Pushing Doors for Modeling Contexts with OWL DL –a Case Study

Anni-Yasmin Turhan* and Thomas Springer** and Michael Berger†

* TU Dresden, Chair for Automata Theory, turhan@tcs.inf.tu-dresden.de

** TU Dresden, Chair for Computer Networks, springet@rn.inf.tu-dresden.de

† Siemens AG, Corporate Technology, Intelligent Autonomous Systems, mberger@siemens.com

Abstract

In this paper we present an integrated view for modeling and reasoning for context applications using OWL DL. In our case study, we describe a task driven approach to model typical situations as context concepts in an OWL DL ontology. At run-time OWL individuals form situation descriptions and by use of realization we recognise a certain context. We demonstrate the feasibility of our approach by performance measurements of available highly optimised Description Logics (DL) reasoners for OWL DL.

1 Introduction and Motivation

Within the last years, context-awareness has evolved from a vision into a key technology for pervasive and ubiquitous computing. By enabling applications to be aware of their current situation, they are able to adapt their behaviour to the changing environment to simplify user interactions and to improve system behaviour to operate more personalized, autonomous and flexible. One of the main tasks in context applications is to recognise the situation and to invoke appropriate actions. This paper proposes an ontology based representation of context information and, more importantly, employs well-defined reasoning services for recognising application relevant situations by employing highly optimised DL reasoners.

Context information is often not explicitly available, but has to be gathered from highly distributed and heterogeneous sources, such as sensor systems, data bases frameworks or other applications were it is available at different levels of abstraction. While a sensor system provides low-level information sensed from the environment, data bases and applications usually provide higher level information. In order to be useful in applications, higher level context often has to be obtained by aggregating, deriving and interpreting the output of context sources in a stepwise fashion. Caused by the way context is gathered and processed, it has special characteristics influencing models and processing of context. It reflects the state of a highly dynamic environment which changes over time and has a history. Since context information is sensed, extracted or derived, it has a certain *quality* and may produce *incorrect* information. A set of context values can be *inconsistent* and might be *incomplete* simply due to the (temporal) lack of context sources. A context application should be able to handle these difficulties in a graceful way. As we will see, DL-based systems can offer means to detect inconsistencies and to reason with incomplete knowledge.

Recent research efforts concentrate on generic solutions for introducing different levels of abstraction for processing (see [1, 2]). The approach described in [1] uses logic rules to determine the current situation of an application. While the approach to use a logical formalism is somewhat similar to our approach, no DL based reasoning was used. More recent projects (e.g. [3] and [4]) covered the creation of comprehensive and generic models of context with the goal to integrate context information characteristics. Furthermore, ontology based modeling and reasoning in OWL for context applications is addressed in, e.g., [5], [6]. However, the authors of these papers do not use formally defined reasoning methods, but rather employ ad-hoc methods that are capable of detecting some of the implicit knowledge, but that are not complete. To the best of our knowledge this paper is the first that studies DL reasoning methods for the context domain based on OWL DL ontologies.

Figure 1 shows the main components in the architecture of our context application. An event triggers the context application to request a *context profile* which groups a set of context data necessary to recognise the current situation of the application. The context application sends the profile to the context service were it is filled by making a “snapshot” of the currently available information. The context application adds the information from the filled profile to the knowledge base of the DL-system (in OWL DL format). The DL reasoner infers the category of context the current situation belongs to and the context application decides what actions have to be be invoked for the recognised situation.

In our approach characteristics of context information are handled at different stages of processing. The context service processes low level context information to obtain higher level information. Alternative context sources have to be maintained by the context service and can be exploited to handle incorrectness or quality variations, e.g. by choos-

