OWL 2 Reasoning To Detect Energy-Efficient Software Variants From Context *

Sebastian Götz[1], Julian Mendez[2], Veronika Thost[2], and Anni-Yasmin Turhan[2]

[1]Software Technology Group
Inst. f. Software and Multimedia Technology
Technische Universität Dresden

[2]Chair for Automata Theory
Inst. f. Theoretical Computer Science
Technische Universität Dresden

Abstract. Runtime variability management of component-based software systems allows to consider the current context of a system for system configuration to achieve energy-efficiency. For optimizing the system configuration at runtime, the early recognition of situations apt to reconfiguration is an important task. To describe these situations on differing levels of detail and to allow their recognition even if only incomplete information is available, we employ the ontology language OWL 2 and the reasoning services defined for it. In this paper, we show that the relevant situations for optimizing the current system configuration can be modeled in the different OWL 2 profiles. We further provide a case study on the performance of state of the art OWL 2 reasoning systems for answering concept queries and conjunctive queries modeling the situations to be detected.

1 Introduction

A typical approach to introduce variability into complex software systems is the application of component-based software engineering, where variability is achieved by multiple implementations for components [21]. A *software component* is an abstract element of reuse, which declares what it requires and provides. Each component comprises several implementations providing the same functionality, but differing in the realization.

A feasible approach to component-based self-adaptive systems are *contract-based negotiations*, which reflect the non-functional behavior of component implementations [17,6,7]. A *contract* comprises several so-called *clauses*, which can be assumptions about the system, its context, or guarantees of the non-functional behavior provided by the respective implementation. For example, a contract for a video streaming component can specify that in case a high bandwidth is given (assumption), a high frame-rate (guarantee) is provided. *Self-adaptive software* [5] exploits this variability to determine an optimal variant for non-functional properties—as in our case energy consumption. As a prerequisite to adaptation, the system and its context have to be monitored and analyzed. That is, self-adaptive software needs to be context-aware [18].

* This work is supported in a part by the German Research Foundation (DFG) in the Collaborative Research Center 912 'Highly Adaptive Energy-Efficient Computing'.

1.1 Ontology-based Situation Recognition for Runtime Variability Management

Variability management for energy saving is carried out at runtime of the software system, to ensure that the current configuration of system components employed uses energy optimally. A key aspect is the ability to adjust the system configuration to the current system load and, more importantly, to the hardware resources available. In the end it is the hardware that consumes the energy directly when being utilized by the software. Depending on the current context (e.g., the available resources, the system load, the software variants available, etc.) a component-based software system can be adapted in different ways to save power at runtime.

Clearly, system adaptation needs to be carried out, if there are 'violations' of contract clauses of a component, which reflect the assumptions that led to the selection of this component. *Contract violations* occur if the requirements of the selected software variants cannot be provided anymore (e.g., if utilization thresholds are exceeded). In order to save energy, adaptations should be performed in situations where energy is (about to be) wasted. This kind of situations does not necessarily involve a violation, but is a type of situations where clauses that were not valid at the time of the last decision have now become valid.

Ideally, a software management system is notified in advance about situations where a reconfiguration might be beneficial to carry out appropriate adjustments in time. To this end, context information characterizing the execution environment has to be collected, and situations apt to optimization have to be recognized in this (abstract) representation of the actual system. In case a critical situation is recognized, the adjustments to be carried out have to be determined in a second step by the software management system. In this paper, we concentrate only on the task of situation recognition.

Typically, the information sources for the overall software system's status providing information relevant to situation recognition are very heterogeneous. For instance, the properties of different implementations of a particular functionality may be given by the software providers and could be stored in a database, while other information, such as the setup of the hardware and the current system parameters, could be retrieved from the operating system directly. These sources yield information of different levels of detail, which can be integrated in a Description Logic (DL) ontology. Moreover, logical reasoning as provided by DL systems, can handle incomplete information on the current situation gracefully, since these systems operate under the *open world assumption*.

