Employing Description Logics in Ambient Intelligence for Modeling and Reasoning about Complex Situations

Thomas Springer and Anni-Yasmin Turhan [a]
[a] *TU Dresden, Faculty of Computer Science, Chair for Computer Networks/Chair for Automata Theory*
E-mail: thomas.springer@tu-dresden.de, turhan@tcs.inf.tu-dresden.de

Abstract.
Ambient Intelligence systems need to represent information about their environment and recognize relevant situations to perform appropriate actions proactively and autonomously. The context information gathered by these systems comes with imperfections such as incompleteness or incorrectness. These characteristics need to be handled gracefully by the Ambient Intelligence system. Moreover, the represented information must allow for a fast and reliable recognition of the current situation.
To solve these problems we propose a method for situation modeling using the Description Logics based ontology language OWL DL and a framework for employing Description Logics reasoning services to recognize the current situation based on context. The benefits from the approach are manifold: the semantics of Description Logics allow for graceful handling of incomplete knowledge. The well-investigated reasoning services do not only allow recognizing the current situation, but also can add to the reliability of the overall system. Moreover optimized reasoning systems are freely available and ready to use.
We underpin the feasibility of our approach by providing a case study based on a smart home application conducting an evaluation of different Description Logics reasoners with respect to our application ontology as well as a discussion of Description Logics systems in Ambient Intelligence.

Keywords: Situation-Awareness, Description Logics, Reasoning services, OWL DL, Modeling context information

1. Introduction

Research on Ubiquitous Computing and especially Ambience Intelligence (AmI) aims at creating systems able to interact in an intelligent way with the environment, especially the user. Weiser characterized this kind of system as: “machines that fit the human environment instead of forcing humans to enter theirs will make using a computer as refreshing as a walk in the woods” [53].

A system able to recognize the environment’s state can adjust its behavior according to that state. For instance, an application for supporting mobile workers during their tasks in the field could adapt the input and output modalities to improve the interaction with the user. Speech input and output could be used if the workers’ hands are not free, or gesture input could be used if the surrounding noise level is very high. In a similar way an assistance application for elderly people could intelligently support planning of daily activities like selecting convenient connections of public transportation systems for carrying out shopping activities, visiting the doctor or meeting relatives or friends.

To realize such systems, they have to be able to capture information about the environment and the involved users based on heterogeneous and distributed information sources, mainly sensors but also extracted application data, user monitoring or other methods for gathering context information. The information captured in this way is usually low-level and has to be aggregated and abstracted to create a higher-level representation of the overall situation a system is currently in. Only the recognition of complex situations enables systems of this kind to operate in a more autonomous, adaptive and intelligent way.

Thus, ambient intelligent systems should be aware of the current situation. We understand *situation-awareness* as the ability of a system to logically aggregate a type of a situation from a complex set of features of the system's context. The derived situation type should be meaningful to the system in the sense that it can adjust its behavior in a defined way to the current situation.

A generalized view on situation-aware systems based on knowledge-based systems as we use it in this paper is depicted in Figure 1 introducing four processing phases. The first phase of the recognition of a situation starts with the capturing of information about the environment using sensor devices. In the second phase, this information is aggregated and abstracted by different operations performed by the sensing devices or components of a context service. The third phase adds the preprocessed data in an adequate format to the knowledge base. Then situation types are inferred from an updated knowledge base. In the fourth phase, the application triggers appropriate actions based on the situation types inferred in phase three.

This process involves approaches from several areas of computer science, namely pervasive and ubiquitous computing, context awareness and artificial intelligence, as it was stated in [40] and [36]. Sensors, actuators and other miniaturized, low-cost computing devices installed in buildings, devices or even clothes and the human body help to capture usually low-level, physical information about the environment like temperature, light intensity, or blood pressure. These technologies are mainly adopted in the first phase. Systems developed for context-awareness aim at generalizing the access to these heterogeneous information sources and to apply technologies for aggregating and abstracting the sensed information to context information usable in applications [18,13,45]. Context models are created to establish a shared understanding between all providers of context information, the context middleware and the context-aware applications [19,48]. The cross-system sharing of context and the modeling of different context features, especially the quality of information are further goals of context models [14,41]. To sum up, context-aware systems provide a set of context information that has to match a context model.

The preprocessing of sensor data is only half the way to situation-aware systems. Classical AI methods come into play in phase two and three of situation-aware systems as described in Figure 1. The second phase uses AI methods as feature extraction and approaches for classification and aggregation of sensed data to obtain higher-level context information, as described in [27] and [32]. In the third phase, different AI techniques could be used to derive a situation type from a complex set of context features. While approaches like Neuronal Networks or Bayesian Networks could be adopted in the whole process from sensor data abstraction up to decision making, we focus on AI systems with explicit representation of knowledge as defined in Figure 1. A major advantage of such systems is the reproducibility of inferences and decisions.

In phase four, the AmI system derives a decision from the recognized situation types. The decoupling of situation recognition and decision making enables a clear separation of concerns and introduces flexibility for the AmI system. Especially, the association between situation types and actions to be triggered can be adjusted at run-time.

In this article, we focus on phase three, the recognition of situation types. This phase comprises the setup of a knowledge base containing a description of the situations to be considered in the application and the reasoning about the situation based on that description involving context as information about the current situation. To this end we discuss the role of context for this kind of systems.

1.1. The role of context in situation-awareness

Context is usually implicit information which has to be sensed or gathered from distributed and heterogeneous sources in order to be usable in applications. Caused by the way context is gathered and as well as by the fact that context reflects the state of a highly dynamic environment, it has special characteristics which influence the modeling and processing of context. A certain piece of information which models a certain aspect of the physical environment is called a *context feature*. We can identify certain characteristics of single context features:

Quality: Context represents a model of aspects of the environment abstracting from the real world. Dependent on the method used for gathering or abstracting context features, they have a certain quality. The quality comprises the relevance of the gathered information in a certain situation or related to a certain entity, the accuracy of a measured value and the probability of a derived context feature.

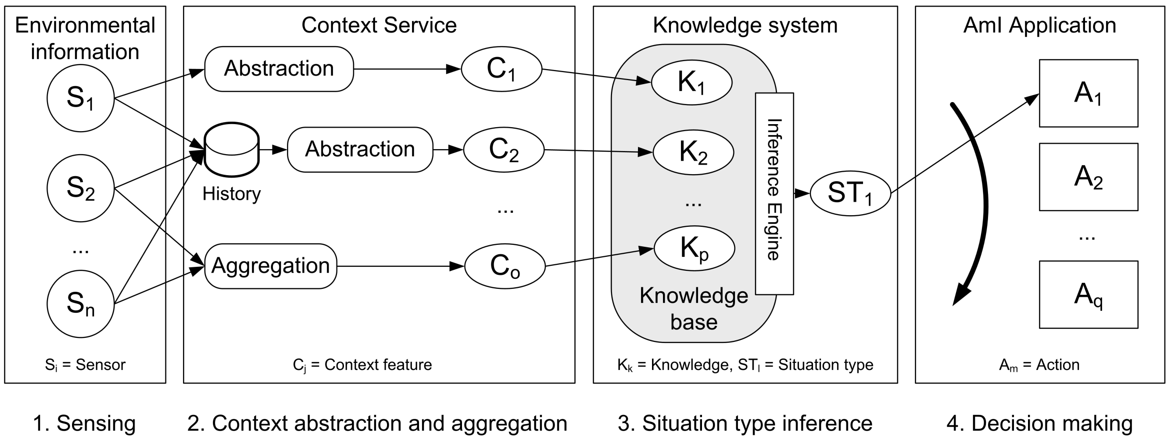

Fig. 1. Conceptual architecture of situation-aware systems.

Incorrectness: Since context features are sensed, extracted or derived, failures in measurement or wrong assumptions for derivation may lead to incorrect information. A system or user can also provide incorrect information to attack context-based authentication and authorization mechanisms.

Multiple sources: Moreover, a certain context feature can be gathered from multiple alternative sources. For instance, location information can be gathered by the client, either using a GPS device, a WLAN-based approach, or by the environment.

Heterogeneous sources: Because of its nature, context is in most cases not explicitly available but has to be gathered from heterogeneous sources. A context source can be any type of sensor system, database, or monitoring component. It can also be derived from user input or application data (see [45]).

In contrast to a single context feature, a set of context features has the following characteristics:

Inconsistency: A set of context features relevant for a certain application can be inconsistent because of contradictory information.

Incompleteness: It can further be incomplete due to the unavailability of certain context sources.

Multidimensional: Context information is multidimensional because it reflects heterogeneous information about the state of a highly dynamic environment comprising physical and technical information as well as personal information like activity, user preferences, social relations or business-related information.

Different abstraction levels: Information from different sources might be available at different levels of abstraction. While a sensor system provides low-level information sensed from the physical environment like temperature or light intensity, application data is usually available on a more abstract level like contact information of a person.

These characteristics of context features require AmI systems that can handle this kind of imperfect information gracefully. More precisely, an AmI system must be able to integrate context information and to deliver a “consistent” view of the current situation. Based on this view its main task is to recognize the type of the current situation in order to invoke the appropriate actions – even if the provided information is incomplete. While context information is usually provided at different abstraction levels, it has to be integrated into the knowledge base seamlessly. In addition, inconsistencies in the knowledge base could occur due to incorrect context and have to be detected by the system. Moreover, AmI systems have to be predictable, reliable and the selection of actions should be transparent to the developer and the user.

In this article, we present an approach to and a framework for modeling of complex situations and inferring the type of the current situation based on context information integrated into the situation model. We use the DL underlying OWL DL as the knowledge representation formalism and realize the task of situation type recognition by the use of Description Logic reasoning services. OWL DL is a formalism with clear semantics that offers reasoning services that remedy the above mentioned problems, especially the handling of incomplete knowledge, the discovery of inconsistencies and the integration of context information at different levels of abstraction. The reasoning services in use are well-investigated. Moreover, sound, complete and terminating reasoning algorithms are available for the Description Logic underlying OWL DL. In addition, these methods are implemented in highly optimized reasoning systems which are freely available. By adopting a standard ontology language and off-the-shelf reasoning tools one can significantly reduce the development overhead and achieve faster prototyping. In addition to the already tested reasoning services of DL reasoners the AmI systems created according to the presented approach are very robust and reliable. Based on a case study and performance measurements we will demonstrate the advantages and limitations of our approach which employs DL systems for the realization of Ambient Intelligence.

In other research work OWL DL is mainly used for two purposes. Firstly, ontologies are created to establish a shared understanding between different components or even systems about the context information which is exchanged between them. Examples are the Standard Ontology for Ubiquitous and Pervasive Applications (SOUPA) and the CONtext ONtology (CONON) [14,19]. In both approaches a common context vocabulary is defined based on a hierarchy of ontologies. An upper ontology defines general terms while domain-specific ontologies define the details for certain application domains. Secondly, context or situation descriptions are employed in combination with different reasoning schemes to derive higher-level context or even situations from context.

Often these solutions adopt rule-based reasoning which could lead to undecidability as it is the case for [14], [15] and [19]. Other approaches for situation recognition adopt OWL DL as the base formalism for the knowledge base, but either adopt the formalism to their needs – as in [2] fuzzy logic and in [37] first order logic was used – or use other reasoning mechanisms than DL systems for the recognition task , e.g. in [47] decision trees were used. Closer to the work presented here comes the approach presented in [28], where TBox classification of OWL DL ontologies was applied to solve situation recognition. ABox realization as the means for recognizing situations is for instance used in [31].

While the use of a standard ontology language and off-the-shelf reasoning tools ensures soundness, completeness and decidability the approach has also some limitations. Situations are described by concepts in the TBox at design time and thus, have to be known to and explicitly modeled by the system developer. Only the context features relevant for situation detection are filled dynamically into the ABox at runtime. Hence, the situations which can be detected by the AmI system are limited to the situations modeled by the system developer at design time. Adding new situation descriptions or changing existing one's means changing the TBox at design time.

Systems like Bayesian networks or decision trees are able to model and especially to learn the associations between context features and situation types from a set of training examples and can thus, also be used to build situation-aware systems. A clear advantage of such approaches is that the developer does not have to specify the complete knowledge about situations manually, since it can be learned by the system based on examples. Major drawbacks of these approaches are that usually a large set of training examples is required to adjust the system and that the situation model is implicit so that system decisions are hardly reproducible.

In addition, probabilistic approaches are able to deal with uncertain knowledge by attaching a probability to all inferred values. With ontologies and Description Logics just facts can be handled.