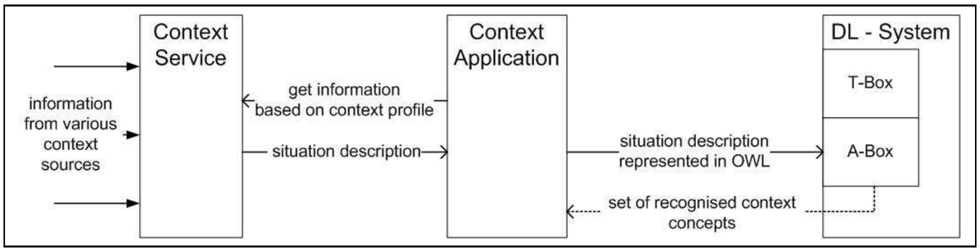

Figure 1. Main components of our context framework.

ing the context value from the context source with the best quality. Furthermore, the context service can compare alternative values to detect incorrect values. Incompleteness of context information can be detected by the context service based on the context profile. However, our approach can handle incompleteness in later processing steps in the DL-System, since DL systems can reason with incomplete information.

1.1 Application: context-aware door lock

Our case study is based on 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 authorised.
2) Ask a resident in case the person is unknown.
3) No reaction/leave message, if no resident is available.

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. fire fighter). The door opens for authorised 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 used TV set. 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 door system informs the person at the door, offers to leave a message and leaves 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 authorised by a resident to enter the house. If in addition all residents are on vacation, the neighbour can be recognised as an authorised 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.

1.2 Contributions

There is a common agreement that context-aware systems can benefit from ontology based approaches to context modeling and reasoning. However, many approaches focus on modeling and knowledge sharing, instead of what to infer from the ontology and what reasoning services to use. Studies on what the implications are on context models are rare. To fill this gap to some extend we present an integrated approach to model and to recognise contexts based on DL reasoning services.

For our case study, we describe the modeling of situations typically appearing in the application as *context concepts* in an OWL DL ontology. We characterised situations using a *task driven approach*, analysing the application scenario according to what tasks the application must fulfill and in what kind of contexts. Situations are represented as OWL individuals which are generated from context information. We use the reasoning service realization to determine into which context concepts a specific situation individual falls.

OWL DL is equipped with formal semantics that facilitates well-defined reasoning services. Moreover, in our case the reasoning services are proven to be sound and complete and terminating, which means *all* implicit relationships can be detected — not only some of them as it might be the case with ad-hoc reasoning procedures. In addition to this, highly optimised DL reasoners for OWL DL are available and can be used off the shelf. We present performance measurements comparing several reasoners and to assess the feasibility of our approach in practice.

2 Components of a framework for context

OWL DL is a W3C standard for an ontology language founded on DLs [7]. Our approach to model and to recognise contexts uses DL reasoning services.

Table 1. OWL syntax, DL syntax and semantics of concept descriptions.

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

2.1 Description Logic Systems

DLs [8] are a class of knowledge representation formalisms, which can be used to represent the knowledge of an application domain in a structured and formally well-understood way. DLs are equipped with formal semantics, which can, e.g., be given by a translation into first-order predicate logic.

Typically, a DL system consists of a *knowledge base* (KB) together with certain *reasoning services*. The knowledge base comprises two components, *TBox* and *ABox*. Intuitively, the TBox forms the ontology and defines the vocabulary by which a concrete world is described in the ABox. Both are defined by means of *concepts*. Concepts are built from concept names and role names using concept operators, whose syntax is introduced in Table 1 for a subset of OWL DL operators. For conciseness and readability we use the DL syntax throughout the paper. The semantics of a concept is defined in terms of an *interpretation* $\mathcal{I} = (\Delta, \cdot^I)$. The domain Δ of $\mathcal{I}$ is a non-empty set and the interpretation function $\cdot^I$ maps each concept name A to a set $A^I \subseteq \Delta$ and each role name r to a binary relation on Δ. The extension of $\cdot^I$ to arbitrary concepts is defined inductively, as shown in Table 1. TBoxes assign names to complex concepts. ABoxes relate individuals to concepts and to other individuals via roles, see [8].