For these reasons, we follow a common approach to situation recognition and use an ontology describing the managed software system and its hardware. We employ DL reasoning services to recognize situations of interest. This approach has been employed in several domains for context-aware applications, for instance, scene interpretation [15], the intelligent home domain [20] and air surveillance [1]. In [15,1,20] it was demonstrated that the expressivity and reasoning capabilities of DL systems suffice to model the domain at hand faithfully and to detect critical situations.

1.2 OWL 2 Reasoning for Situation Recognition

DLs are the logical under-pinning of the ontology language OWL 2 [26]. In order to apply DL reasoning for situation recognition, we model a modular video platform and the software variants for video-related services (e.g., up- or downloading services) in an OWL 2 ontology. In this paper we introduce the necessary notions only in an intuitive sense. For details on DLs we refer the reader to [3].

In OWL 2, categories from the application domain are described by *concept expressions* and binary relations by so-called *roles*. For example, the concept VideoServer, which is a server that hosts components to provide transcoding (i.e., the conversion of video encodings), up- and downloading services can be characterized by the expression:

$$\begin{aligned}\mathsf{VideoServer} \equiv \mathsf{Server} \sqcap (\exists \mathsf{hosts}.\mathsf{TranscoderComponent}) \sqcap \\ (\exists \mathsf{hosts}.\mathsf{UploaderComponent}) \sqcap (\exists \mathsf{hosts}.\mathsf{DownloaderComponent})\end{aligned}$$

Such definitions of concepts are stored in the *TBox*. In addition, the TBox can contain characteristics of roles (e.g., transitivity). Our TBox contains information on concepts common in the domain of modular video platforms (e.g., server or transcoder component) and on relations.

The *ABox* stores concrete facts about a particular application—in our case the servers available together with their utilization. These facts are expressed by *concept assertions*, which state that an individual belongs to a (possibly complex) concept or *role assertions* that relate two individuals via a role. For example, we can state in our ABox $\mathcal{A}_1$ that the individual *Server1* belongs to the class VideoServer and that its related transcoder component is individual *Variant1*, which belongs to the concept FastTranscoderImplementation, by writing:

$$\begin{aligned}\mathcal{A}_1 = \{\ &\mathsf{VideoServer}(\mathit{Server1}), \mathsf{hosts}(\mathit{Server1}, \mathit{Variant1}), \\ &\mathsf{FastTranscoderImplementation}(\mathit{Variant1})\ \}.\end{aligned}$$

A TBox and an ABox together constitute an *ontology*.

Now, such an ontology can be used to detect complex situations by applying the DL reasoning services: concept query answering and conjunctive query answering. Both are applied to retrieve (tuples of) specific individuals from an ontology. Given a concept and an ontology, *concept query answering* returns all individuals of the ABox that belong to that concept regarding the information captured in the ontology. *Conjunctive query answering* allows to retrieve tuples of ABox individuals that fulfill the conditions specified in the query. Concept and conjunctive queries also differ in the expressive power for specifying the situations. While the former are limited by the expressivity of the ontology language, the latter can make use of variables to describe arbitrary structures as aspects of situations. This addition in expressivity comes at the cost of higher computational complexity of reasoning.

The OWL 2 standard for ontology languages comprises so-called *OWL 2 profiles* which differ w.r.t. their expressivity [26]. Depending on the profile, different concept constructors and kinds of relations are allowed in the TBox:

- OWL 2 is the most expressive ontology language in the W3C standard. Concept query answering in the corresponding DL $\mathcal{SROIQ}$ is 2NExpTime-complete [9,10].
- OWL 2 EL corresponds to the DL $\mathcal{EL}^{++}$, where concept query answering is in P [2]. However, conjunctive query answering in the sublogic $\mathcal{EL}$ is already P-complete w.r.t. the size of the ABox alone.
- OWL 2 QL provides only very limited concept descriptions. For its corresponding DL DL-Lite$_\mathcal{R}$ query answering is in AC^0, (which is a proper subclass of the class of P), if measured w.r.t. the size of the ABox alone [4].

Over the past years, highly optimized reasoners for answering concept or conjunctive queries have been developed. Although the computational complexity of the implemented algorithms for OWL 2 EL and QL is promisingly low, it is not clear whether these implementations are yet fast enough to realize situation recognition for applications that deal with complex situations and require fast response times—such as component-based systems reconfiguring.