In the following, we first give an overview of OWL DL, Description Logic and their reasoning services. Then we show how to model complex situations based on OWL DL. Especially we present a systematic modeling method by the composition of basic situations to complex scenarios. Next we describe how to apply DL reasoners for recognizing situation types from context features. By the use of a case study from the smart home domain we demonstrate the feasibility of our approach and give modeling examples. We assess the usefulness of DL systems for realizing AmI systems with respect to the expressivity of the formalism, the handling of imperfect context information and the performance of the DL reasoners and provide an evaluation of today's available Description Logic reasoners. We end the article with a summary and an outlook to future trends.

2. Description Logics for situation-aware systems

Description Logics (DLs) are a family of knowledge representation formalisms. The main asset of DLs is two-fold: on the one hand they are based on formal semantics and on the other they come with powerful reasoning services. These reasoning services make facts that are captured only implicitly in the represented knowledge explicit. Most of the DL reasoning services are nowadays readily available in tools.

Historically, DLs stem from semantic networks [33, 44], which were introduced as a graphical knowledge representation formalism. Early versions of semantic networks lacked a clear definition of the meaning of the formalism. Thus reasoning algorithms developed for this kind of knowledge representation depended on the understanding of the developer. As a consequence implementations of the reasoning algorithms delivered different results for the same semantic network. To remedy this problem DLs were introduced as representation formalisms equipped with formal semantics [6], which then allowed to give formal definitions of their reasoning services. This in turn is the basis for implementations to be used in practical applications, which deliver predictable and reliable results.

DL research focuses on algorithms for DL reasoning services, the analysis of these algorithms in terms of computational complexity and the development of efficient implementations. For instance in case of subsumption tests, algorithms for a whole range of DLs has been investigated, see [7,25,23]. Moreover, there is a collection of very efficient DLs systems available that implement the investigated algorithms in optimized ways, see for instance [21,5,50,43]. These DL systems are employed in many different practical applications. The most prominent application area for DLs is the bio-medical domain, where the huge terminologies are built to represent facts from the biological domain as, for instance, genomic information [8,55] or from the medical domain, such as anatomy facts or medical procedures, see [38,17,16,39]. In the bio-medical domain, the modeling of the domain knowledge in a formal representation language is a benefit in itself, since this formalizes and to some extent standardizes what the community understands about certain terms in an unambiguous way. Thus by agreeing upon an ontology and the definition of concepts in it, a community can create a shared understanding of their domain of study as [56]. The obtained ontologies simply serve as a community knowledge reference and thereby remove heterogeneity in the community.

In the last couple of years the application area of the *Semantic Web* [11] brought more attention to DLs and their powerful reasoning systems. The semantic web is a future version of today's World Wide Web, where web content or services will be annotated with formal representation of their meaning. The annotation can state in which context a keyword is used. Based on a formal description of the keywords in an ontology, better search results or matching services can be obtained using reasoning methods. Despite a future vision, the semantic web is already a strong motivation for the development of powerful reasoning algorithms for very expressive DLs on the one hand and for the implementation of DL systems and DL tools on the other, see [21,20].

An important step towards realizing the semantic web was the standardization of the web ontology language OWL [10,24] and its DL-based dialect OWL DL, which we will discuss in more detail in Section 2.1.3 and which was used in our application.

In the remainder of this section we give a brief introduction to the main ingredients of a Description Logic system. We introduce some of the reasoning services employed in practice and emphasize those used in our application of context-aware systems.

2.1. Description Logics and their reasoning services

Typically, a Description Logic system consists of four parts:

1. *Description Language* formalism for capturing the notions from the domain.
2. *TBox*: a collection of concepts, which capture the main categories of the domain of interest,
3. *ABox*: a collection of facts about concrete instances in the application domain, and
4. the reasoning component.

The elements in TBox and ABox are formulated in the description language. Typically the TBox is also referred to as *ontology*. However, in the OWL lingo, where individuals are allowed in the TBox in the form of nominals, the term OWL ontology can refer to TBox *and* ABox collectively. We use both terms interchangeably throughout the paper. The TBox and ABox together are often referred to as the *knowledge base*. The concept descriptions in the knowledge base are given in the actual description logic. Next, we take a brief look at the syntax and semantics of the description logic $\mathcal{ALC}$. For a thorough introduction to Description Logics we refer the reader to [6].

2.1.1. DL knowledge bases

The main ingredients for representing terminological knowledge are *concept descriptions*. For example, such a concept description can characterize the category of 'mother' as a female person who has a person as a child, in the following way:

$$\mathtt{Person} \sqcap \mathtt{Female} \sqcap \exists \mathtt{has\text{-}child}.\mathtt{Person}.$$

In this expression `Person` and `Female` are *concepts* and `has-child` is a so-called *role* — a binary relation.

Starting from a set of primitive names, complex concept descriptions can be composed by using *concept constructors*. In Table 1 we see the concept constructors provided by the DL $\mathcal{ALC}$. $\mathcal{ALC}$ is the minimal propositionally closed DL. $\mathcal{ALC}$ provides negation, conjunction and disjunction of concepts. Furthermore, it provides existential restrictions and value restrictions. Intuitively, existential restrictions state that for every individual i_1 which belongs to the concept $\exists r.C$, there is an individual i_2 that is of concept C and i_1 and i_2 are related via the role r. Value restrictions state that for every individual i_1 which belongs to the concept $\forall r.C$, *all* individuals that are related to i_1 via the role r belong to concept C.

The semantics of concept descriptions are given in a set-theoretic way. The semantics is defined in terms of an *interpretation* $\mathcal{I} = (\Delta, \cdot^{\mathcal{I}})$. The domain Δ of $\mathcal{I}$ is a non-empty set of individuals and the interpretation function $\cdot^{\mathcal{I}}$ maps each concept name to a set $A^{\mathcal{I}} \subseteq \Delta$. Each role name r is mapped to a binary relation $r^{\mathcal{I}} \subseteq \Delta \times \Delta$.

Starting from an interpretation of concept and role names, the extension of $\cdot^{\mathcal{I}}$ to arbitrary concept descriptions is defined inductively, as shown in the third column of Table 1.

The TBox allows to introduce names for concept descriptions. For instance we can store

$$\texttt{Mother} \equiv \texttt{Person} \sqcap \texttt{Female} \sqcap \exists\texttt{has-child}.\texttt{Person}$$

as a concept definition in the TBox. Besides definitions there are other forms of TBox statements.

Definition 1 (TBox axioms) *Let A be a concept name and C and D be concept descriptions, then a*

primitive concept definition *is an expression of the form $A \sqsubseteq C$.*
concept definition *is an expression of the form $A \equiv C$.*
general concept inclusion (GCI) *is an expression of the form $C \sqsubseteq D$.*
concept equivalence *is an expression of the form $C \equiv D$.*

All of the above statements are called TBox axioms. *The semantics of TBox axioms are given by the interpretation function: $A^{\mathcal{I}} \subseteq C^{\mathcal{I}}$, $A^{\mathcal{I}} = C^{\mathcal{I}}$, $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ or $C^{\mathcal{I}} = D^{\mathcal{I}}$ respectively.*

Primitive concept definitions and GCIs give only necessary conditions while concept definitions and concept equivalence axioms state necessary *and* sufficient conditions for the concept. Obviously, GCIs are the most general type of concept axioms.

If in a concept definition $A \equiv D$ the concept description D refers directly or indirectly to the concept name A, we call such a concept A a *cyclic concept*. Based on this, we define different notions of a TBox.

Definition 2 (TBox) *Let A, B be concept names and let C, D be concept descriptions. A finite set of TBox axioms $\mathcal{T}$ is called a* TBox.

unfoldable TBoxes *only contain (primitive) concept definitions, where each name appears at most once on the left-hand side of a (primitive) definition and the TBox must be acyclic, i.e. without cyclic concepts.*
cyclic TBoxes *may contain cyclic (primitive) concept definitions.*
general TBoxes *may contain GCIs and concept equivalence.*

An interpretation is a model *of a TBox, if for all $A \sqsubseteq C \in \mathcal{T}$, $A \equiv C \in \mathcal{T}$, $C \sqsubseteq D \in \mathcal{T}$ and $C \equiv D \in \mathcal{T}$ it holds that $A^{\mathcal{I}} \subseteq C^{\mathcal{I}}$, $A^{\mathcal{I}} \equiv C^{\mathcal{I}}$, $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ and $C^{\mathcal{I}} = D^{\mathcal{I}}$. An interpretation $\mathcal{I}$ satisfies a TBox $\mathcal{T}$ iff $\mathcal{I}$ satisfies every axiom in $\mathcal{T}$. In this case $\mathcal{I}$ is a* model of $\mathcal{T}$.

OWL supports all of the above mentioned TBox axioms for modeling. The kind of TBox axioms a TBox contains has great effect on the computational complexity of reasoning methods for TBoxes – unfoldable TBoxes are often easier to handle than general ones.

Besides concept constructors many DLs provide means to declare properties of roles in the TBox. For instance, roles can be declared to be

- a *transitive role*, which is interpreted as a transitive relation.
- the *inverse role* of another role *inverse(R_1, R_2)*, which are interpreted as $R_2^{\mathcal{I}} = \{(a,b) \mid (b,a) \in R_1^{\mathcal{I}}\}$.
- a *super-role* of another one. Role inclusion axioms $R \sqsubseteq S$ enforce that every pair $(a,b) \in R^{\mathcal{I}}$ is also $(a,b) \in S^{\mathcal{I}}$. The set of these kind of statements forms the *role hierarchy*.

All of these role declarations are supported in OWL DL to build ontologies.

The knowledge about individual entities from the application domain can be expressed by so-called *ABox assertions*. For instance, we can state that the in-

Table 1
DL syntax and semantics of $\mathcal{ALC}$-concept descriptions and the corresponding OWL syntax.

constructor name	DL syntax	semantics	OWL syntax
negation	$\neg C$	$\Delta \setminus C^{\mathcal{I}}$	complementOf
conjunction	$C \sqcap D$	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$	intersectionOf
disjunction	$C \sqcup D$	$C^{\mathcal{I}} \cup D^{\mathcal{I}}$	unionOf
existential restriction	$\exists r.C$	$\{x \in \Delta \mid \exists y : (x,y) \in r^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\}$	someValuesFrom
value restriction	$\forall r.C$	$\{x \in \Delta \mid \forall y : (x,y) \in r^{\mathcal{I}} \rightarrow y \in C^{\mathcal{I}}\}$	allValuesFrom

dividual Alice is a software programmer and she has the colleague Fred in the following way:

`SoftwareProgrammer`(Alice),
`hasColleague`(Alice, Fred)

There are two kinds of ABox assertions used for DL systems—one kind expresses that an individual belongs to a concept and the other one specifies that two individuals are related via a role.

Definition 3 (ABox, ABox assertion) *Let C be an arbitrary concept description, r a role name and i, j two individual names be two individual names. Then a*

concept assertion *is a statement of the form $C(i)$.*
role assertion *is a statement of the form $r(i, j)$.*

An ABox $\mathcal{A}$ is a set of concept assertions and role assertions.

In order to capture ABoxes the interpretation function is extended to individual names, which are mapped to elements of the domain Δ.

Definition 4 (Semantics of ABox) *Let C be an arbitrary concept, r be a role name and i, j be two individuals. Then an interpretation $\mathcal{I}$ satisfies*

- *the concept assertion $C(i)$ iff $i^{\mathcal{I}} \in C^{\mathcal{I}}$ and*
- *the role assertion $r(i, j)$ iff $(i^{\mathcal{I}}, j^{\mathcal{I}}) \in r^{\mathcal{I}}$.*

An interpretation $\mathcal{I}$ satisfies an ABox $\mathcal{A}$ iff $\mathcal{I}$ satisfies every assertion in $\mathcal{A}$. In this case $\mathcal{I}$ is a model of $\mathcal{A}$.

Equipped with the formalisms for TBoxes and ABoxes and their meaning, we can turn to the reasoning services available once we have represented our information in this way.

2.1.2. DL reasoning services

Often the statements in the knowledge base capture other facts implicitly. To detect these facts and to make them explicit is the idea behind DL reasoning services. Furthermore the reasoning algorithms that infer new facts must fulfill certain requirements to ensure that applications using these services are reliable. To begin with, reasoning algorithms should be sound and complete, i.e. every answer returned by the service must be correct and we get all the answers. Furthermore the reasoning algorithm must be terminating so that an answer is always obtained. Based on the formal semantics of DLs, reasoning services can be defined and, more interestingly for our application, the requirements for reasoning algorithms that provide these services can be proven. These requirements hold for the inference algorithms developed for DLs, and they are implemented in today's typical DL systems. In the following we introduce some of the reasoning services that are readily available in DL systems and that we have used in our application of context-aware systems.

When adding a concept definition to a TBox, it is crucial to know whether the specified concept description contains a contradiction (w.r.t. the knowledge base) or whether it could be fulfilled by any individual and thus models something possibly meaningful. This leads to the formal notion of consistency.

