRI=”http://www.semanticweb.org/IoTUDresden#OnCommand”/>
    <Class IRI=”http://www.semanticweb.org/IoTUDresden#Command”/>
</SubClassOf>
<SubClassOf>
    <Class IRI=”http://www.semanticweb.org/IoTUDresden#Sensor”/>
    <Class IRI=”http://www.semanticweb.org/IoTUDresden#Device”/>
</SubClassOf>
<SubClassOf>
    <Class IRI=”http://www.semanticweb.org/IoTUDresden#Sensor”/>
    <Class IRI=”http://www.semanticweb.org/IoTUDresden#Thing”/>
</SubClassOf>
```

```
<SubClassOf>
    <Class IRI=”http://www.semanticweb.org/IoTUDresden#Tag”/>
    <Class IRI=”http://www.semanticweb.org/IoTUDresden#Device”/>
</SubClassOf>
<SubClassOf>
    <Class IRI=”http://www.semanticweb.org/IoTUDresden#Tag”/>
    <Class IRI=”http://www.semanticweb.org/IoTUDresden#Thing”/>
</SubClassOf>
<ClassAssertion>
    <Class IRI=”http://www.semanticweb.org/IoTUDresden#Actuator”/>
    <NamedIndividual IRI=”http://www.semanticweb.org/IoTUDresden#
        AudioPlayer”/>
</ClassAssertion>
<ClassAssertion>
    <Class IRI=”http://www.semanticweb.org/IoTUDresden#Device”/>
    <NamedIndividual IRI=”http://www.semanticweb.org/IoTUDresden#
        AudioPlayer”/>
</ClassAssertion>
<ClassAssertion>
    <Class IRI=”http://www.semanticweb.org/IoTUDresden#Thing”/>
    <NamedIndividual IRI=”http://www.semanticweb.org/IoTUDresden#
        AudioPlayer”/>
</ClassAssertion>
<ClassAssertion>
    <Class IRI=”http://www.semanticweb.org/IoTUDresden#PhysicalProperty”/>
    <NamedIndividual IRI=”http://www.semanticweb.org/IoTUDresden#
        AudioStatus”/>
</ClassAssertion>
<ClassAssertion>
    <Class IRI=”http://www.semanticweb.org/IoTUDresden#PhysicalEntity”/>
    <NamedIndividual IRI=”http://www.semanticweb.org/IoTUDresden#Corridor”
        />
</ClassAssertion>
<ClassAssertion>
    <Class IRI=”http://www.semanticweb.org/IoTUDresden#Thing”/>
    <NamedIndividual IRI=”http://www.semanticweb.org/IoTUDresden#
        DigitalStromController”/>
</ClassAssertion>
<ClassAssertion>
    <Class IRI=”http://www.semanticweb.org/IoTUDresden#Actuator”/>
    <NamedIndividual IRI=”http://www.semanticweb.org/IoTUDresden#
        DoorActuator”/>
```

```
</ClassAssertion>
<ClassAssertion>
    <Class IRI=”http://www.semanticweb.org/IoTUDresden#Device”/>
    <NamedIndividual IRI=”http://www.semanticweb.org/IoTUDresden#
        DoorActuator”/>
</ClassAssertion>
<ClassAssertion>
    <Class IRI=”http://www.semanticweb.org/IoTUDresden#Thing”/>
    <NamedIndividual IRI=”http://www.semanticweb.org/IoTUDresden#
        DoorActuator”/>
</ClassAssertion>
<ClassAssertion>
    <Class IRI=”http://www.semanticweb.org/IoTUDresden#Role”/>
    <NamedIndividual IRI=”http://www.semanticweb.org/IoTUDresden#
        DoorOpener”/>
</ClassAssertion>
<ClassAssertion>
    <Class IRI=”http://www.semanticweb.org/IoTUDresden#PhysicalProperty”/>
    <NamedIndividual IRI=”http://www.semanticweb.org/IoTUDresden#
        DoorStatus”/>
</ClassAssertion>
<ClassAssertion>
    <Class IRI=”http://www.semanticweb.org/IoTUDresden#PhysicalEntity”/>
    <NamedIndividual IRI=”http://www.semanticweb.org/IoTUDresden#
        FrontDoor”/>
</ClassAssertion>
<ClassAssertion>
    <Class IRI=”http://www.semanticweb.org/IoTUDresden#Command”/>
    <NamedIndividual IRI=”http://www.semanticweb.org/IoTUDresden#
        GetIdentification”/>
</ClassAssertion>
<ClassAssertion>
    <Class IRI=”http://www.semanticweb.org/IoTUDresden#GetDataCommand”/
        >
    <NamedIndividual IRI=”http://www.semanticweb.org/IoTUDresden#
        GetIdentification”/>
</ClassAssertion>
<ClassAssertion>
    <Class IRI=”http://www.semanticweb.org/IoTUDresden#Thing”/>
    <NamedIndividual IRI=”http://www.semanticweb.org/IoTUDresden#
        KeyMaticLock”/>
</ClassAssertion>
```

```
<ClassAssertion>
    <Class IRI=”http://www.semanticweb.org/IoTUDresden#PhysicalEntity”/>
    <NamedIndividual IRI=”http://www.semanticweb.org/IoTUDresden#Kitchen”/>
</ClassAssertion>
<ClassAssertion>
    <Class IRI=”http://www.semanticweb.org/IoTUDresden#Thing”/>
    <NamedIndividual IRI=”http://www.semanticweb.org/IoTUDresden#KodiPlayer”/>
</ClassAssertion>
<ClassAssertion>
    <Class IRI=”http://www.semanticweb.org/IoTUDresden#Thing”/>
    <NamedIndividual IRI=”http://www.semanticweb.org/IoTUDresden#LibeliumHealthKit”/>