While [15,1] argued that the reasoning capabilities of DL systems can suffice to recognize complex situations, little is known about whether the performance of the implementation of DL reasoners is yet good enough for this kind of task. This question was addressed in the study [25] back in 2006 for concept queries, where it turned out that for rather small ontologies and applications that require moderate response times (of about 20 seconds), the performance of the DL reasoners was barely adequate. Since then, DL reasoners have evolved in terms of reasoning services offered and in terms of performance. This motivates our empiric study to measure the performance of today's reasoning systems for concept queries and conjunctive queries for the different OWL profiles. The application is to recognize complex situations relevant for energy efficiency in a component system that realizes a video platform such that it can be reconfigured at runtime.

In this paper, we demonstrate that the different OWL profiles can be applied for modeling a modular video platform and critical situations for system reconfiguration (in Section 2). Second, we present an empirical evaluation w.r.t. the different OWL profiles on how current DL reasoners perform on concept and conjunctive queries in our scenario (in Section 3).

2 Use Case: Managing a Modular Video Platform

To show the benefits of the ontology-based approach to variability management, we consider a modular video platform as application scenario. Among others, the platform provides software components for transcoding, up- and downloading of videos. These functionalities are complex and resource-intensive, which have to be provided in a required quality. In order to effectively exploit the available resources, we optimize the configuration of the system at runtime. We choose software variants that allow for optimal configuration of the hardware resources, to reduce the overall energy consumption of the video platform. Consider a server providing a transcoding service. If the server's load is currently decreasing, the CPU frequency can be reduced to save energy. To ensure, however, that the

system still provides the required quality, such a change might necessitate a reconfiguration (i.e., a switch to implementations that simply allow for a lower CPU frequency).

To recognize situations where reconfigurations can be beneficial, we create ontologies modeling information about the system, capture situations apt for reconfiguration as query concepts or as conjunctive queries and then apply DL reasoners to detect situations by invoking concept query answering or conjunctive query answering, respectively. More precisely, at design time, the general domain knowledge about video platforms (e.g., the kinds of services provided) and notions of components of the systems (e.g., dependencies on hardware resources or specific software variants) are modeled in the TBox. The situations to be recognized are modeled as concept or conjunctive queries—depending on the reasoning task to be employed. We assume the TBox and the set of queries to be fixed over the runtime of the system. The ABox describes the actual architecture of the specific application (i.e., the currently available hard- and software resources) and their current state (e.g., the load of the system, and the implementations executed). Most of this data is collected at runtime of the software system. Due to the highly dynamic nature of the system, the ABox is refreshed several times a minute storing information from several information sources. So-called *preprocessors* are applied for the task of converting numerical sensor data into primitive named concepts (following the approach in [1,22]). For example, if the load measured for a server $Server1$ has been very high, the assertions

$$\mathsf{OverUtilized}(\mathit{Load1}),\ \mathsf{hasLoadAverage}(\mathit{Server1}, \mathit{Load1})$$

are added to the ABox created for the past interval. Once the ABox is refreshed, we apply the DL reasoner to answer the concept or conjunctive queries.

2.1 Modeling the Component System

Now we describe how the overall video platform system is modeled in the ontology that is the base ontology for our tests and how the queries are captured in the different OWL 2 variants. Our TBox contains concepts describing the video platform domain (e.g., characteristics of a TranscodingService) and notions for component systems (e.g., FastTranscoderImplementation), written in the DL $\mathcal{ALCI}$, a proper sub-logic of OWL 2. For this DL, testing concept queries is PSpace-complete [23], while conjunctive query answering is even 2ExpTime-complete [12]. Our ABox contains assertions describing the architecture of the video platform and its current state.