Definition 5 (Consistency) *Let C be a concept description and $\mathcal{T}$ be a TBox. The concept description C is* consistent *iff it has a model, i.e., there exists an interpretation $\mathcal{I}$ where $C^{\mathcal{I}} \neq \emptyset$. In this case $\mathcal{I}$ is a model of C. A TBox $\mathcal{T}$ is* consistent *iff every concept in $\mathcal{T}$ is consistent.*

If a concept or TBox is not consistent, it is called *inconsistent*. Even for very expressive DLs inconsistencies can be detected automatically by the DL reasoner.

Another terminological inference task is to determine whether one concept description is more general than another, i.e., whether one concept description C is implied by another concept description D. This is the case, if every individual that is an instance of C also is an instance of D. The following definition formalizes this notion of subsumption.

Definition 6 (Subsumption) *Let C, D be concept descriptions and $\mathcal{T}$ a TBox. The concept description C*

is subsumed w.r.t. $\mathcal{T}$ *by the concept description* D *(*$C \sqsubseteq_{\mathcal{T}} D$*), iff* $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ *holds in every model* $\mathcal{I}$ *of* $\mathcal{T}$.

By *TBox classification* we denote the computation of all the subsumption relationships that hold for the named concepts in a TBox. By testing for subsumption relationships or by classifying the whole TBox the modeler can determine whether relations between notions from the domain are faithfully captured in the TBox. For instance, if two concepts, which stand for different notions in the application domain, turn out to be equivalent in the TBox, more information needs to be provided to distinguish them by means of their descriptions. Thus the subsumption tester in DL systems can help to spot these modeling deficiencies. This kind of test was helpful when we built our KB for the application scenario that we will describe in Section 3.1.

Beyond the concepts in the TBox there are also reasoning services available for the individuals in the ABox.

Definition 7 (Instance of, ABox realization) *Let* C *be an arbitrary concept description,* i *an individual name and (*$\mathcal{T}$*,*$\mathcal{A}$*) a DL knowledge base. The individual* i *is an* instance of C *w.r.t. (*$\mathcal{T}$*,*$\mathcal{A}$*), iff* $a^{\mathcal{I}} \in C^{\mathcal{I}}$ *for every model* $\mathcal{I}$ *of (*$\mathcal{T}$*,*$\mathcal{A}$*).*

ABox realization *of* i *w.r.t. (*$\mathcal{T}$*,*$\mathcal{A}$*) returns all concepts* C *defined in* $\mathcal{T}$ *for which* $i^{\mathcal{I}} \in C^{\mathcal{I}}$ *for every model* $\mathcal{I}$ *of (*$\mathcal{T}$*,*$\mathcal{A}$*).*

So, instance checking denotes the task of testing whether a given individual is an instance of a given concept. Realization of an individual, in turn denotes the retrieval of all (most specific) named concepts from the knowledge base that a given individual is an instance of. ABox retrieval realizes this task for all individuals described in the ABox. This reasoning service is the central one to realize the task of recognition of situation types in our approach.

Another ABox reasoning service that we only mention briefly here and that is a good alternative to realize situation type recognition are *conjunctive ABox queries*. Here one can pose more sophisticated queries to the knowledge base. The queries itself are complex expressions – typically conjunctions – containing variables. These variables are instantiated by the DL reasoner with ABox individuals to answer the query. Conjunctive ABox queries are a very powerful way to query the DL knowledge base. However, in the remainder of the paper we resort to ABox realization, since this service is provided by most of the current DL reasoner systems and thus is a better starting point for evaluation.

2.1.3. The web ontology language OWL

In 2004 the W3C made the web ontology language OWL a recommendation for the semantic web. The OWL standard incorporates ideas from the earlier ontology language DAML+OIL and from RDF Schema.

The W3C recommendation for OWL specifies three dialects. While the most expressive dialect *OWL full* is beyond the expressivity of DLs and reasoning in it is undecidable, the other two dialects correspond to DLs for which sound and complete reasoning procedures exist. *OWL DL* can express ontologies written in the DL $\mathcal{SHOIN}$[1]. The less expressive *OWL lite* can express ontologies written in the DL $\mathcal{SHIF}$[2].

In OWL lingo concepts are called classes and roles are referred to as object properties. In Table 1 on page 7 in the third column some of the concept constructors available in OWL DL are displayed in correspondence to the DL ones. The semantics of the OWL DL constructors is the same as for DLs. Thus all reasoning services introduced in the last section are applicable for OWL DL as well.

The standardization of OWL has brought DLs and their reasoning systems to the attention of people from many different application areas and the ontology language is used in many novel application areas – context-aware systems are some of them. Next, we turn to the usage of DLs (or OWL DL resp.) in this application domain.

2.2. DLs for context-aware systems

The main idea how to use DL systems for context-aware systems is to build a TBox with the main notions from the domain and the description of types of situations relevant to the application. This kind of TBox needs to be “hand-crafted” by a human modeler at design time. This task can be supported by the automated consistency tests and classification that DL systems provide. At run-time of the context-aware system a description of the current situation is generated by the context application and written into the ABox in form of ABox assertions. Based on this description, the DL system can determine automatically into which situation-type this situation falls by computing ABox realization for the situation ABox.

[1]$\mathcal{SHOIN}$ is the DL $\mathcal{ALC}$ augmented with number restrictions, nominals, functional roles, transitive and inverse roles and role hierarchies.

[2]$\mathcal{SHIF}$ is the DL $\mathcal{ALC}$ augmented with functional roles, transitive and inverse roles, and role hierarchies.

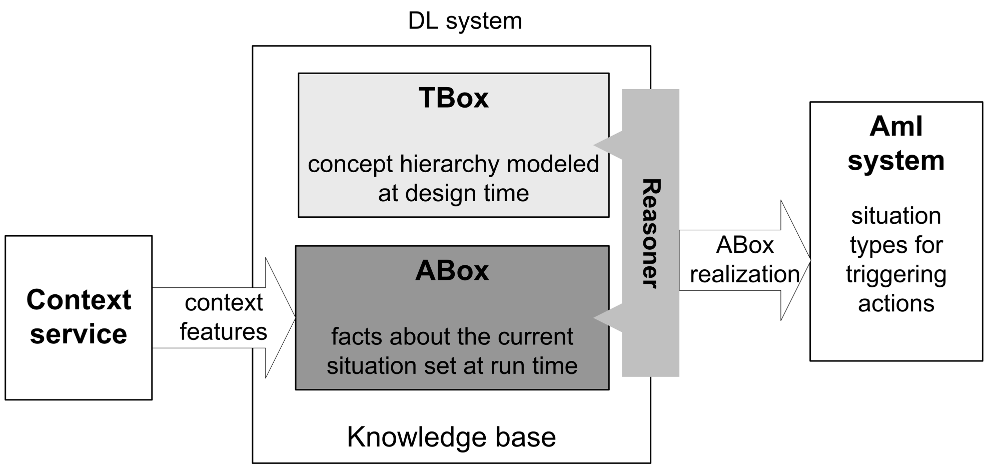
DL system
TBox
concept hierarchy modeled at design time
ABox
facts about the current situation set at run time
Knowledge base
Reasoner
Context service
context features
ABox realization
AmI system
situation types for triggering actions

Fig. 2. Using DL systems for recognizing situation types.

Description Logics have several characteristics that make them a well-suited choice for context-aware systems. As knowledge representation formalisms DLs are designed to model hierarchies of notions from the respective domain. Thus DLs naturally support the requirement of context-aware systems to be able to model information on different levels of detail or abstraction as a means to deal with heterogeneous information sources. Moreover the formalism of DLs allows to combine information from different sources and combine them to aggregates in form of complex descriptions capturing different aspects of an informational entity.

However, the most important feature of DLs that makes them suitable for context-aware systems is their *open-world semantics*. Intuitively, this kind of semantics assume that the KB does not have complete information about the world, but that some information might be missing. So, from the absence of a fact in the KB, say `SoftwareProgrammer`(Alice) it cannot be inferred that the negation ¬`SoftwareProgrammer` (Alice) holds. In systems that adopt *closed world semantics*, such as databases, the absence of a fact means that the contrary holds, since these systems assume complete knowledge about the world. For context-aware systems open world semantics is clearly the better option, since context information is often incomplete. Thus DL systems offer a way to handle this kind of characteristics of context information in a graceful way.

As a collection of reasoning services DLs have even more to offer for context-aware systems. DL reasoning services support the building of the TBox by detecting inconsistencies or missing subsumption relations early. The reasoning task of classification gives the modeler an overview of how aggregated concepts relate to each other. Furthermore, ABox reasoning services can be used to infer the situation type of a certain situation and thus solve the problem of the actual context recognition. Since the methods for these reasoning tasks are implemented in highly optimized reasoner systems, these services are available to the ambient intelligence community directly.

To the best of our knowledge this approach of using DL systems for context-aware systems was first described by us in [52], but was also pursued by others in [54] and in [1]. The next section introduces the whole approach in detail.

3. Approach for ontology-based recognition of situation types

Our approach for situation-aware systems is based on DL systems. As described in the Section 2.1 a DL system consists of a knowledge base defined using the Description Logics based language OWL DL and a reasoning component providing a set of reasoning services. With OWL DL a *knowledge base* consists of a TBox and an ABox which together represent the knowledge of the AmI system about the current situation. The TBox contains *concepts* organized in a hierarchy which model the aspects of the situations relevant to the considered AmI system. The ABox contains a collection of *facts* describing the current situation of the AmI system.

At design time the TBox is created by the system developer. All conceptual knowledge about situations which should be recognized by the system have to be expressed by OWL DL elements. Part of the presented approach is a modeling methodology to systematically decompose complex real world situations. The main idea of the methodology is to identify these aspects of situations which are relevant for decision making. Although the analysis of the application scenario and the identification of situation types remains an intuitive task, the modeler is assisted by the DL system when building the TBox. First, inconsistent descriptions can be detected automatically and second, the concept hierarchy can be computed automatically so that the modeler can discover unintended sub/super category-relations between concepts directly.

At run-time context information is added to the ABox as the information about the current situation. The reasoning service *ABox realization* is used as a means to recognize the type of situation from the situation description in the knowledge base. We propose a framework consisting of a context service, a DL system and an AmI system. The context management, especially the management and access of heterogeneous, alternative and distributed sources is covered by the context service. The AmI system specifies the relevant context information in a context profile. The concept definitions in the TBox are used as a common vocabulary specifying context to be exchanged between all system components. The DL system manages the knowledge base and provides the DL reasoning tasks. A summarized view on the proposed approach is depicted in Figure 2.

3.1. Intelligent door lock scenario

To illustrate our approach we use a scenario taken from the smart home domain. An automatic door lock should pick the next action to be taken depending on the person ringing at the door. We assume that the door system is equipped with a video camera and a microphone and provides information about the ringing person. Based on this information the door lock has to determine one of the following actions:

1. Open the door, if the person is authorized.
2. Ask a resident in case the person is unknown.
3. Do not respond at all or let the ringing person leave a message if no resident is available (similar to: nobody at home).

In a first step the person ringing is identified (e.g. as a resident of the house or a neighbour) or classified as a member of a group of persons (e.g. a fire fighter or a

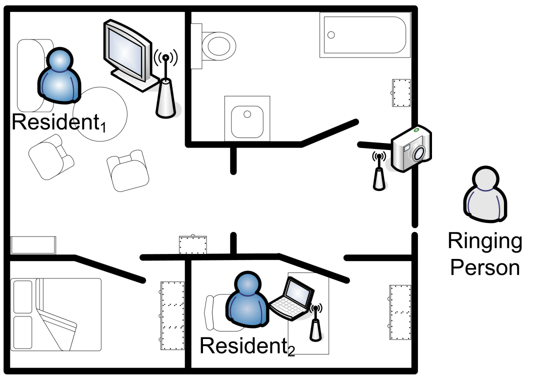

Fig. 3. Intelligent door-lock scenario.

postman). Per definition the door opens for authorized persons only (e.g. a resident or family member). If the person at the door cannot be unambiguously classified, the decision whether to open is forwarded to a resident. The door tries to contact a resident taking her current situation into account (e.g. her activity and currently used devices). If a resident is watching TV, the image captured by the camera can be redirected to the TV set being used. If no resident is at home, the system tries to contact a resident via a cellular phone or another mobile device currently in use. If no resident can be reached within a short while, the system informs the person at the door, offers to leave a message and keeps the door closed.

Example: For the holiday season a neighbour is asked to water the flowers while the residents are on vacation. The door lock system identifies the person ringing as the neighbour. Furthermore, the door system checks whether the ringing neighbour is authorized by a resident to enter the house. If in addition all residents are on vacation, the neighbour can be recognized as an authorized person and the door opens. We decided to use this fairly simple scenario for our first case study to ensure that it is easy to model. Nevertheless, as the scenario description already shows, if modeled in detail, it becomes sufficiently complex to illustrate pitfalls of context modeling and the use of reasoning services.