</ClassAssertion>
<ClassAssertion>
    <Class IRI=”http://www.semanticweb.org/IoTUDresden#PhysicalEntity”/>
    <NamedIndividual IRI=”http://www.semanticweb.org/IoTUDresden#LivingRoom”/>
</ClassAssertion>
<ClassAssertion>
    <Class IRI=”http://www.semanticweb.org/IoTUDresden#Role”/>
    <NamedIndividual IRI=”http://www.semanticweb.org/IoTUDresden#MusicPlayer”/>
</ClassAssertion>
<ClassAssertion>
    <Class IRI=”http://www.semanticweb.org/IoTUDresden#Thing”/>
    <NamedIndividual IRI=”http://www.semanticweb.org/IoTUDresden#NFCReader”/>
</ClassAssertion>
<ClassAssertion>
    <Class IRI=”http://www.semanticweb.org/IoTUDresden#Device”/>
    <NamedIndividual IRI=”http://www.semanticweb.org/IoTUDresden#NFCSensor”/>
</ClassAssertion>
<ClassAssertion>
    <Class IRI=”http://www.semanticweb.org/IoTUDresden#Sensor”/>
    <NamedIndividual IRI=”http://www.semanticweb.org/IoTUDresden#NFCSensor”/>
</ClassAssertion>
<ClassAssertion>
    <Class IRI=”http://www.semanticweb.org/IoTUDresden#Thing”/>
    <NamedIndividual IRI=”http://www.semanticweb.org/IoTUDresden#
        NFCSensor”/>
</ClassAssertion>
<ClassAssertion>
    <Class IRI=”http://www.semanticweb.org/IoTUDresden#Role”/>
    <NamedIndividual IRI=”http://www.semanticweb.org/IoTUDresden#
        PersonAuthenticator”/>
</ClassAssertion>
<ClassAssertion>
    <Class IRI=”http://www.semanticweb.org/IoTUDresden#Actuator”/>
    <NamedIndividual IRI=”http://www.semanticweb.org/IoTUDresden#
        PowerSupplySwitch”/>
</ClassAssertion>
<ClassAssertion>
    <Class IRI=”http://www.semanticweb.org/IoTUDresden#Device”/>
    <NamedIndividual IRI=”http://www.semanticweb.org/IoTUDresden#
        PowerSupplySwitch”/>
</ClassAssertion>
<ClassAssertion>
    <Class IRI=”http://www.semanticweb.org/IoTUDresden#Thing”/>
    <NamedIndividual IRI=”http://www.semanticweb.org/IoTUDresden#
        PowerSupplySwitch”/>
</ClassAssertion>
<ClassAssertion>
    <Class IRI=”http://www.semanticweb.org/IoTUDresden#PhysicalEntity”/>
    <NamedIndividual IRI=”http://www.semanticweb.org/IoTUDresden#
        ResidentAlice”/>
</ClassAssertion>
<ClassAssertion>
    <Class IRI=”http://www.semanticweb.org/IoTUDresden#PhysicalContext”/>
    <NamedIndividual IRI=”http://www.semanticweb.org/IoTUDresden#
        ResidentBob”/>
</ClassAssertion>
<ClassAssertion>
    <Class IRI=”http://www.semanticweb.org/IoTUDresden#PhysicalEntity”/>
    <NamedIndividual IRI=”http://www.semanticweb.org/IoTUDresden#
        ResidentBob”/>
</ClassAssertion>
<ClassAssertion>
    <Class IRI=”http://www.semanticweb.org/IoTUDresden#PhysicalEntity”/>
    <NamedIndividual IRI=”http://www.semanticweb.org/IoTUDresden#
        SmartHome”/>
```

```
</ClassAssertion>
<ClassAssertion>
    <Class IRI=”http://www.semanticweb.org/IoTUDresden#PhysicalContext”/>
    <NamedIndividual IRI=”http://www.semanticweb.org/IoTUDresden#
        SmartHomeContext”/>
</ClassAssertion>
<ClassAssertion>
    <Class IRI=”http://www.semanticweb.org/IoTUDresden#PhysicalEntity”/>
    <NamedIndividual IRI=”http://www.semanticweb.org/IoTUDresden#
        SmartHomeContext”/>
</ClassAssertion>
<ClassAssertion>
    <Class IRI=”http://www.semanticweb.org/IoTUDresden#Thing”/>
    <NamedIndividual IRI=”http://www.semanticweb.org/IoTUDresden#
        SmartPhone”/>
</ClassAssertion>
<ClassAssertion>
    <Class IRI=”http://www.semanticweb.org/IoTUDresden#Thing”/>
    <NamedIndividual IRI=”http://www.semanticweb.org/IoTUDresden#
        SmartWatch”/>
</ClassAssertion>
<ClassAssertion>
    <Class IRI=”http://www.semanticweb.org/IoTUDresden#PhysicalEntity”/>
    <NamedIndividual IRI=”http://www.semanticweb.org/IoTUDresden#Stove”/>
</ClassAssertion>
<ClassAssertion>
    <Class IRI=”http://www.semanticweb.org/IoTUDresden#Role”/>
    <NamedIndividual IRI=”http://www.semanticweb.org/IoTUDresden#
        StoveController”/>
</ClassAssertion>
<ClassAssertion>
    <Class IRI=”http://www.semanticweb.org/IoTUDresden#PhysicalProperty”/>
    <NamedIndividual IRI=”http://www.semanticweb.org/IoTUDresden#
        StoveStatus”/>
</ClassAssertion>
<ClassAssertion>
    <Class IRI=”http://www.semanticweb.org/IoTUDresden#Command”/>