Example 1 *To illustrate the use of the constructors, we define some main concepts from our application example:*

$\mathsf{VacationResident} = \mathsf{Resident} \sqcap \exists \mathsf{hasActivity.Vacation}$

$\mathsf{AuthorisedPerson} = \mathsf{Resident} \sqcup \exists \mathsf{AuthorisedBy.Resident}$

$\mathsf{AuthorisedNeighbour} = \mathsf{Neighbour} \sqcap (\exists \mathsf{AuthorisedBy.VacationResident})$

A resident on vacation is defined as a resident who has the activity vacation. An authorised person is defined as either a resident or someone authorised by a resident. An authorised neighbour in turn is a neighbour, who is authorised by a resident on vacation.

DL systems provide users with various reasoning services that deduce implicit knowledge from the explicitly represented knowledge. Among these inference services subsumption and realization are used in our context application. *Subsumption* determines subconcept-superconcept relationships of concept names occurring in a TBox, and hence to compute the concept hierarchy: C is subsumed by D iff all instances of C are also instances of D in every interpretation. The computation of all subsumption relations in a TBox is called *classification*. *Realization* computes to which concepts a given individual necessarily belongs, i.e., whether this instance relationship logically follows from the descriptions of the concept and of the individual.

These reasoning services are investigated for a great range of DLs (see [9]) and sound and complete algorithms are devised for them. The latter is a big benefit for our application since *all* implicit subsumption or instance relationships can be detected, which is not necessarily the case for ad-hoc reasoning methods, which might fail to detect some implicit relationships.

2.2 Framework for the context application

The framework and general procedure of processing contexts is outlined in Figure 1. The main idea is to use ABox realization as a means to recognise a context from a given situation description. To this end we model the contexts that typically appear in the application as concepts in the TBox and individual situation descriptions as ABox individuals.

Recall our task-based notion of contexts: a context comprises all the information necessary to solve a task. The context TBox contains a concept for each of these contexts – called *task contexts* in the following. Each task context is refined by a hierarchy of sub-concepts, that further specify and refine aspects of this kind of context. In our door lock scenario, the first task is to make a choice between: "open door", "ask resident" or "leave door closed". Only if necessary, the task how to contact the resident is solved subsequently, thus we obtain the task contexts "door context" and "contact resident context" for our scenario. Figure 1 shows the interaction of the components to solve one task. The context application sends a request with a context profile to the context server. The context server retrieves the requested information about the current situation by the information sources at hand, maps them to the context profile by processing them further and sends it to the context application. Once this situation description is obtained, it is converted to OWL DL and added to the ABox of the DL system. This situation individual is then realised by the DL

$\mathsf{DoorContext} =$
$\quad \mathsf{Context} \sqcap \exists \mathsf{hasContextAgent.(Person} \sqcap \exists \mathsf{isRinging.Home)}$
$\quad \sqcap\ \exists \mathsf{hasContextResident.Resident}$

$\mathsf{AuthorisedPersonRingingContext} =$
$\quad \mathsf{DoorContext} \sqcap \exists \mathsf{hasContextAgent.AuthorisedPerson}$

$\mathsf{AuthorisedNeighbourRingingContext} =$
$\quad \mathsf{DoorContext} \sqcap \exists \mathsf{hasContextAgent.AuthorisedNeighbour}$

$\mathsf{ResidentOutOfHomeContext} =$
$\quad \mathsf{DoorContext} \sqcap \exists \mathsf{hasContextResident.ResidentOutOfHome}$

Figure 2. Concept definitions of door context.

system and the most specific contexts instantiated by the situation individual are returned to the context application. The application looks-up and carries out the action associated with the returned set of contexts. Note, that the context profiles are set up before run-time and that the TBox is set up and classified before run-time as well.

Our framework relies on assumptions regarding the context service, e.g., the situation description being consistent. Furthermore, a method for the mapping of data available from context sources to the context profile by the context service remains future work. However, the approach to use DL reasoning for the recognition of contexts offers a graceful way of handling incomplete data. In such a case realization returned contexts that simply might be a little too general, but still a context is recognised and an action associated with it will be performed. Furthermore, the separation of context recognition and choice of action allows to adapt this association at run-time according to user preferences.

3 Modeling the door lock context in OWL DL

In contrast to other modeling approaches ([3, 4]) we concentrate on modeling the application domain, instead of devising a top-level ontology for context in general, as in [6]. We model the context concepts specifically useful for certain application tasks and reasoning about the situation the application is in.