3.2. Ontology-based situation modeling

Recalling the conceptual architecture in Figure 1, the DL system bridges the context-awareness and decision making phases. Thus, the goal of the modeling of situations is to recognize the current situation in a way that a decision can be made about the actions to be triggered based on the recognized situation. For instance, if our example AmI system is in the situation, that an authorized person is ringing the doorbell, it should be able to derive the action to open the door immediately.

3.2.1. Decomposing the scenario

It is the task of the AmI system developer to analyze the application scenario, to identify relevant situations and to derive the conceptual knowledge about these situations. To handle this complex exercise the notion of situation decomposition is introduced as a systematic approach for creating ontology-based situation models.

The main idea is to use a task-based approach to identify major situations relevant to the triggering of actions in the AmI system so that the system can fulfill its foreseen tasks. Therefore, for all tasks the system should perform, situation types have to be identified, which allow an unambiguous selection of the actions to be triggered by the AmI systems to fulfill the appropriate task. The identified situation types can then be decomposed into a hierarchy of sub-situations. This method is derived from our observations, that these sub-situations can be modeled by independent aspects of a situation which can later be composed to model the identified situation types. For example, in the door-lock scenario the ringing person and the situation of the residents of the house are independent aspects of the overall situation.

For a systematic decomposition, the applications scenario can be analyzed according to so called *"aspects of interest"*. These aspects should be atomic in the ideal case, but at least fine-grained enough to be modeled by a small set of concepts. This usually results in a step-by-step decomposition with an increasing granularity in each step. For instance, in the door-lock scenario, the identity or role of the ringing person, and the presence of the residents are aspects of interest. Each of these aspects can be decomposed further. For instance, for the presence of the residents the aspects of reachability and the willingness to communicate are of interest, which again depend on the activity, location and the devices nearby. The following criteria can be used for scenario decomposition:

Spatial decomposition: In almost every scenario it will be possible to adopt a spatial decomposition. In our scenario, interesting locations are the door, where the ringing person is situated and the rooms of the house were the residents stay. Moreover, also outside locations might be of interest in the case that no resident is at home and should be contacted remotely due to urgency.

Temporal decomposition: Temporal aspects should also be taken into account when decomposing the scenario. Usually, different points in time contribute independently or in relation to the overall situation. In our example the resident might have left the house 10 minutes ago when a postman is ringing. Knowing that the resident went just out to buy a newspaper which usually takes him 15 minutes, the system could inform the postman that it expects the resident to be back in 5 minutes.

Acting persons: In many scenarios several persons play different roles. The roles they play can also be modeled as independent aspects of the overall scenario. In the door-lock example we distinguish between residents, relatives, neighbours and different types of professions like postman, fire fighter or police man.

In addition to these criteria further scenario-specific aspects can be identified. In our scenario the technical reachability, activity and presence of persons play an important role.

The goal of the decomposition is the identification of basic situations which can be unambiguously identified based on a small set of context features. Moreover, basic situations should also be processable independent from other basic situations. Therefore, in parallel to the decomposition of situations, the context features relevant for describing particular situations have to be identified. In the door-lock scenario for instance the location and the capabilities of the devices used by a resident are relevant context features to determine the reachability of the resident.

3.2.2. Modeling situations in the TBox

After this aspect-wise decomposition, the identified sub-situations have to be modeled in the TBox. Moreover, tests should be performed to validate the consistency and correctness of the created model. In addition, performance aspects play an important role, so performance issues at run-time might lead to changes of the TBox as well. According to our experiences gained in the process of modeling several scenarios, we encountered four main activities of knowledge base development:

1. Building of the TBox,
2. Building of the test ABox,
3. Testing the inferences for the ontology, and optionally
4. Performance tuning of the ontology w.r.t. the needed inferences

These activities, however, should be performed interlaced and should not be read as a strict sequence.

Building the TBox The set up of the TBox can be performed in a task-oriented and incremental way. That means, in the beginning one task of the AmI system can be selected and all identified sub-situations contributing to the decision for performing that task have to be modeled. The modeling should start with the most fine-grained sub-situations which are independent of each other, as stated above. More situations can be added step-by-step as soon as one task is completely covered. The sub-situations can also be modeled step-by-step and later on used to compose more complex situation.

In our modeling approach the resulting TBox is twofold. One part of the ontology contains situation descriptions and the second part consists of general concepts required for the definition of situations. Generic concepts of the door-lock example are for instance location, device and person. A situation is usually described based on several generic concepts. Situations are introduced into the TBox as named concepts which are ordered in the expected subsumption order manually. The situation concepts might be provided with a definition later on. As mentioned in the introduction, generic ontologies could be used as upper ontologies which are refined according to the scenario requirements. The basic relations from the scenario domain should be captured as roles in the TBox.

Once the basic situations are introduced as named concepts, the definition for these concepts should be provided (or successively refined). At the early stages these definitions should be "close to the intuition" of the notion to be modeled and all language constructs that express this intuition best should be used. Generally, it is a good strategy to model the TBox with as little redundancy as possible. This eases debugging of the ontology, i.e., tracking the cause of inconsistencies. Furthermore, keeping redundancy low is a good design principle when developing an ontology in a team. Furthermore, it is advisable to perform consistency checks often when extending the TBox. Sources of inconsistencies can be tracked more easily, if only a few definitions have been changed since the last consistency check.

For our application it is desirable to have a fine-grained concept hierarchy for the hierarchy of situation concepts and for collection of concepts from the sub-domain that will be central to distinguishing different context concepts (such as the collection of different resident concepts in the Doors scenario). So, the concept hierarchy should be computed and checked against the intended hierarchy. In case intended subsumption relations are missing the concept definitions must be revised and in case concepts collapse (i.e., are equivalent although intended as different concepts), the concepts must be refined. Often this involves the modeling of a new aspect, as devices or persons in our context applications. Each subdomain models a different aspect of the context concepts collection in a separate hierarchy.

At run-time of the application another benefit of the fine-grained hierarchy is to infer relatively specific context concepts via realization even in cases where information about the situation of the application is incomplete. For example, when the exact location of the resident is unknown, the situation that the Resident is out of home can still be inferred and appropriate actions can be taken. In contrast, if there is no fine-grained concept hierarchy, we can probably only infer a generic situation type, if this information does not suffice to derive that the resident is currently traveling, for example.

Building the test ABox In activity 2, we build an ABox for testing

1. whether the vocabulary in the TBox is already elaborated enough to be used for a detailed situation description and
2. whether concept definitions in the TBox are already precise enough to give the expected reasoning results for a situation description in the ABox.

The situation descriptions in this ABox should be similar to what is to be expected to occur in the application in terms of aspects that are supplied at run-time by the context service. If it turns out that the concepts defined in the TBox do not allow to describe a situation detailed enough, or if necessary "ingredients" for such a description are simply missing, the TBox must be extended appropriately. Thus, the activities 1 and 2 should be carried out in parallel.

Testing Activity 3 starts as soon as the TBox is filled with a few concept definitions. At the early stages it should be tested whether the TBox is consistent, i.e., whether it contains contradictions. These consistency tests should accompany the whole process of building the TBox. As soon as the concepts in the TBox are sufficiently elaborated to represent (sub-)situations from the application it is interesting to determine whether the achieved level of detail is enough to infer the desired information. To this end classification tests and realization tests are performed. On the one hand it must be checked if the classification results meet the intuition of the modeler, for instance, if unintended subsumption relations are detected. On the other hand ABox realization has to be performed to check if the most specific concepts are detected for the situation. If the results do not meet the expectation of the modeler, the concept definitions should be refined.

Performance tuning Activity 4 is performance tuning. Now the knowledge base is analyzed w.r.t. the syntactic constructs in use and run-times for the inferences (needed at run-time of the application). The analysis of the syntactic constructs should yield what syntactic constructs are used and also how often they occur. If, for example, just one transitive role is used in the knowledge base, it should be carefully checked if the transitivity of this role is essential for intended result of the inferences. If syntactic constructs that are notorious for making reasoning harder and degrading the performance of DL reasoner, can be omitted or replaced by others without losing (important) inferences, these constructs should be deleted or replaced. Please refer also to Section 5.1.

Another way of enhancing performance of the reasoner is to add information to the TBox. One can add subsumption information so-called told subsumers to the (primitive) definition of concepts. For example, the concept C1 is a sub-concept of C2, but is not defined in terms of C2, then the reasoner has to "discover" this subsumption relation, which can be costly in terms of run-time. If this subsumption information is added to the definition of C1, the reasoner only needs to test for consistency. Similarly, one can also add "told non-subsumption" information, e.g. by adding more disjointness constraints, to avoid the effort of discovering non-subsumption by costly methods.

3.3. Filling the ABox

If the TBox is created the AmI system is ready to run. During the startup of the AmI system the TBox has to be loaded into the DL system and classified once before the first ABox realization service usage can be performed. As stated above context information is used as the information describing the current situation. It has to be added to the ABox as individuals to be processable in the DL system. To cover a subset of the characteristics of context named in Section 1.1 of the introduction, we assume the availability of a context service. This context service should be able to integrate and manage heterogeneous and geographically dispersed context sources. By accessing this context, context features representing facts about the current situation are written into the ABox.

To identify the context feature which have to be retrieved we apply the creation of a so called context profile. This context profile contains all concepts from the TBox for which the context service can provide information. This context profile have to be created manually or may be derived automatically. The latter can be achieved if the information about the managed information provided by the context service can be mapped to the concepts in the TBox.

The context profile is then used for either request all information at once in case that the AmI system is requested to perform an action. In our example this is the case if someone is ringing at the door. This is similar to taking a snapshot of all relevant information about the current situation at a certain point in time. If the AmI system should act proactively, it can subscribe for all situations in the profile to the context service. The system is then notified by the context service about the changes of any of the information in the context profile. If a change notification is received by the AmI system it can update the ABox and perform reasoning in order to check whether the situation has changed in a way that an action has to be performed.

3.4. Perform reasoning

As already described in the previous section reasoning can be performed on demand or event-based. In both cases all relevant and available context information have to be added as individuals to the ABox. To compute the types of the current situation the situation is itself modeled as an individual of the general situation concept. The context information is related to that situation based on roles which set different individuals in relation. To compute the types of a situation, ABox realization is performed on the situation individual. As a result the DL reasoner responds with a list of concept names. In particular, this list contains the most specific concepts for that situation individual.

Dependent on the available context information, the computed situation types, i.e., the concept names, are more or less specific. If not enough information about the current situation is available, the reasoner responds with a generic situation concept. For instance, if no information about the resident of the house are available, just a concept describing that the doorbell rings can be computed. If a large set of context information is available, a very specific situation type can be computed. As an example, if it is known that the location of the resident is the "living room", the TV is "on" and the ringing person is the postman, concepts describing that a postman is ringing and the resident should be informed by presenting a video on the TV can be computed. Thus, dependent on the amount of context information available, the computed situation types are more or less specific. A result is available in any case, even if the provided context information is incomplete.

Based on the computed situation types it is the task of the AmI system to determine the right action. This is a separate step in our approach to decouple the identification of the current situation and the decision process in order to enable a high flexibility within applications for context dependent decision making.

4. A case study of situation-awareness based on DL systems

We use the scenario introduced in Section 3.1 to implement a case study of validating the feasibility of our approach. Especially, a framework was developed which serves as the foundation for implementing various application scenarios. In the following we introduce a framework for DL-based situation-awareness and the situation model created for our scenario.

4.1. Framework architecture

The main components of our framework are the Context Service, responsible for providing the required context information, the AmI system which is interested in the current situation and the DL System, responsible for processing ontologies and performing ABox realization for situation recognition (see Figure 4.1). A further component is the Context Profile used by AmI systems for requesting context information from the context service as described in Section 3.3.

4.1.1. Context Service

In our implementation we use the distributed context service described in [45]. The context service is available to each application in the form of a local component which provides an access interface for contexts. The context service is able to manage a large set of highly distributed context sources and handles the access, transformation and distribution of applications to context information in a transparent manner. Where and how the information is gathered is not visible to the context application, it just uses a well-defined API for requesting context information based on context profiles.

The context management inside the context service is based on meta-model derived from topic maps. It contains entities which may have a set of attributes. Relations between entities are modeled as associations. The semantics of these elements is defined based on an ontology specified in OWL DL. Thus, we defined a mapping between the context domain model supported by the context service as described in [45] and our scenario ontology. Based on an extension of the API for accessing context information we support now a profile-based access of context information. Each time the Door-lock system should react on the ringing of the door bell, it sends the context profile with a request to fill it to the context service. The context service responds with the requested information integrated into the profile. The profile might be completely or partially filled according to the current availability of context sources.