    <NamedIndividual IRI=”http://www.semanticweb.org/IoTUDresden#closeDoor
        ”/>
</ClassAssertion>
<ClassAssertion>
    <Class IRI=”http://www.semanticweb.org/IoTUDresden#OffCommand”/>
    <NamedIndividual IRI=”http://www.semanticweb.org/IoTUDresden#closeDoor
        ”/>
</ClassAssertion>
<ClassAssertion>
    <Class IRI=”http://www.semanticweb.org/IoTUDresden#Command”/>
    <NamedIndividual IRI=”http://www.semanticweb.org/IoTUDresden#openDoor
        ”/>
</ClassAssertion>
<ClassAssertion>
    <Class IRI=”http://www.semanticweb.org/IoTUDresden#OnCommand”/>
    <NamedIndividual IRI=”http://www.semanticweb.org/IoTUDresden#openDoor
        ”/>
</ClassAssertion>
<ClassAssertion>
    <Class IRI=”http://www.semanticweb.org/IoTUDresden#Command”/>
    <NamedIndividual IRI=”http://www.semanticweb.org/IoTUDresden#playAudio
        ”/>
</ClassAssertion>
<ClassAssertion>
    <Class IRI=”http://www.semanticweb.org/IoTUDresden#OnCommand”/>
    <NamedIndividual IRI=”http://www.semanticweb.org/IoTUDresden#playAudio
        ”/>
</ClassAssertion>
<ClassAssertion>
    <Class IRI=”http://www.semanticweb.org/IoTUDresden#Capability”/>
    <NamedIndividual IRI=”http://www.semanticweb.org/IoTUDresden#
        prohibitAccessToHome”/>
</ClassAssertion>
<ClassAssertion>
    <Class IRI=”http://www.semanticweb.org/IoTUDresden#Capability”/>
    <NamedIndividual IRI=”http://www.semanticweb.org/IoTUDresden#
        provideAccessToHome”/>
</ClassAssertion>
<ClassAssertion>
    <Class IRI=”http://www.semanticweb.org/IoTUDresden#Capability”/>
    <NamedIndividual IRI=”http://www.semanticweb.org/IoTUDresden#
        provideAudio”/>
</ClassAssertion>
<ClassAssertion>
    <Class IRI=”http://www.semanticweb.org/IoTUDresden#Capability”/>
    <NamedIndividual IRI=”http://www.semanticweb.org/IoTUDresden#
        provideStoveControl”/>
```

```
</ClassAssertion>
<ClassAssertion>
    <Class IRI=”http://www.semanticweb.org/IoTUDresden#Command”/>
    <NamedIndividual IRI=”http://www.semanticweb.org/IoTUDresden#stopAudio
        ”/>
</ClassAssertion>
<ClassAssertion>
    <Class IRI=”http://www.semanticweb.org/IoTUDresden#OffCommand”/>
    <NamedIndividual IRI=”http://www.semanticweb.org/IoTUDresden#stopAudio
        ”/>
</ClassAssertion>
<ClassAssertion>
    <Class IRI=”http://www.semanticweb.org/IoTUDresden#Command”/>
    <NamedIndividual IRI=”http://www.semanticweb.org/IoTUDresden#
        switchOffStove”/>
</ClassAssertion>
<ClassAssertion>
    <Class IRI=”http://www.semanticweb.org/IoTUDresden#OffCommand”/>
    <NamedIndividual IRI=”http://www.semanticweb.org/IoTUDresden#
        switchOffStove”/>
</ClassAssertion>
<ClassAssertion>
    <Class IRI=”http://www.semanticweb.org/IoTUDresden#Command”/>
    <NamedIndividual IRI=”http://www.semanticweb.org/IoTUDresden#
        switchOnStove”/>
</ClassAssertion>
<ClassAssertion>
    <Class IRI=”http://www.semanticweb.org/IoTUDresden#OnCommand”/>
    <NamedIndividual IRI=”http://www.semanticweb.org/IoTUDresden#
        switchOnStove”/>
</ClassAssertion>
<InverseObjectProperties>
    <ObjectProperty IRI=”http://www.semanticweb.org/IoTUDresden#affects”/>
    <ObjectProperty IRI=”http://www.semanticweb.org/IoTUDresden#monitors”/
        >
</InverseObjectProperties>
<InverseObjectProperties>
    <ObjectProperty abbreviatedIRI=”owl:topObjectProperty”/>
    <ObjectProperty abbreviatedIRI=”owl:topObjectProperty”/>
</InverseObjectProperties>
<SymmetricObjectProperty>
    <ObjectProperty abbreviatedIRI=”owl:topObjectProperty”/>
    </SymmetricObjectProperty>
    <TransitiveObjectProperty>
        <ObjectProperty IRI=”http://www.semanticweb.org/IoTUDresden#contains”/
            >
    </TransitiveObjectProperty>
    <TransitiveObjectProperty>
        <ObjectProperty IRI=”http://www.semanticweb.org/IoTUDresden#
            manipulates”/>
    </TransitiveObjectProperty>
    <TransitiveObjectProperty>
        <ObjectProperty abbreviatedIRI=”owl:topObjectProperty”/>
    </TransitiveObjectProperty>
</Ontology>
```

Bibliography

[AAD$^+$07] Ashish Agrawal, Mike Amend, Manoj Das, Mark Ford, Chris Keller, Matthias Kloppmann, Dieter König, Frank Leymann, Ralf Müller, Gerhard Pfau, et al. Web services human task (ws-humantask). *White Paper*, 2007.