The Door OWL DL ontology captures the main notions of our application domain in a descriptive way. The context concepts are described by other roles and concepts defined in the TBox. The door context is refined by different sub-concept hierarchies each modeling a certain aspect of the door context, e.g. the location of the resident or the person at the door. While the former is related to the door context via the role hasContextResident, the later is related via the role hasContextAgent. In Figure 2 this part of the door TBox is shown. The concepts AuthorisedPersonRingingContext and AuthorisedNeighbourRingingContext refine the aspect of the person ringing, while ResidentOutOfHomeContext specialises the door context according (the location of) the resident. Taking into account the concept definitions given in Example 1, the DL reasoner detects the implicit subsumption relation between AuthorisedPersonRinging-Context and AuthorisedNeighbourRingingContext during classification. The description of the current situation is obtained from the filled in context profile from the context service, converted to OWL and added to the ABox.

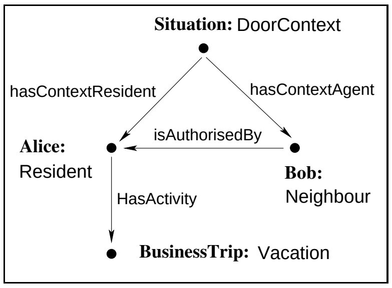

Figure 3. An ABox fragment.

In Figure 3 we see an ABox fragment describing a situation where the resident Alice is on vacation and the neighbour Bob is authorised by Alice and rings at the door. It is easy to see that Alice is an instance of the concept VacationResident defined in Example 1. Thus Bob is an instance of the concept AuthorisedNeighbour. In our ontology the concept VacationResident is a sub-concept of ResidentOutOf-HomeContext. Consequently, Alice is also an instance of ResidentOutOfHomeContext. The DL system derives these relationships and that Situation is an instance of ResidentOutOfHomeContext and AuthorisedNeighbour-RingingContext. Now, these concept names are returned to the context application, which triggers the action associated with this set of concepts. However, each ABox in our approach describes only one situation. The situation descriptions are not kept after realization, thus the size of the ABox does not vary much over time. The door ontology consists so far of 172 concepts, 28 roles and about 98 individuals in all ABoxes. Though this is a rather small ontology, it is interesting to see whether reasoning can be performed sufficiently fast in this setting. The door ontology uses the DL $\mathcal{SHIF}$. This DL lays between the fragment shown in Table 1 and OWL DL.

4 Performance of Realization

The complexity of ABox reasoning for the DL employed here is NExpTime complete in the worst case, see [10]. However, there exist some highly optimised DL reasoners for OWL DL that behave well in practice. Since our door lock application does not allow for computations times longer than a couple of seconds, it is interesting to get an impression of the performance of the reasoners for computing realization. The door ontology uses general concept inclusions (GCIs) which are known to make reasoning harder in practice. A natural question is whether one should disallow these from the ontology. To get a clearer picture, we

Table 2. ABox realization Run-times (in s).

	Door	no-GCIs	**1-Sit**	2-Sit	4-Sit
Racer	24.70	4.68	**1.04**	1.39	3.57
RacerPro	4.31	1.58	**0.28**	0.38	0.86
Pellet	27.75	7.70	**0.31**	0.59	1.79

ran an evaluation with two versions of the door ontology: **Door** (KB described in the last Section; contains 12 Situation descriptions.) and **no-GCIs** (Door KB without GCIs, i.e., disjointness and domain and range restrictions for roles are deleted.) Obviously the number of individuals is crucial for the performance of ABox realization. In our context application framework an ABox contains only a single situation description. Nevertheless to get an impression of the scalability we ran with 12 situations (in Door) and also tests with 2 and 4 situation descriptions in the ABox of the door ontology. We tested three DL reasoners that implement realization: the well-known system Racer (v. 1.7.24) [11], its commercial successor system RacerPro (v. 1.9) [12] and Pellet (v. 1.3 beta2) [13] a comparatively new OWL DL reasoner. The run-times shown in Table 2 were measured on a Pentium IV System, 2 GHz system. For a single situation and even for 2 or 4 situations the run-times for all systems are feasible for the door lock application. The increase of run-time w.r.t. the increase of situations modeled, indicates that keeping several situation description in the ABox will not result in drastic performance degradation.