4.1.2. DL system

The DL system is the second component of the framework. It is able to maintain multiple ontologies and provides the service to perform reasoning tasks on these ontologies. An AmI application can register a knowledge base (i.e., an OWL DL based ontology) at the DL system, which processes the TBox of the ontology and builds up an internal representation of the processed TBox. Each registered ontology is assigned with a unique identifier for later referral. During the run-time of context applications we assume a constant TBox allowing the reuse of the internal representation of the TBox built up at registration time. At run-time the AmI application requests reasoning tasks based on varying ABoxes.

Thus, the DL system is a system service which can be used by several context applications in parallel. It can be placed on a high-performance system server residing within the infrastructure. Especially, this enables the remote processing of reasoning tasks for applications running on mobile or resource limited devices. The DL system is represented by off-the-shelf DL reasoner systems, e.g., RACERPRO [22], the FACT++ system [51] or PELLET [42]. They all implement the DIG interface [9] and thus, can be exchanged on that basis.

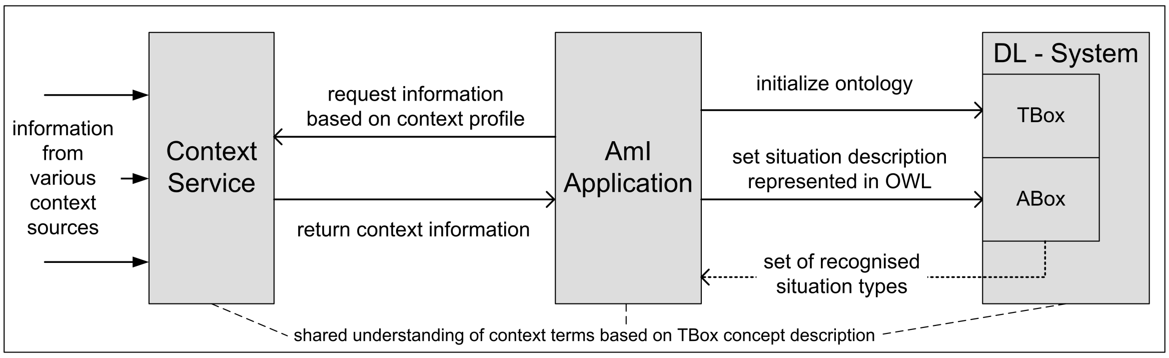
information
from
various
context
sources
Context Service
request information based on context profile
return context information
AmI Application
initialize ontology
set situation description represented in OWL
set of recognised situation types
DL - System
TBox
ABox
shared understanding of context terms based on TBox concept description

Fig. 4. Architecture of the framework for situation-awareness.

4.1.3. AmI Application

The AmI application is the component which coordinates the context access and reasoning. It registers an application specific ontology at the DL system and requests reasoning tasks from this DL system. The context is gathered from the context service component which can be local to the application component or remote on an infrastructure server. The context application uses a context profile to request the context values required for classifying the current situation or a certain aspect of a situation. The context service responds with a filled context profile containing at least a subset of the context values requested by the application.

The context values are used to create individuals of an ABox according to the defined context ontology. These individuals belong to a certain situation individual. The ABox is then sent to the DL system which performs realization on the situation individual to classify the situation. Based on the classification of the situation, the context application can determine what action has to be performed. Thus, the determination of the action to perform is separated from the situation detection.

The implementation of the AmI application is based on a set of up-to-date technologies. We used Java as implementation language for the framework according to the Java 5.0 specification. For the integration of ontologies and deterministic reasoning schemes we used the Semantic Web Framework Jena in Version 2.5.1. In addition to using the DIG interface we also integrated the DL reasoner Pellet in Version 1.5. via its provided Java interface. To publish our upper ontology we used Apache 2.2 as the web server. In ongoing work we also embedded this framework in a more general framework for situation-awareness. The extended framework is described in [46] and covers the integration of heterogeneous sensors (e.g., wireless sensor networks, microphones and other stand-alone sensing devices), the integration of classifiers for these sensors and the use of various reasoners for situation detection.

4.2. Implementing the intelligent door-lock scenario

We describe now how to model the ontology for the door-lock scenario according to the methodology introduced in Section 3. Starting with the decomposition of the scenario and the identification of relevant context information, the situation types for the door-lock scenario were identified and modeled in the TBox.

4.2.1. Decomposing the scenario

The scenario can be decomposed based on the criteria described in Section 3.2. Starting with the tasks of the system – opening the door, asking a resident or keeping the door closed – aspects of the overall situations can be identified. For all tasks a spatial decomposition is relevant, i.e., aspects are the situations in front of the door and in the house. Moreover, acting persons can be identified and their current situations can be modeled. For opening the door the identity and role of the ringing person is relevant. For instance for a resident or an authorized neighbour the door can be opened immediately, while if the ringing person is not authorized, the task of "asking a resident" has to be performed.

Relevant context. To reason about the situation of the door scenario, the following contextual information is relevant: The *identity and social relations* of and between persons or group of persons can be used to distinguish between residents, their relatives, friends and neighbours (e.g., while some of them may be authorized to enter the house, while others may only enter if another person is already at home) and categories of persons which can be determined by their clothes or

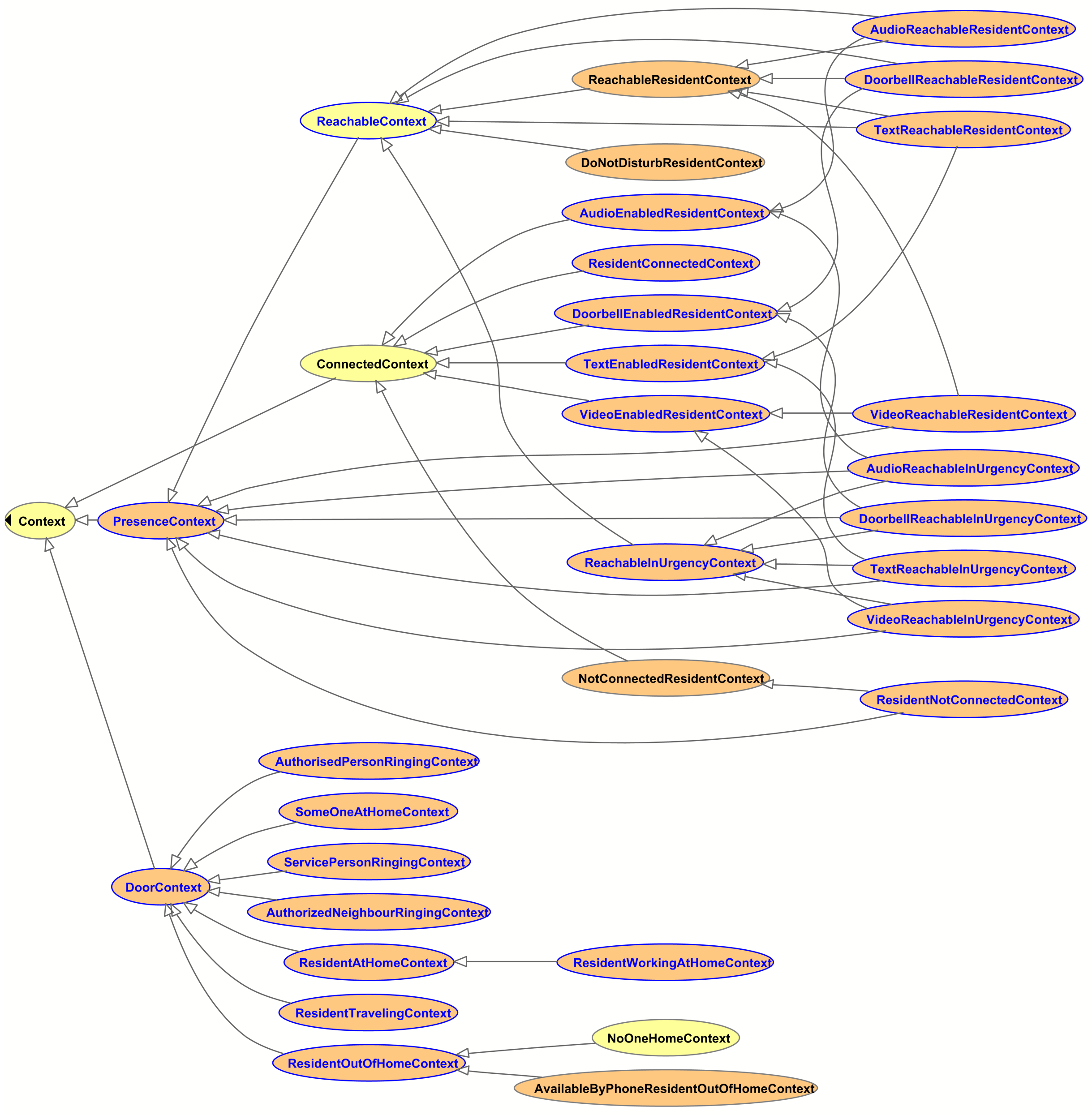
AudioReachableResidentContext
ReachableResidentContext
DoorbellReachableResidentContext
ReachableContext
TextReachableResidentContext
DoNotDisturbResidentContext
AudioEnabledResidentContext
ResidentConnectedContext
DoorbellEnabledResidentContext
ConnectedContext
TextEnabledResidentContext
VideoEnabledResidentContext
VideoReachableResidentContext
AudioReachableInUrgencyContext
Context
PresenceContext
DoorbellReachableInUrgencyContext
ReachableInUrgencyContext
TextReachableInUrgencyContext
VideoReachableInUrgencyContext
NotConnectedResidentContext
ResidentNotConnectedContext
AuthorisedPersonRingingContext
SomeOneAtHomeContext
ServicePersonRingingContext
DoorContext
AuthorizedNeighbourRingingContext
ResidentAtHomeContext
ResidentWorkingAtHomeContext
ResidentTravelingContext
NoOneHomeContext
ResidentOutOfHomeContext
AvailableByPhoneResidentOutOfHomeContext

Fig. 5. Context concepts for the intelligent door scenario.

other characteristic features (e.g., the uniform of a police man or fire fighter, or the pizza boy carrying pizza boxes).

The *location* is relevant for persons who are residents or otherwise able to help the system to decide about the action related to the ringing person, i.e., to authorize ringing persons to enter. Especially of interest is the location of the residents. *Time* as contextual information is relevant in combination with other information, e.g. to determine the activity or current location of a person (e.g. person is working if it is 8 am and located at his office room). Information about the *presence* describes if and how persons/residents would like to be contacted (e.g. a resident is working at his office and do not want to be disturbed by anybody).

The concept `activity` of a person is relevant for determining the presence of a person. Information about the activity is usually not directly available but has to be extracted out of several information, e. g. the schedule, the activity, time, and persons nearby.

Information about the *devices* a person owns and which of them are currently active and in use (e.g. located close to the person) is relevant to determine how to contact a person. In combination with the current connectivity of the devices information can be extracted to describe how to contact a person (e. g. if the device and connection supports audio and video communication or email/text communication only).

4.2.2. Modeling situations in the TBox.

We modeled the doors ontology according to our approach described in Section 3.2 with the task based notion of context in mind. The resulting ontology contains concepts for describing the relevant context and roles for describing the interrelations between these concepts. Furthermore, the ontology contains concepts describing the situation itself in form of a hierarchy of contexts which describe certain aspects of the situation (e.g., who is ringing, the presence, activity and location of the residents and the interrelations between the residents and the ringing person).

Modeling locations. Location is a fundamental concept in the door scenario. We use the four basic concepts `InDoor`, `OutDoor`, `Mobile`, and `ImMobile` to describe the basic features of a location. Each location in the ontology is a sub-concept of either `InDoor` or `OutDoor` as well as either `Mobile` or `ImMobile` using multiple inheritance. The role `nearby` defines that two locations are close together. The properties `hasLightLevel` and `hasNoiseLevel` describe the features of locations relevant in our scenario to derive the activity of a person at a certain location.

`Building` is a sub-concept of `InDoor` and `Im—mobile` and consists of several parts (e.g., Rooms). A sub-concept of `Building` is `House` which enables the description of any house relevant to the situations of the door lock scenario. Another sub-concept `Home` describes the building which contains the "intelligent door lock" system. For `Home` and `House` a close distance can be defined using the `nearby` role to describe the neighbourhood of the home.

Each `Building` can consist of parts. We currently defined only `Rooms` as parts. The other way around, each room is part of a certain building described by the role `isPartOf`. Several sub-concepts of Room are defined to model the rooms relevant to the scenario (e.g. `LivingRoom`, `Kitchen`, `BathRoom`, `BedRoom`, and `OfficeRoom`). Locations of the category `OutDoor` and `Mobile` are also defined within the ontology (e.g. `Car`, `Plane`, `Bus`, and `Train`). These locations are relevant for describing activities of residents.