[ACD$^+$03] Tony Andrews, Francisco Curbera, Hitesh Dholakia, Yaron Goland, Johannes Klein, Frank Leymann, Kevin Liu, Dieter Roller, Doug Smith, Satish Thatte, et al. Business process execution language for web services, 2003.

[ADG09] Raian Ali, Fabiano Dalpiaz, and Paolo Giorgini. A goal modeling framework for self-contextualizable software. In *Enterprise, Business-Process and Information Systems Modeling*, pages 326–338. Springer, 2009.

[AFFB13] Stefan Appel, Sebastian Frischbier, Tobias Freudenreich, and Alejandro Buchmann. Event stream processing units in business processes. In *Business Process Management*, pages 187–202. Springer, 2013.

[AIM10] Luigi Atzori, Antonio Iera, and Giacomo Morabito. The internet of things: A survey. *Computer networks*, 54(15):2787–2805, 2010.

[AKF$^+$14] Stefan Appel, Pascal Kleber, Sebastian Frischbier, Tobias Freudenreich, and Alejandro Buchmann. Modeling and execution of event stream processing in business processes. *Information Systems*, 46:140–156, 2014.

[All03] OSGi Alliance. *Osgi service platform, release 3*. IOS Press, Inc., 2003.

[BBDL$^+$13] Martin Bauer, Nicola Bui, Jourik De Loof, Carsten Magerkurth, Andreas Nettsträter, Julinda Stefa, and Joachim W Walewski.

Iot reference model. In *Enabling Things to Talk*, pages 113–162. Springer, 2013.

[BC08] Dario Bonino and Fulvio Corno. Dogont-ontology modeling for intelligent domotic environments. In *International Semantic Web Conference*, pages 790–803. Springer, 2008.

[BD77] Charles W Bachman and Manilal Daya. The role concept in data models. In *Proceedings of the third international conference on Very large data bases-Volume 3*, pages 464–476. VLDB Endowment, 1977.

[BG14] Patrick Brézillon and Avelino J Gonzalez. *Context in Computing: A Cross-Disciplinary Approach for Modeling the Real World.* Springer, 2014.

[BGK⁺11] Christian Brand, Matthias Gorning, Tim Kaiser, Jürgen Pasch, and Michael Wenz. Development of high-quality graphical model editors. *Eclipse Magazine*, 2011.

[BHK⁺08] Matthias Born, Jörg Hoffmann, Tomasz Kaczmarek, Marek Kowalkiewicz, Ivan Markovic, James Scicluna, Ingo Weber, and Xuan Zhou. Semantic annotation and composition of business processes with maestro. In *European Semantic Web Conference*, pages 772–776. Springer, 2008.

[BLHL⁺01] Tim Berners-Lee, James Hendler, Ora Lassila, et al. The semantic web. *Scientific american*, 284(5):28–37, 2001.

[Bor14] Eleonora Borgia. The internet of things vision: Key features, applications and open issues. *Computer Communications*, 54:1–31, 2014.

[BPG⁺04] Paolo Bresciani, Anna Perini, Paolo Giorgini, Fausto Giunchiglia, and John Mylopoulos. Tropos: An agent-oriented software development methodology. *Autonomous Agents and Multi-Agent Systems*, 8(3):203–236, 2004.

[BS11] Debasis Bandyopadhyay and Jaydip Sen. Internet of things: Applications and challenges in technology and standardization. *Wireless Personal Communications*, 58(1):49–69, 2011.

[BV07] Thomas Bernhardt and Alexandre Vasseur. Esper: Event stream processing and correlation. *ONJava, in http://www. onjava. com/lpt/a/6955, OReilly*, 2007.

[CDD$^+$12] Fabio Casati, Florian Daniel, Guenadi Dantchev, Joakim Eriksson, Niclas Finne, Stamatis Karnouskos, Patricio Moreno Montero, Luca Mottola, Felix Jonathan Oppermann, Gian Pietro Picco, et al. Towards business processes orchestrating the physical enterprise with wireless sensor networks. In *Proceedings of the 34th International Conference on Software Engineering*, pages 1357–1360. IEEE Press, 2012.