For the ontology no-GCIs the deletion of GCIs speeds-up the realization of the individuals. Nevertheless, reduction of the language constructs comes at the cost of missing implicit subsumption or instance relations. For no-GCIs several of these relations can no longer be detected and not the most specific context can be recognised from the information available. So, it is not advisable for this application to degrade expressivity to obtain better run-times.

To sum up today's DL reasoners can compute realization for ABoxes with few situation descriptions in a run time acceptable for the door lock application.

5 Conclusions and Future work

We have proposed a framework for processing context information based on the modeling of contexts in OWL DL. We have realised the task for context recognition by employing the DL reasoning service realization and have shown the applicability of our approach by using it in our application scenario of an automatic door lock. Furthermore, our approach seems to be practicable as a first performance evaluation indicates.

Our application scenario appears to be simple, but it exemplifies the main difficulties of context-aware applications. However, it is of course necessary to refine the modeling and, moreover, to apply our approach also to other application domains and more complex scenarios. Besides this our work can be extended in many ways. On the practical side we have to investigate how the DL systems can be coupled to context applications. On the theoretical side the automatic generation of context profiles from the concepts in the TBox and to develop a method to match context profiles can with the data sources available to the context service are most prominent. Moreover, the trade-off between expressiveness and performance as well as the limits of modeling in OWL DL, especially in comparison to other modeling and reasoning approaches have to be investigated in detail.

References

[1] A. Schmidt, K. Aidoo, A. Takaluoma, U. Tuomela, K. V. Laerhoven, and W. V. de Velde, "Advanced interaction in context," in *Proc. of HUC'99*. LNCS, 1999.

[2] D. Salber, A. K. Dey, and G. D. Abowd, "The context toolkit: Aiding the development of context-enabled applications," in *Proc. of CHI'99*, 1999.

[3] K. Henricksen, S. Livingstone, and J. Indulska, "Towards a hybrid approach to context modelling, reasoning and interoperation," in *Proc. of UbiComp'4*, 2004, pp. 54–61.

[4] F. Fuchs, I. Hochstatter, M. Krause, and M. Berger, "A metamodel approach to context information," in *Proc. of Context Modeling and Reasoning Workshop (CoMoRea)*, 2005.

[5] T. Gu, X. Wang, H. Pung, and D. Zhang, "An ontology-based context model in intelligent environments," in *Proc. of CNDS'04*, 2004.

[6] H. Chen, T. Finin, and A. Joshi, "An ontology for context aware pervasive computing environments," in *Knowledge Engineering Review*. Cambridge Univ. Press, 2004.

[7] D. McGuinness and F. van Harmelen, "OWL Web Ontology Language Overview, W3C Recommendation," 2004. [Online]. Available: http://www.w3.org/TR/owl-features/

[8] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. Patel-Schneider, Eds., *The Description Logic Handbook*. Cambridge Univ. Press, 2003.

[9] D. Calvanese and G. D. Giacomo, "Expressive description logics," in *[8]*, pp. 178–219.

[10] S. Tobies, "The complexity of reasoning with cardinality restrictions and nominals in expressive description logics," *Journal of AI Research*, vol. 12, pp. 199–217, 2000.

[11] V. Haarslev and R. Möller, "RACER system description," in *Proc. of IJCAR '01*, ser. LNCS. Springer, 2001.

[12] V. Haarslev, R. Möller, and M. Wessel, "RacerPro User Guide," 2005, see http://www.racer-systems.com/products/racerpro/users-guide-1-9.pdf.

[13] E. Sirin and B. Parsia, "Pellet: An OWL DL reasoner," in *Proc. of the 2004 Description Logic Workshop (DL 2004)*, ser. CEUR-WS, no. 104, 2004.