Modeling of identity and social context. The identity and social context of the persons relevant to the door scenario are modeled based on the `person` concept. Each person has a current location (role `locatedAt`) and lives at a certain location (role `livesAt`). A person which lives at `Home` is defined as a resident $\texttt{Resident} \equiv \texttt{Person} \sqcap \exists\texttt{livesAt}.\texttt{Home}$. A resident at home is described by: $\texttt{ResidentAtHome} \equiv \texttt{Resident} \sqcap \exists\texttt{locatedAt}.(\texttt{Home} \sqcup \exists\texttt{isPartOf}.\texttt{Home})$. Similarly a resident currently out of home is described by: $\texttt{ResidentOutOfHome} \equiv \texttt{Resident} \sqcap \exists\texttt{locatedAt}.(\texttt{Home} \sqcup (\exists\texttt{isPartOf}.\texttt{Home}))$. The social relations between residents and other persons are modeled based on the concepts neighbour and relative. A neighbour is a person who lives not at home but at a house nearby home: $\texttt{Neighbour} \equiv \texttt{Person} \sqcap \exists\texttt{livesAt}.(\neg\ \texttt{Home} \sqcap (\exists\texttt{nearby}.\texttt{Home}))$.

A relative is a relative of a resident by definition: $\texttt{Relative} \equiv \texttt{Person} \sqcap \exists\texttt{isRelativeOf}.\texttt{Resident}$. Service persons like police man or pizza boy are modeled as sub-concepts of person. In the ontology several service persons are modeled for example.

Modeling activity. The activity of residents is essential for determining the current presence of residents. Because activity usually can not be captured directly, it has to be derived from available information like time, location and schedule.

For modeling features of a location we use so called value partitions. A Value Partition is a design pattern to define an exhaustive list of values for a certain set. For instance, for describing the light level of a room the set of possible values can be defined by a value partition `LightLevel`. The possible values are: full, dimmed and off. Because Description Logics are based on the open world assumption, value partitions have to be used to restrict the valid values of a certain set (otherwise, further (currently undefined) values could be elements of this set). A covering axiom has to be defined to make the list of value types exhaustive. The value partition `Lightlevel` can be defined in form of a value partition as follows: $\texttt{LightLevel} \equiv \texttt{Off} \sqcup \texttt{Dimmed} \sqcup \texttt{Full}$ (where `Off`, `Dimmed` and `Full` are mutually disjoint).

1. AudioEnabledDevice ≡ Handy ⊔ SmartPhone ⊔ PDA ⊔ Laptop ⊔ Phone ⊔ DoorBell
2. SMSEnabledDevice ≡ Handy ⊔ SmartPhone ⊔ CellularPhone
3. AudioEnabledConnectedDevice ≡ AudioEnabledDevice ⊓
   ∃isConnectedVia.(∃ hasBandwidthLevel.(LowBandwidth ⊔ MediumBandwidth ⊔ HighBandwidth))
4. SMSEnabledConnectedDevice ≡ SMSEnabledDevice ⊓ ∃isConnectedVia.(WAN ⊔ PSTN)
5. AudioEnabledResidentContext ≡ ∃ hasContextResident.(Resident ⊓ (∃uses.AudioEnabledConnectedDevice))
6. DoorbellEnabledResidentContext ≡ ResidentAtHomeContext
7. ReachableInUrgencyContext ≡ ∃hasContextAgent.(FireFighter ⊔ Policeman)
8. DoNotDisturbResidentContext ≡
   ∃hasContextResident.(SleepingResident ⊔ VacationResident ⊔ (∃hasPresence.DoNotDisturb))
9. AudioReachableResidentContext ≡ AudioEnabledResidentContext ⊓ ReachableResidentContext

Fig. 6. Concept definitions from the Doors ontology.

The activity of a person can be set by definition or can be derived from contextual information. We have defined to classes of activities:

– `Vacation` containing the sub-concepts `BusinessTrip` and `Holiday`
– `Working` containing the sub-concepts `Reading` and `Writing`

To derive the activity of a resident the location and its features are used. For instance the activity sleeping of a resident is derived from the location `BedRoom` and the `LightLevel` of the `BedRoom` which have to be `Dimmed` or `Off`: `SleepingResident` ≡ `Resident atHome` ⊓ ∃`locatedAt`.(`BedRoom` ⊓ (∃`hasLightLevel`.(`Dimmed` ⊔ `Off`))).

Modeling devices, connectivity and presence. The reachability of a resident can be modeled based on the used devices and their connectivity. It describes how a resident can be contacted at a certain point in time from a technical point of view. The ontology contains concepts and properties to model the devices owned (role `isOwnedBy`) and used (role `isUsedBy`) by a certain person and the network links which are supported by these devices (role `supportsLink`).

Devices are modeled as sub-concepts of the root concept `Device`. A basic feature of each device is its mobility. Thus, each device is a sub-concept of either `Stationary` or `Mobile`. Stationary devices relevant for the scenario are `PC` and the `Doorbell`, relevant mobile devices are `CellularPhone`, `PDA` and `Laptop`. A device can be connected via a certain network link (role `isConnectedVia`).

Network links are discriminated into wired and wireless links. Wired links are for instance DSL and Ethernet, wireless links are subdivided into `PAN`, `LAN` and `WAN` links containing wireless links such as `Bluetooth`, `WLAN`, `GSM` and `UMTS`. Each network has a certain bandwidth level (role `hasBandwidthLevel`).

To describe the options of the door system to contact a resident the communication categories of text, audio and video are defined. Text communication can be further discriminated into SMS messages and instant messages. For each device the supported communication categories are defined, describing the communication capabilities of the devices (see definitions 1 and 2 in Figure 6.

To involve the bandwidth of the network connection the concept of the `EnabledConnectedDevice` is defined. Examples are the definitions 3 and 4 in Figure 6. The inferred concept hierarchy for the modeled devices is depicted in Figure 7.

Modeling situation types. A situation is described by the concept `Context` and its sub-concepts. Each situation is described by a person or group of persons ringing at the door (role `hasContextPerson`) and the residents of the house (role `hasContextResident`). In Figure 4.2 the complete hierarchy of situation concepts of the door-lock scenario are depicted.

To enable a decision of the door system to open the door automatically, ask a resident or let the ringing person leaving a message, two basic contexts are modeled which are called `DoorContext` and `PresenceContext`. The `DoorContext` contains concepts describing the aspects of the ringing person and if residents are at home or not. This supports the decision to open the door immediately if an authorized person is ringing (i.e., a resident or a person authorized by a resident). Furthermore, based on a black list the system can immediately decide to keep the door closed and to let the ringing person leave a message without contacting a resident. A necessary prerequisite therefore is to identify the person or their category.

If no clear decision is possible, one of the residents should be contacted. To decide how to contact a resident the system reasons about the `PresenceContext`.

The `PresenceContext` describes at one hand what communication modes can be used to contact a resident from the technical point of view with the concept `ConnectedContext`. On the other hand the willingness of a person to communicate is modeled with the concept `ReachableContext`. The `ConnectedContext` is defined in definition 5 and 6 in Figure 6.

To describe the presence of a resident we use for instance the definitions 7 and 8 in Figure 6. To combine the aspects of technical connectivity and reachability, the `PresenceContext` has been defined as in definition 9 in Figure 6.

4.2.3. Testing

To demonstrate the usage of the ontology we have defined several situations based on individuals of the concept hierarchy in the ABox. Two types of situations have been defined as instances of the concepts `DoorContext` and `PresenceContext`.

5. Evaluation

The proposed approach of modeling situations with OWL DL and the adoption of ABox realization for recognizing situation types focuses on the employment of DL systems. The choice of DL systems for the implementation of AmI systems has several consequences regarding the performance of DL reasoners, the expressivity of the formalism and the handling of imperfect context information. Therefore, we take a closer look at DL systems to assess their usefulness with respect to these requirements. Especially, we want to point out the advantages and limits of the presented approach.

5.1. Performance evaluation

The complexity of ABox reasoning for the DL employed in our case is NExpTime complete in the worst case, see [49]. However, these worst case complexities do not need to appear in concrete scenarios. Moreover, there exist a couple of highly optimized DL reasoners for OWL DL that behave well in practice. Since applications for Ambient Intelligence usually expect computation times no longer than a couple of seconds or even might require response within milliseconds in some application domains, it is interesting to get an impression of the performance of the reasoners for classification and realization. Some language constructs have higher worst case complexity and thus, might require longer computation times. A natural question is whether it is advisable to disallow these constructs from the ontology.

To answer these questions, benchmarks have been performed using Protegé-OWL 3.4 beta from Standford University running under Windows XP. As hardware platform a Lenovo T60 Laptop with 2 GByte of memory and an Intel Centrino Duo processor running at 2 GHz was used. To compare the performance of the reasoners the times for classifying the concepts defined in the TBox and for computing the inferred types from the individuals in the ABox of our ontology were measured. For the tests three different variants of our Doors ontology were used:

Doors: The knowledge base (KB) described in the last Section. contains 6 Situation individuals.

Doors-no-GCIs: Door KB without GCIs, i.e., disjointness and domain and range restrictions for roles were deleted.

Doors-easy-roles: transitivity, symmetric role and inverse roles statements as well as all domain and range restrictions were deleted. Functional roles were left in the TBox, since they do not increase the run-time dramatically.

The DL expressivity of the Doors and Doors-no-GCIs ontology variants is ALCHIF, while the DL expressivity of the Doors-easy-roles ontology is ALCF. All ontology variants contain 135 concepts, 98 individuals and 27 object properties. We tested four DL reasoners that implement ABox realization for OWL DL. We used the well-known system RACERPRO [22], the FACT++ system [51] and PELLET [42].

FACT++**:** is a successor system of the FACT system [26], that was a ground-breaking TBox reasoning system developed in the late nineties. FACT was the first DL system that could handle GCIs in an efficient way. FACT++ is implemented in C++. We used version 1.2.0 (25.9.2008) of FACT++ for our tests. In contrast to the other tableau-based systems, FACT++ implements an eager approach for realization. It sets up its data structures for realization already during the classification phase of the TBox. This results in longer run-times for classification, but speeds-up realization considerably, as we will see.

PELLET**:** The PELLET system is developed at the University of Maryland in 2003. It supports DL reasoning services for the DL SHOIQ(D) - where the qualified number restrictions are limited to 1 or 0

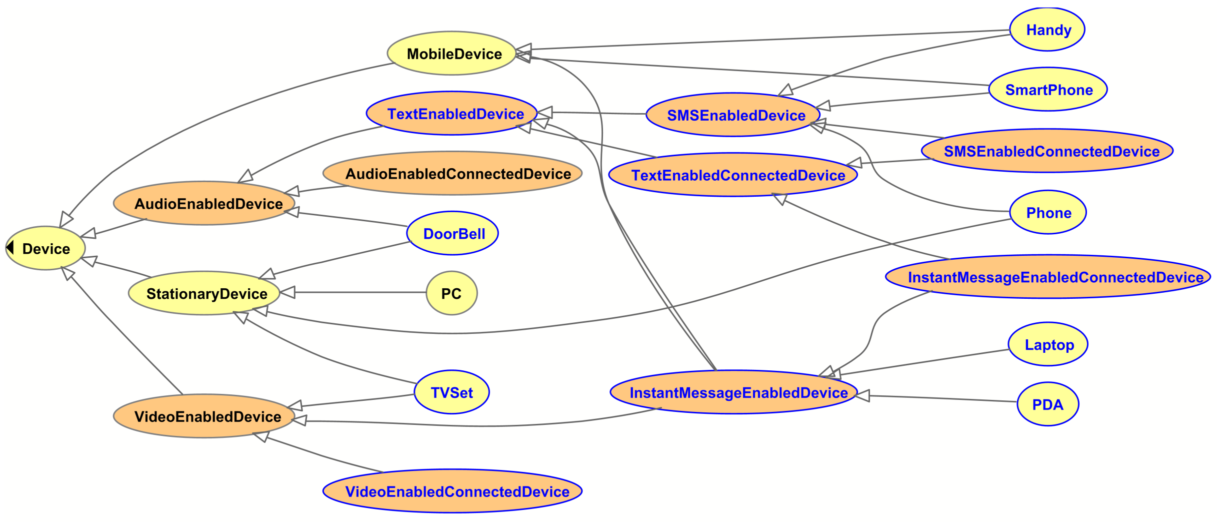
MobileDevice
Handy
SmartPhone
TextEnabledDevice
SMSEnabledDevice
SMSEnabledConnectedDevice
AudioEnabledConnectedDevice
TextEnabledConnectedDevice
AudioEnabledDevice
DoorBell
Phone
Device
StationaryDevice
PC
InstantMessageEnabledConnectedDevice
Laptop
TVSet
InstantMessageEnabledDevice
PDA
VideoEnabledDevice
VideoEnabledConnectedDevice

Fig. 7. Inferred concept hierarchy of device concept in the doors ontology.

and the concrete domains to what the OWL standard supports. Thus the PELLET system supports the full range of concept constructors of OWL DL. Besides the classical DL reasoning services, PELLET offers a range of additional functionality. For a full overview, please refer to the PELLET web pages [42]. In our evaluation of the systems we used PELLET 1.5.2 (1.5.2008).