[CKM02] Jaelson Castro, Manuel Kolp, and John Mylopoulos. Towards requirements-driven information systems engineering: the tropos project. *Information systems*, 27(6):365–389, 2002.

[CSB16] Chii Chang, Satish Narayana Srirama, and Rajkumar Buyya. Mobile cloud business process management system for the internet of things: a survey. *ACM Computing Surveys (CSUR)*, 49(4):70, 2016.

[CSM15] Chii Chang, Satish Narayana Srirama, and Jakob Mass. A middleware for discovering proximity-based service-oriented industrial internet of things. In *Services Computing (SCC), 2015 IEEE International Conference on*, pages 130–137. IEEE, 2015.

[Dee98] Stephen E Deering. Internet protocol, version 6 (ipv6) specification. 1998.

[Dey01] Anind K Dey. Understanding and using context. *Personal and ubiquitous computing*, 5(1):4–7, 2001.

[DPB17] Flávia C Delicato, Paulo F Pires, and Thais Batista. Resource management for internet of things, 2017.

[DPS$^+$08] S Dagtas, G Pekhteryev, Z Sahinoglu, H Cam, and N Challa. Realtime and secure wireless health monitoring. *International Journal of Telemedicine and Applications*, 2008:1, 2008.

[DSR15] Maryam Davoudpour, Alireza Sadeghian, and Hossein Rahnama. canthings(context aware network for the design of connected things) service modeling based on timed cpn. In *Semantic Computing (ICSC), 2015 IEEE International Conference on*, pages 127–130. IEEE, 2015.

[DTB$^+$15] Kashif Dar, Amir Taherkordi, Harun Baraki, Frank Eliassen, and Kurt Geihs. A resource oriented integration architecture for the internet of things: A business process perspective. *Pervasive and Mobile Computing*, 20:145–159, 2015.

[DV08] Adam Dunkels and Jean-Philippe Vasseur. Ip for smart objects. *Ipso alliance white paper*, 1, 2008.

[FLM$^+$09] Dirk Fahland, Daniel Lübke, Jan Mendling, Hajo Reijers, Barbara Weber, Matthias Weidlich, and Stefan Zugal. Declarative versus imperative process modeling languages: The issue of understandability. In *Enterprise, Business-Process and Information Systems Modeling*, pages 353–366. Springer, 2009.

[FM11] Ian Fette and Alexey Melnikov. The websocket protocol, 2011.

[GBMP13] Jayavardhana Gubbi, Rajkumar Buyya, Slaven Marusic, and Marimuthu Palaniswami. Internet of things (iot): A vision, architectural elements, and future directions. *Future generation computer systems*, 29(7):1645–1660, 2013.

[GCFP10] Pau Giner, Carlos Cetina, Joan Fons, and Vicente Pelechano. Developing mobile workflow support in the internet of things. *IEEE Pervasive Computing*, 9(2):18–26, 2010.

[GdV98] Paul Grefen and Remmert Remmerts de Vries. A reference architecture for workflow management systems. *Data & Knowledge Engineering*, 27(1):31 – 57, 1998.

[GEPF11] Nils Glombitza, Sebastian Ebers, Dennis Pfisterer, and Stefan Fischer. Using bpel to realize business processes for an internet of things. In *International Conference on Ad-Hoc Networks and Wireless*, pages 294–307. Springer, 2011.

[GKGK16] Imen Graja, Slim Kallel, Nawal Guermouche, and Ahmed Hadj Kacem. Bpmn4cps: A bpmn extension for modeling cyber-physical systems. In *Enabling Technologies: Infrastructure for Collaborative*

Enterprises (WETICE), 2016 IEEE 25th International Conference on, pages 152–157. IEEE, 2016.

[GMNS03] Paolo Giorgini, John Mylopoulos, Eleonora Nicchiarelli, and Roberto Sebastiani. Formal reasoning techniques for goal models. In *Journal on Data Semantics I*, pages 1–20. Springer, 2003.

[GMS05] Paolo Giorgini, John Mylopoulos, and Roberto Sebastiani. Goal-oriented requirements analysis and reasoning in the tropos methodology. *Engineering Applications of Artificial Intelligence*, 18(2):159–171, 2005.

[Gua92] Nicola Guarino. Concepts, attributes and arbitrary relations: some linguistic and ontological criteria for structuring knowledge bases. *Data & Knowledge Engineering*, 8(3):249–261, 1992.

[HH95] David Hollingsworth and UK Hampshire. Workflow management coalition: The workflow reference model. *Document Number TC00-1003*, 19, 1995.

[HSK$^+$16] Steffen Huber, Ronny Seiger, Andr Khnert, Vasileios Theodorou, and Thomas Schlegel. Goal-based semantic queries for dynamic processes in the internet of things. *International Journal of Semantic Computing*, 10(02):269–293, 2016.

[HSKS16] Steffen Huber, Ronny Seiger, André Kühnert, and Thomas Schlegel. A context-adaptive workflow engine for humans, things and services. In *Proceedings of the 2016 ACM International Joint Conference on Pervasive and Ubiquitous Computing: Adjunct*, pages 285–288. ACM, 2016.

[HSS16] S. Huber, R. Seiger, and T. Schlegel. Using semantic queries to enable dynamic service invocation for processes in the internet of things. In *Semantic Computing (ICSC), 2016 IEEE International Conference on*, pages 214–221, Feb 2016.

[IBGB16] Valérie Issarny, Georgios Bouloukakis, Nikolaos Georgantas, and Benjamin Billet. Revisiting service-oriented architecture for the iot: A middleware perspective. In *International Conference on Service-Oriented Computing*, pages 3–17. Springer, 2016.

[INF08] D INFSO. Networked enterprise & rfid infso g. 2 micro & nanosystems, in co-operation with the working group rfid of the etp eposs, internet of things in 2020, roadmap for the future [r]. *Information Society and Media, Tech. Rep*, 2008.

[KBRA16] Thomas Kühn, Kay Bierzynski, Sebastian Richly, and Uwe Aßmann. Framed: full-fledge role modeling editor (tool demo). In *Proceedings of the 2016 ACM SIGPLAN International Conference on Software Language Engineering*, pages 132–136. ACM, 2016.

[KG04] R Krikorian and N Gershenfeld. Internet 0inter-device internetworking. *BT technology journal*, 22(4):278–284, 2004.

[KKL$^+$05] Matthias Kloppmann, Dieter Koenig, Frank Leymann, Gerhard Pfau, Alan Rickayzen, Claus von Riegen, Patrick Schmidt, and Ivana Trickovic. Ws-bpel extension for people–bpel4people. *Joint white paper, IBM and SAP*, 183:184, 2005.

[KL04] Evangelia Kavakli and Pericles Loucopoulos. Goal modeling in requirements engineering: Analysis and critique. *Information Modeling Methods and Methodologies: Advanced Topics in Database Research: Advanced Topics in Database Research*, 102, 2004.

[KLG$^+$14] Thomas Kühn, Max Leuthäuser, Sebastian Götz, Christoph Seidl, and Uwe Aßmann. A metamodel family for role-based modeling and programming languages. In *International Conference on Software Language Engineering*, pages 141–160. Springer, 2014.

[KNM10] Reto Krummenacher, Barry Norton, and Adrian Marte. Towards linked open services and processes. In *Future internet symposium*, pages 68–77. Springer, 2010.

[KRG$^+$15] Alireza Khoshkbarforoushha, Rajiv Ranjan, Raj Gaire, Prem P Jayaraman, John Hosking, and Ehsan Abbasnejad. Resource usage estimation of data stream processing workloads in datacenter clouds. *arXiv preprint arXiv:1501.07020*, 2015.

[KSS$^+$10] Stamatis Karnouskos, Domnic Savio, Patrik Spiess, Dominique Guinard, Vlad Trifa, and Oliver Baecker. Real-world service interaction with enterprise systems in dynamic manufacturing environments. In *Artificial Intelligence Techniques for Networked Manufacturing Enterprises Management*, pages 423–457. Springer, 2010.

[Lee12] Sunguk Lee. Unified modeling language (uml) for database systems and computer applications. *International Journal of Database Theory and Application*, 5(1):157–164, 2012.

[Lok03] Seng Loke. Service-oriented device ecology workflows. *Service-Oriented Computing-ICSOC 2003*, pages 559–574, 2003.

[LYS07] Minsoo Lee, Hyejung Yoon, and Hyoseop Shin. Supporting dynamic workflows in a ubiquitous environment. In *Multimedia and Ubiquitous Engineering, 2007. MUE'07. International Conference on*, pages 272–277. IEEE, 2007.

[MPPM15] Sven Mielke, Martin Pelke, Sebastian Pospiech, and Robert Mertens. Flexible semantic query expansion for process exploration. In *Semantic Computing (ICSC), 2015 IEEE International Conference on*, pages 440–443. IEEE, 2015.

[MRH15] Sonja Meyer, Andreas Ruppen, and Lorenz Hilty. The things of the internet of things in bpmn. In *International Conference on Advanced Information Systems Engineering*, pages 285–297. Springer, 2015.

[MRM13] Sonja Meyer, Andreas Ruppen, and Carsten Magerkurth. Internet of things-aware process modeling: Integrating IoT devices as business process resources. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 7908 LNCS:84–98, 2013.

[MS14a] Sunilkumar S Manvi and Gopal Krishna Shyam. Resource management for infrastructure as a service (iaas) in cloud computing: A survey. *Journal of Network and Computer Applications*, 41:424–440, 2014.

[MS14b] Pradeep K Murukannaiah and Munindar P Singh. Xipho: Extending tropos to engineer context-aware personal agents. In *Proceedings of the 2014 international conference on Autonomous agents and multi-agent systems*, pages 309–316. International Foundation for Autonomous Agents and Multiagent Systems, 2014.

[MSDPC12] Daniele Miorandi, Sabrina Sicari, Francesco De Pellegrini, and Imrich Chlamtac. Internet of things: Vision, applications and research challenges. *Ad Hoc Networks*, 10(7):1497–1516, 2012.

[MSMP11] Sonja Meyer, Klaus Sperner, Carsten Magerkurth, and Jacques Pasquier. Towards modeling real-world aware business processes. In *Proceedings of the Second International Workshop on Web of Things*, page 8. ACM, 2011.

[Mul07] Geoff Mulligan. The 6lowpan architecture. In *Proceedings of the 4th workshop on Embedded networked sensors*, pages 78–82. ACM, 2007.