RACERPRO: is the successor system of RACER provided by RacerSystems. This system is a commercial system with free licenses for academic institutions. In comparison to RACER, RACERPRO shows a much better performance. Furthermore, RACERPRO offers more functionality than most other DL reasoners [34,35], which helps to maintain and query the KB. As its successor RACERPRO also supports more expressive concrete domains than the OWL standard demands. For our evaluation we used the currently available stable version 1.9.0 of RACERPRO and the beta version of RACERPRO 1.9.2 beta.

With KAON2 [29] and HermiT [30] other DL reasoners are available. We have not included these reasoners in our test because of the lack of ABox realization reasoning service adopted in our approach.

The measured run-times are shown in Table 2. It contains measured times for the four reasoners. For each reasoner we measured the times for classifying all concepts of the TBox in our test ontologies and to perform ABox realization, as stated in Definition 7. The measured times for the classification task are below 1s, except for PELLET. While classification is only performed once during the start of the AmI system, these times are mainly relevant for the design time. Fast computation times especially allow for an interactive development of ontologies. The results for this reasoning tasks are much more heterogeneous. For the Doors ontology FACT++ needs orders of magnitude less time than PELLET. As stated in the description of FACT++ above, an eager approach was used for the implementation of the ABox realization. Thus, data structures for realization are already set up during the classification of the TBox. Compared to RACERPRO 1.9.2 beta, FACT++ needs a little more time for classification than RACERPRO. The ABox realization task is than performed much faster using FACT++ than using RACERPRO.

For the Doors-no-GCIs ontology the deletion of GCIs (see Definition 1) speeds-up the realization of the individuals. Especially, PELLET profits from that deletion. It performs the classification and realization tasks one order of magnitude faster than for the complete Doors ontology. But also both RACERPRO versions need less time for processing the Doors-no-GCIs ontology. The reduction of the Doors ontology to the Doors-easy-roles ontology causes an improvement of the computation times of classification for all reasoners. Anyway, only the both versions of RACERPRO can profit with respect to ABox realization. FACT++ and PELLET need more time for the realization of the Doors-easy-roles ontologies ABox than for the Doors ontology ABox.

Table 2
TBox classification and ABox realization run-times (in s)

	Doors		**Doors-no-GCIs**		**Doors-easy-roles**	
	TBox classification	ABox realization	TBox classification	ABox realization	TBox classification	ABox realization
FaCT++	0,8	0,02	0,71	0,02	0,73	0,06
Pellet	3,79	18,45	0,66	2,19	2,57	30,85
RacerPro 1.9.0	0,6	2,02	0,48	1,13	0,38	0,76
RacerPro 1.9.2 beta	0,62	0,74	0,39	0,2	0,26	0,51

At one hand the reduction of language constructs used in the ontology may improve the performance of DL reasoners. On the other hand reduction comes at the cost of missing implicit subsumption or instance relations. For the Doors-no-GCIs ontology several of these relations can no longer be detected and not the most specific context can be recognized from the information available. So, it is clearly not advisable for this application to degrade expressivity to obtain better run-times. In general, this trade-off should be considered during the development of AmI systems following the proposed approach.

To sum up, today's DL reasoners can compute realization for ABoxes in a run time acceptable for the the intelligent door lock scenarios. FaCT++ provides the best performance for ABox realization. While both newer version of RacerPro also responds in times below 1s for ABox realization, Pellet is significantly slower with that task. Compared to previous measurements carried out with older versions of the reasoners but using the same ontologies [52], reasoners show up significant improvements in performance for both TBox classification and ABox reasoning. Because all reasoners are currently under development performance improvements can be expected in the future.

5.2. *Expressivity Considerations*

In addition to the performance of DL reasoners it is also important to what degree OWL DL supports the developer with the creation of situation descriptions. Major questions are how complex a situation description can be, what aspects of a situation can be modeled and which not, if it is transparent how situation types are recognized and to what degree situation models can be refined and adjusted at run-time and during the AmI systems life-cycle. Moreover, the modeling effort should be assessed.

From the experiences of the authors the modeling of aspects of situation types in the TBox is an intuitive task. Situation aspects are described as concepts, refinements of concepts can be expressed by defining a more specific context as a subconcept of an already defined concept. This way of thinking about the world is intuitive for developers, especially if they are familiar with object orientation. Similar to this approach, child concepts in OWL DL inherit all properties of their parent concepts. Based on the creation of a concept hierarchy even complex scenarios remain to be manageable when modeled with OWL DL.

In addition, the recognition of situation types is completely transparent for the system developer. If the design methodology proposed in Section 3 is adopted, during the activity of testing the developer creates a test ABox and performs ABox realization on the ontology to compute the most specific concepts characterizing a situation. If the TBox is specified correctly, the system behaves according to that specification. In contrast to DL systems, in case of adopting neuronal networks or Bayesian networks for situation recognition the phase for decision making is also performed by the network. For the developer it is not transparent how and why a certain situation was detected and a particular action was triggered.

Anyway, OWL DL also has some limitations a developer should be aware of. One of the most obvious limits of OWL DL was the limited expressivity and reasoning capabilities regarding numbers. One would like to specify ranges of numbers to map them to qualitative measures, like slow := has-velocity. ($<$ 0) and ($>$ 50). Another application of numbers would be to compare, for example the velocity of the user with the maximal velocity that is supported by a certain network technology. Although OWL DL offers XML data types for the use of numbers in ontologies, they do not allow to model the facts just mentioned, since there are only unary predicates available in OWL. One would like to model these facts by the use of concrete domains [4], which are supported by RacerPro. We tried to emulate the ranges by number restrictions in an earlier version of the ontology, but since it was a very unintuitive way of modeling and the performance of the reasoners was slowed down drastically, we gave up this approach immediately.

Often one would like to express facts such as "the provider of the available network is the same provider that the user has a contract with" – sometimes called agreements. The underlying DL concept constructor for this is called *role value maps* (or feature chain agreements). Unfortunately it is a well-known result that these constructors make reasoning for even small fragments of OWL DL (e.g., $\mathcal{ALC}$) undecidable. Thus, these concept constructors were not included in OWL DL. We tried to capture the above mentioned facts by the use of inverse roles. Sometimes the above mentioned agreements do not only refer to a single concept, but to roles. For example the role one would like to add "$\texttt{hasUncle} \equiv \texttt{hasParent} \circ \texttt{hasBrother}$" to the ontology. These kind of statements are not supported in OWL 1.0. However, it is planned to include them in the forthcoming OWL 1.1 standard.

In the step of the authoring of the knowledge base some typical effects can occur that lead to unexpected or unintuitive reasoning results.

Primitive definition vs. full definition: Sometimes when only necessary, but not the sufficient conditions are supplied for a concept C, by giving only a primitive definition and a subsumption relation seems to be missing.

Open world vs. closed world: Especially for users who have worked with systems that use closed world semantics (such as Prolog or Data bases) find it unintuitive, if from leaving out the fact that an individual is, for example located in the house, it cannot be derived that the individual is outside of the house. An insistent case of this are at-most restrictions, which can hardly ever be derived, but have to be supplied by the modeler in most cases.

Value restrictions vs. domain & range restrictions: often it is not clear whether the restriction of role-fillers to a certain type of concepts is only valid for a concept C or whether it is a property of a role.

Last but not least, the extensibility of the knowledge base at run-time plays an important role, especially in very dynamic scenarios. As a fact, situation descriptions are modeled at design time. That means, all relevant situations have to be known at design time. Moreover, situation descriptions depend on a static set of context features, i.e., new context types can't be involved in the situation model at run-time. Especially, new aspects of a situation can't be added at run-time even if our situation model is extensible at design time.

5.3. Handling of context characteristics

For our approach we see the handling of the described characteristics at different stages of context processing which we understand as a stepwise execution of operations for context interpretation, aggregation and derivation to produce higher-level context which can be used at application level. We assume that most of the processing is done within a context service (see Figure 4.1). While the application requests as set of context features according to a context profile at a certain point in time (what we call "to make a snapshot") the context service is responsible for handling dynamics to provide up to date information. This includes to provide the history if requested by the application. As for a certain context value multiple alternative sources can be available, they have to be maintained by the context service. These alternatives can be exploited to handle incorrectness and quality variations. The context service can apply operations for choosing the context value with the highest available quality out of several alternative values. Furthermore, it can compare alternative values to detect incorrect values.

Moreover, DL reasoning can handle *incomplete information* gracefully. Incompleteness can be detected by the context service based on the context profile. Nevertheless, we can handle it in the DL system because meaningful reasoning is also possible on incomplete data.Even if the context service can provide a subset of the requested context information, realization still is able to recognize concepts, even if they might be more general. However, even in such cases the system is still able to recognize context and the application can perform actions associated with that context.

Inconsistency can also be detected by the DL system and can cause the context application to request additional, clarifying information from the context service. We currently see the handling of data quality out of scope of the application level model, assuming that the handling is done by the context service as described above. We have so far focused on modeling the application domain, especially the context concepts useful for certain application tasks and reasoning about the situation the application is within.

Moreover, the approach to use DL reasoning for the recognition of contexts offers a graceful way of handling incomplete data. In such a case the realization would simply return contexts that might be too general, but still a context is recognized and an action associated with the returned context will be performed. Furthermore, the separation of context recognition and choice of action allows adapting this association at run-time according to user preferences.

6. Conclusions and Future work

We proposed a method for situation modeling using the Description Logics based ontology language OWL DL and a framework for employing Description Logics reasoning services to recognize the current situation based on context. Especially, we employed the DL reasoning service ABox realization to identify the most specific types of a situation, representing the current situation. According to the introduced framework, the application then determines the actions to be performed in correspondence to the current situation.

The benefits from the approach are manifold: the semantics of Description Logics allow for graceful handling of incomplete knowledge. The well-investigated reasoning services do not only allow recognizing the current situation, but also can add to the reliability of the overall system. Moreover optimized reasoning systems are freely available and ready to use.

On the one hand the explicit specification of the situation model in the TBox may limit the ways in which an AmI system can adapt at run-time to changing situation types, but on the other hand the explicit specification clearly provides transparency to application developers and users and thus contributes to the reliability of the overall system. In addition, our approach requires neither a learning phase nor a large amount of training examples to set-up the initial knowledge base. However, it does not support full dynamic adaptation despite the separation into situation type inference and decision making, which allows a certain degree of flexibility at run-time.

The feasibility of the approach has been demonstrated with a case study based on a smart home application. The aspects of situations from that scenario could be modeled completely. The evaluation has shown that DL systems support the characteristics of context, especially the issue of incomplete knowledge can be handled very well. Further characteristics like heterogeneous and distributed context sources as well as multiple alternative context sources can also be handled in our framework, this is achieved by the context service adopted in our framework.

The performance of DL reasoners is appropriate for scenarios which can tolerate response times of a few seconds. Dependent on the chosen reasoner, the constructs used in the ontology and the number of concepts, roles and individuals run-times in the range of below 1s up to several second can be achieved by the currently available reasoning tools. While these tools are under development, mainly driven by the strong community of Semantic Web research, significant performance improvements can be expected for the future.

As pointed out in the evaluation in Section 5.2 there are also some limitations of the expressivity of DLs, a developer of AmI systems should be aware of. Solutions to overcome these limitations exists. Especially, mathematical or classification operations dealing with numbers can already be performed during the context-awareness phase. The approach presented in [46] supports a step-wise abstraction process of sensor measurements and can be seen as one possible solution for the issue of missing expressivity and reasoning capabilities regarding numbers in DLs. Another solution is to model these facts by the use of concrete domains [4], which are supported by RACERPRO.

To sum it up, OWL DL is extremely helpful as a standard, since it encourages the implementation of a lot of ontology tools and reasoners helpful for context applications. As a modeling language it offers wide range of language constructs that allow to model a lot of complex notions from context applications. However, some concept constructors central to modeling with numbers are missing in OWL DL.