[MW17] Incorporated Merriam-Webster. The merriam-webster dictionary, 2017.

[NWVL07] Jörg Nitzsche, Daniel Wutke, and Tammo Van Lessen. An ontology for executable business processes. In *SBPM*, 2007.

[OMSJ06] Chris Otto, Aleksandar Milenkovic, Corey Sanders, and Emil Jovanov. System architecture of a wireless body area sensor network for ubiquitous health monitoring. *Journal of mobile multimedia*, 1(4):307–326, 2006.

[Pan16] Kasey Panetta. 7 technologies underpin the hype cycle for the internet of things, 2016, 2016.

[PP12] Federica Paganelli and David Parlanti. A dht-based discovery service for the internet of things. *Journal of Computer Networks and Communications*, 2012, 2012.

[Pre17] Cambridge University Press. Cambridge dictionary, 2017.

[PRS$^+$13] Tao Peng, Marco Ronchetti, Jovan Stevovic, Annamaria Chiasera, and Giampaolo Armellin. Business process assignment and execution from cloud to mobile. In *International Conference on Business Process Management*, pages 264–276. Springer, 2013.

[PZCG14] Charith Perera, Arkady Zaslavsky, Peter Christen, and Dimitrios Georgakopoulos. Context aware computing for the internet of things: A survey. *IEEE Communications Surveys & Tutorials*, 16(1):414–454, 2014.

[QCG⁺09] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y Ng. Ros: an open-source robot operating system. In *ICRA workshop on open source software*, volume 3, page 5, 2009.

[RJB04] James Rumbaugh, Ivar Jacobson, and Grady Booch. *Unified modeling language reference manual, the.* Pearson Higher Education, 2004.

[RSDS12] Michele Ruta, Floriano Scioscia, and Eugenio Di Sciascio. Enabling the semantic web of things: Framework and architecture. In *Semantic Computing (ICSC), 2012 IEEE Sixth International Conference on*, pages 345–347. IEEE, 2012.

[RTHEvdA04a] Nick Russell, Arthur HM Ter Hofstede, David Edmond, and Wil MP van der Aalst. Workflow data patterns. 2004.

[RTHEvdA04b] Nick Russell, Arthur HM Ter Hofstede, David Edmond, and Wil MP van der Aalst. Workflow resource patterns. Technical report, BETA Working Paper Series, WP 127, Eindhoven University of Technology, Eindhoven, 2004.

[SBMP08] Dave Steinberg, Frank Budinsky, Ed Merks, and Marcelo Paternostro. *EMF: eclipse modeling framework.* Pearson Education, 2008.

[Sch09] Thomas Schlegel. Object-oriented interactive processes in decentralized production systems. In MichaelJ. Smith and Gavriel Salvendy, editors, *Human Interface and the Management of Information. Designing Information Environments*, volume 5617 of *Lecture Notes in Computer Science*, pages 296–305. Springer Berlin Heidelberg, 2009.

[SGFW10] Harald Sundmaeker, Patrick Guillemin, Peter Friess, and Sylvie Woelfflé. Vision and challenges for realising the internet of things. *Cluster of European Research Projects on the Internet of Things, European Commision*, 2010.

[SGM04] Roberto Sebastiani, Paolo Giorgini, and John Mylopoulos. Simple and minimum-cost satisfiability for goal models. In *International*

Conference on Advanced Information Systems Engineering, pages 20–35. Springer, 2004.

[SH07] Holger Schmidt and Franz J Hauck. Samproc: middleware for self-adaptive mobile processes in heterogeneous ubiquitous environments. In *Proceedings of the 4th on Middleware doctoral symposium*, page 11. ACM, 2007.

[SHA17] Ronny Seiger, Stefan Herrmann, and Uwe Aßmann. Self-healing for distributed workflows in the internet of things. In *IEEE International Conference on Software Architecture (ICSA 2017) Workshops*, 2017.

[SHHA16] Ronny Seiger, Steffen Huber, Peter Heisig, and Uwe Assmann. Enabling self-adaptive workflows for cyber-physical systems. In *International Workshop on Business Process Modeling, Development and Support*, pages 3–17. Springer International Publishing, 2016.

[SHS15] Ronny Seiger, Steffen Huber, and Thomas Schlegel. Proteus: An integrated system for process execution in cyber-physical systems. In Khaled Gaaloul, Rainer Schmidt, Selmin Nurcan, Sergio Guerreiro, and Qin Ma, editors, *Enterprise, Business-Process and Information Systems Modeling*, volume 214 of *Lecture Notes in Business Information Processing*, pages 265–280. 2015.

[SHS16] Ronny Seiger, Steffen Huber, and Thomas Schlegel. Toward an execution system for self-healing workflows in cyber-physical systems. *Software & Systems Modeling*, pages 1–22, 2016.

[SKNS14] Ronny Seiger, Christine Keller, Florian Niebling, and Thomas Schlegel. Modelling complex and flexible processes for smart cyber-physical environments. *Journal of Computational Science*, 2014.

[SMR$^+$08] Helen Schonenberg, Ronny Mans, Nick Russell, Nataliya Mulyar, and Wil van der Aalst. Process flexibility: A survey of contemporary approaches. In *Advances in Enterprise Engineering I*, pages 16–30. Springer, 2008.

[SNS14] Ronny Seiger, Florian Niebling, and Thomas Schlegel. A distributed execution environment enabling resilient processes for ubiquitous systems. In *Pervasive Computing and Communications Workshops (PERCOM Workshops), 2014 IEEE International Conference on*, pages 220–223, March 2014.

[SRN15] Oumaima Saidani, Colette Rolland, and Selmin Nurcan. Towards a generic context model for bpm. In *System Sciences (HICSS), 2015 48th Hawaii International Conference on*, pages 4120–4129. IEEE, 2015.

[SSMS14] Ronny Seiger, Susann Struwe, Sandra Matthes, and Thomas Schlegel. A resilient interaction concept for process management on tabletops for cyber-physical systems. In *International Conference on Human Interface and the Management of Information*, pages 347–358. Springer, 2014.

[SSOK13] C Timurhan Sungur, Patrik Spiess, Nina Oertel, and Oliver Kopp. Extending BPMN for Wireless Sensor Networks. *2013 IEEE 15th Conference on Business Informatics*, pages 109–116, 2013.

[ST05] Pinar Senkul and Ismail H Toroslu. An architecture for workflow scheduling under resource allocation constraints. *Information Systems*, 30(5):399–422, 2005.

[Ste00] Friedrich Steimann. On the representation of roles in object-oriented and conceptual modelling. *Data & Knowledge Engineering*, 35(1):83–106, 2000.

[SZZ14] L Smirek, G Zimmermann, and D Ziegler. Towards universally usable smart homes-how can myui, urc and openhab contribute to an adaptive user interface platform. In *IARIA Conference, Nice, France*, pages 29–38, 2014.

[TMS$^+$12] Matthias Thoma, Sonja Meyer, Klaus Sperner, Stefan Meissner, and Torsten Braun. On iot-services: Survey, classification and enterprise integration. In *Green Computing and Communications (GreenCom), 2012 IEEE International Conference on*, pages 257–260. IEEE, 2012.

[TW87] Andrew S Tanenbaum and Albert S Woodhull. *Operating systems: design and implementation*, volume 2. Prentice-Hall Englewood Cliffs, NJ, 1987.

[VDAL08] Wil MP Van Der Aalst and Kristian Bisgaard Lassen. Translating unstructured workflow processes to readable bpel: Theory and implementation. *Information and Software Technology*, 50(3):131–159, 2008.

[vdAtH05] W.M.P. van der Aalst and A.H.M. ter Hofstede. Yawl: yet another workflow language. *Information Systems*, 30(4):245 – 275, 2005.

[vDATHKB03] Wil MP van Der Aalst, Arthur HM Ter Hofstede, Bartek Kiepuszewski, and Alistair P Barros. Workflow patterns. *Distributed and parallel databases*, 14(1):5–51, 2003.

[VDATHW03] Wil MP Van Der Aalst, Arthur HM Ter Hofstede, and Mathias Weske. Business process management: A survey. In *International conference on business process management*, pages 1–12. Springer, 2003.

[VDAVH04] Wil Van Der Aalst and Kees Max Van Hee. *Workflow management: models, methods, and systems*. MIT press, 2004.

[vdM15] Rob van der Meulen. Press release - gartner says 6.4 billion connected "things" will be in use in 2016, up 30 percent from 2015, 2015.

[Wei91] Mark Weiser. The computer for the 21st century. *Scientific american*, 265(3):94–104, 1991.

[WJ12] Qiang Wei and Zhi Jin. Service discovery for internet of things: a context-awareness perspective. In *Proceedings of the Fourth Asia-Pacific Symposium on Internetware*, page 25. ACM, 2012.

[WSBL15] Matthias Wieland, Holger Schwarz, Uwe Breitenbücher, and Frank Leymann. Towards situation-aware adaptive workflows: Sitopta general purpose situation-aware workflow management system. In *Pervasive Computing and Communication Workshops (PerCom Workshops), 2015 IEEE International Conference on*, pages 32–37. IEEE, 2015.

[WvdAD$^+$06] P. Wohed, W.M.P. van der Aalst, M. Dumas, A.H.M. ter Hofstede, and N. Russell. On the suitability of bpmn for business process modelling. In Schahram Dustdar, JosLuiz Fiadeiro, and AmitP.

Sheth, editors, *Business Process Management*, volume 4102 of *Lecture Notes in Computer Science*, pages 161–176. Springer Berlin Heidelberg, 2006.

[YBSD16] Alaaeddine Yousfi, Christine Bauer, Rajaa Saidi, and Anind K Dey. ubpmn: A bpmn extension for modeling ubiquitous business processes. *Information and Software Technology*, 74:55–68, 2016.

[YdFDS16] Alaaeddine Yousfi, Adrian de Freitas, Anind K Dey, and Rajaa Saidi. The use of ubiquitous computing for business process improvement. *IEEE Transactions on Services Computing*, 9(4):621–632, 2016.

[Zúñ01] Gloria L Zúñiga. Ontology: its transformation from philosophy to information systems. In *Proceedings of the international conference on Formal Ontology in Information Systems-Volume 2001*, pages 187–197. ACM, 2001.