The research work presented in this article was partially sponsored by Siemens AG. Moreover, the authors would like to thank Bootawee Suntisrivaraporn from TU Dresden and Michael Pirker from Siemens AG, Intelligent Autonomous Systems for their helpful comments on earlier versions of this paper.

References

[1] A. Agostini, C. Bettini, and D. Riboni. A performance evaluation of ontology-based context reasoning. In *Proc. of Fifth Annual IEEE International Conference on Pervasive Computing and Communications - Workshops*, pages 3–8. IEEE Computer Society, 2007.

[2] C. B. Anagnostopoulos, Y. Ntarladimas, and S. Hadjiefthymiades. Situational computing: An innovative architecture with imprecise reasoning. *Journal of Systems and Software*, 80(12):1993–2014, 2007.

[3] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. Patel-Schneider, editors. *The Description Logic Handbook:*

Theory, Implementation, and Applications. Cambridge University Press, 2003.

[4] F. Baader and P. Hanschke. A Schema for Integrating Concrete Domains into Concept Languages. In *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence (IJCAI-91)*, pages 452–457, Sydney, 1991.

[5] F. Baader, C. Lutz, and B. Suntisrivaraporn. CEL—a polynomial-time reasoner for life science ontologies. In U. Furbach and N. Shankar, editors, *Proc. of the 3rd Int. Joint Conf. on Automated Reasoning (IJCAR-06)*, volume 4130 of *Lecture Notes In Artificial Intelligence*, pages 287–291. Springer-Verlag, 2006. CEL download page: `http://lat.inf.tu-dresden.de/systems/cel/`.

[6] F. Baader and W. Nutt. *[3]*, chapter Basic Description Logics, pages 43–96. Cambridge University Press, 2003.

[7] F. Baader and U. Sattler. An overview of tableau algorithms for description logics. *Studia Logica*, 69:5–40, 2001.

[8] P. Baker, C. Goble, S. Bechhofer, N. Paton, R. Stevens, and A. Brass. An ontology for bioinformatics applications. *Bioinformatics*, 15(6):510–520, 1999.

[9] S. Bechhofer. The dig description logic interface: Dig/1.1. Technical Report, 2003.

[10] S. Bechhofer, F. van Harmelen, J. Hendler, I. Horrocks, D. L. McGuinness, P. F. Patel-Schneider, and L. A. Stein. OWL web ontology language reference. W3C Recommendation, February 2004. `http://www.w3.org/TR/owl-ref/`.

[11] T. Berners-Lee, J. Hendler, and O. Lassila. The Semantic Web. *Scientific American*, 284(5):34–43, 2001.

[12] R. Brachman and H. Levesque. *Readings in Knowledge Representation*. Morgan Kaufmann, Los Altos, 1985.

[13] G. Chen, M. Li, and D. Kotz. Design and implementation of a large-scale context fusion network. pages 246 – 255, Aug. 2004.

[14] H. Chen, F. Perich, T. Finin, and A. Joshi. SOUPA: Standard Ontology for Ubiquitous and Pervasive Applications. In *International Conference on Mobile and Ubiquitous Systems: Networking and Services*, Boston, MA, August 2004.

[15] E. Christopoulou, C. Goumopoulos, and A. Kameas. An ontology-based context management and reasoning process for ubicomp applications. In *sOc-EUSAI '05: Proceedings of the 2005 joint conference on Smart objects and ambient intelligence*, pages 265–270, New York, NY, USA, 2005. ACM Press.

[16] R. Cornet and A. Abu-Hanna. Using description logics for managing medical terminologies. In P. B. M. Dojat, E. Keravnou, editor, *Artificial Intelligence in Medicine: 9th Conference on Artificial Intelligence, in Medicine in Europe (AIME 2003)*, Lecture Notes in Computer Science, pages 61–70. Springer, 2003.

[17] R. Cote, D. Rothwell, J. Palotay, R. Beckett, and L. Brochu. The systematized nomenclature of human and veterinary medicine. Technical report, SNOMED International, Northfield, IL: College of American Pathologists, 1993.

[18] A. K. Dey, D. Salber, and G. D. Abowd. A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. *Human-Computer Interaction (HCl) Journal*, 16(2–4):97–166, 2001.

[19] T. Gu, H. Pung, and D. Zhang. Toward an OSGi-based infrastructure for context-aware applications. *IEEE Pervasive Computing*, 3(4):66–74, Oct.-Dec. 2004.

[20] V. Haarslev, R. Möller, and M. Wessel. Querying the semantic web with racer + nrql. In *In Proceedings of the KI-2004 International Workshop on Applications of Description Logics (ADLŠ04*, 2004.

[21] V. Haarslev and R. Möller. Racer: A core inference engine for the semantic web. In *Proc. of the 2nd International Workshop on Evaluation of Ontology-based Tools (EON2003), located with ISWC*, pages 27–36, 2003.

[22] V. Haarslev, R. Möller, and M. Wessel. RacerPro reasoner, 2005. See `http://www.racer-systems.com/`.

[23] C. Haase and C. Lutz. Complexity of subsumption in the $\mathcal{EL}$ family of description logics: Acyclic and cyclic tboxes. In M. Ghallab, C. D. Spyropoulos, N. Fakotakis, and N. Avouris, editors, *Proc. of the 18th European Conference on Artificial Intelligence (ECAI08)*, volume 178 of *Frontiers in Artificial Intelligence and Applications*, pages 25–29. IOS Press, 2008.

[24] I. Horrocks, P. Patel-Schneider, and F. van Harmelen. From $\mathcal{SHIQ}$ and RDF to OWL: The making of a web ontology language. *Journal of Web Semantics*, 1(1):7–26, 2003.

[25] I. Horrocks and U. Sattler. A tableau decision procedure for $\mathcal{SHOIQ}$. *J. of Automated Reasoning*, 39(3):249–276, 2007.

[26] I. R. Horrocks. Using an expressive description logic: Fact or fiction. pages 636–647, 1998.

[27] P. Korpipää, J. Mäntyjärvi, J. Kela, H. Kernen, and E.-J. Malm. Managing context information in mobile devices. *IEEE Pervasive Computing*, 2(3):42–51, 2003.

[28] M. Luther, Y. Fukazawa, M. Wagner, and S. Kurakake. Situational reasoning for task-oriented mobile service recommendation. *The Knowledge Engineering Review*, 23(Special Issue 01):7–19, 2008.

[29] B. Motik and U. Sattler. A Comparison of Techniques for Querying Large Description Logic ABoxes. In M. Hermann and A. Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning*, volume 4246 of *Lecture Notes in Computer Science*, pages 227–241, Phnom Penh, Cambodia, November 13–17 2006. Springer. KAON2 download page: `http://kaon2.semanticweb.org/`.

[30] B. Motik, R. Shearer, and I. Horrocks. Optimized Reasoning in Description Logics using Hypertableaux. Lecture Notes in Artificial Intelligence, pages 67–83, Bremen, Germany, July 17–20 2007. Springer.

[31] B. Mrohs, M. Luther, R. Vaidya, M. Wagner, S. Steglich, W. Kellerer, and S. Arbanowski. OWL-SF–a distributed semantic service framework. In *Proc. of the Workshop on Context Awareness for Proactive Systems (CAPS'05), Helsinki*, pages 67–77, 2005.

[32] V. Peltonen, J. Tuomi, A. Klapuri, J. Huopaniemi, and T. Sorsa. Computational auditory scene recognition. In *In IEEE IntŠl Conf. on Acoustics, Speech, and Signal Processing*, pages 1941–1944, 2002.

[33] M. R. Quillian. Word concepts: A theory and simulation of some basic capabilities. *Behavioral Science*, 12:410–430, 1967. Republished in [12].

[34] Racer Systems GmbH & Co. KG. Racerpro reference manual version 1.9, dec. 2005., 2005.

[35] Racer Systems GmbH & Co. KG. Racerpro User's guide version 1.9, dec. 2005., 2005.

[36] C. Ramos, J. C. Augusto, and D. Shapiro. Ambient intelligence – the next step for artificial intelligence. *IEEE Intelligent Systems*, 23(2):15–18, March-April 2008.

[37] A. Ranganathan and R. Campbell. An infrastructure for context-awareness based on first order logic. *Personal and Ubiquitous Computing*, (7):353–364, 2003.

[38] A. Rector. Medical informatics. In *[3]*, pages 406–426. Cambridge University Press, 2003.

[39] S. Schulz, B. Suntisrivaraporn, and F. Baader. SNOMED CT's problem list: Ontologists' and logicians' therapy suggestions. In *Proceedings of The Medinfo 2007 Congress*, Studies in Health Technology and Informatics (SHTI-series). IOS Press, 2007.

[40] N. Shadbolt. Ambient intelligence. *IEEE Intelligent Systems*, 18(4):2–3, 2003.

[41] K. Sheikh, M. Wegdam, and M. van Sinderen. Middleware support for quality of context in pervasive context-aware systems. In *PERCOMW '07: Proceedings of the Fifth IEEE International Conference on Pervasive Computing and Communications Workshops*, pages 461–466, Washington, DC, USA, 2007. IEEE Computer Society.

[42] E. Sirin and B. Parsia. Pellet: An OWL DL reasoner. In V. Haarslev and R. Möller, editors, *Proc. of the 2004 Description Logic Workshop (DL 2004)*, number 104 in CEUR Workshop, 2004. See also `http://www.mindswap.org/2003/pellet/index.shtml`.

[43] E. Sirin and B. Parsia. Pellet system description. In B. Parsia, U. Sattler, and D. Toman, editors, *Description Logics*, volume 189 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2006.

[44] J. F. Sowa, editor. *Principles of Semantic Networks*. Morgan Kaufmann, Los Altos, 1991.

[45] T. Springer, K. Kadner, F. Steuer, and M. Yin. Middleware support for context-awareness in 4G environments. In *World of Wireless, Mobile and Multimedia Networks, 2006. WoWMoM 2006. International Symposium on a*, pages 203–211, 26-29 June 2006.

[46] T. Springer, P. Wustmann, I. Braun, W. Dargie, and M. Berger. A comprehensive approach for situation-awareness based on sensing and reasoning about context. In F. E. Sandnes, Y. Zhang, C. Rong, L. T. Yang, and J. Ma, editors, *UIC*, volume 5061 of *Lecture Notes in Computer Science*, pages 143–157. Springer, 2008.

[47] V. Stankovski and J. Trnkoczy. Application of decision trees to smart homes. In *Designing Smart Homes*, volume 4008 of *Lecture Notes in Computer Science*, pages 132–145. Springer Berlin / Heidelberg, 2006.

[48] T. Strang and C. Linnhoff-Popien. A context modeling survey. In *In: Workshop on Advanced Context Modelling, Reasoning and Management, UbiComp 2004 - The Sixth International Conference on Ubiquitous Computing, Nottingham/England*, 2004.

[49] S. Tobies. The complexity of reasoning with cardinality restrictions and nominals in expressive description logics. *Journal of Artificial Intelligence Research*, 12:199–217, May 2000.

[50] D. Tsarkov and I. Horrocks. FaCT++ description logic reasoner: System description. In *Proc. of the 3rd Int. Joint Conf. on Automated Reasoning (IJCAR-06)*, 2006. FaCT++ download page: `http://owl.man.ac.uk/factplusplus/`.

[51] D. Tsarkov and I. Horrocks. FaCT++ description logic reasoner: System description. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 4130:292–297, 2006.

[52] A.-Y. Turhan, T. Springer, and M. Berger. Pushing doors for modeling contexts with OWL DL – a case study. In J. Indulska and D. Nicklas, editors, *Proceedings of the Workshop on Context Modeling and Reasoning (CoMoRea'06)*, pages 13–17. IEEE Computer Society, March 2006.

[53] M. Weiser. The Computer for the 21st Century. *Scientific American*, pages 66–75, Sep 1991.

[54] M. Wessel, M. Luther, and M. Wagner. The difference a day makes – recognizing important events in daily context logs. In P. Bouquet, J. Euzenat, C. Ghidini, D. L. McGuinness, L. Serafini, P. Shvaiko, and H. Wache, editors, *Proc. of the International Workshop on Contexts and Ontologies: Representation and Reasoning (C&O:RR)*, volume 298 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2007.

[55] K. Wolstencroft, A. Brass, I. Horrocks, P. Lord, U. Sattler, R. Stevens, and D. Turi. A little semantic web goes a long way in biology. In *Proc. of the 2005 International Semantic Web Conference (ISWC 2005)*, number 3729 in Lecture Notes in Computer Science, pages 786–800. Springer, 2005.

[56] K. Wolstencroft, P. W. Lord, L. Tabernero, A. Brass, and R. Stevens. Protein classification using ontology classification. In *In Proceedings 14th International Conference on Intelligent Systems for Molecular Biology ISMB'06 (Supplement of Bioinformatics)*, pages 530–538, 2006.