Example 1. Assume the load of our *Server1* has changed to 'underutilized' in the past intervals. Now, the characterization of *Server1*, its resources, and states at runtime can be captured by:

$\mathsf{VideoServer}(\mathit{Server1}),$
$\mathsf{FastTranscoderImplementation}(\mathit{Variant1}),\ \mathsf{hosts}(\mathit{Server1}, \mathit{Variant1}),$
$\mathsf{LowCPUFrequency}(\mathit{CPU1}),\quad \mathsf{hasCPUFrequency}(\mathit{Server1}, \mathit{CPU1}),$
$\mathsf{UnderUtilized}(\mathit{Load2}),\quad \mathsf{hasLoadAverage}(\mathit{Server1}, \mathit{Load2}).$

It turned out that even the expressivity of the lightweight profiles OWL 2 EL and QL allow to describe the main characteristics of the domain knowledge of our application scenario. This is because the TBox primarily captures the hard- and software architecture of the system, which is exactly the conceptual modeling use case DL-Lite has been developed for. Further, complex concept definitions that cannot be represented in the lightweight profiles can be captured, alternatively, using fine-granular conjunctive queries for modeling the situations to be recognized. Consider the following concept definition for underutilized servers, which have a load average that is underutilized or almost underutilized:

$$\mathsf{UnderUtilizedServer} \equiv \exists \mathsf{hasLoadAverage}.(\mathsf{AlmostUnderUtilized} \sqcup \mathsf{UnderUtilized})$$

It cannot be expressed in an OWL 2 EL/QL ontology, due to the disjunction. However, it can be expressed with unions of conjunctive queries and thus be used in these situation descriptions.

2.2 Modeling the Situations Apt for Reconfiguration

To recognize critical situations, we apply either answering of concept or of (unions of) conjunctive queries. For the former, the situations need to be specified as concepts, while for the latter, as (unions of) conjunctive queries.

Example 2. Consider a situation apt for switching between transcoder implementations with two different specifications: one needs a medium CPU frequency and the other requires a high CPU frequency. Both implementations must be hosted on the same machine. The server has to have a high CPU frequency while its load average is low and anticipated to stay so. The resulting query concept is displayed in Figure 1 in the upper half as the concept $\mathsf{SwitchTranscoderSituationAnticipatory}$ and the corresponding conjunctive query $q_{\mathsf{SwitchTranscoderSituationAnticipatory}}$ in the lower half of the figure. In Figure 1, a situation that generalizes the above one is characterized in the query $q_{\mathsf{SwitchTranscoderSituation}}$. In this situation the average load trend cannot be forecasted; apart from that, the situations are the same. Clearly, this situation is refined by the first one. It is fruitful to model such refinements of situations in order to allow for graceful handling of incomplete information. Assume that, for a particular moment in time, it cannot be determined that the load average will show some constant development in the next intervals. Then, a corresponding assertion is not generated, and the next ABox is incomplete. In such a case, a situation indicating that a change of a transcoder implementation is beneficial cannot be recognized. More precisely, the concept $\mathsf{SwitchTranscoderSituationAnticipatory}$ does not have an instance in the current ABox and the query $q_{\mathsf{SwitchTranscoderSituationAnticipatory}}$ yields no tuples. However, a more general concept $\mathsf{SwitchTranscoderSituation}$ would have an instance and the query $q_{\mathsf{SwitchTranscoderSituation}}$ would yield a result tuple. Thus, this kind of situation could be recognized, and it could be decided (e.g., by using other context information) whether a reconfiguration should take place.

$$\begin{aligned}
&\mathsf{SwitchTranscoderAnticipatorySituation} \equiv \\
&\quad \mathsf{TranscoderComponent} \sqcap \exists \mathsf{requires.MedCPUFreq} \sqcap \\
&\quad \exists \mathsf{isHostedBy}.(\mathsf{UnderUtilizedServer} \sqcap \\
&\qquad\qquad \exists \mathsf{hosts}.(\mathsf{TranscoderComponent} \sqcap \exists \mathsf{requires.HighCPUFreq}) \sqcap \\
&\qquad\qquad \exists \mathsf{hasLoadAvgTrend.DecreasingLAT} \sqcap \exists \mathsf{hasCPUFrequency.HighCPUFreq})
\end{aligned}$$

$$\begin{aligned}
&q_{\mathsf{SwitchTranscoderAnticipatorySituation}} = \\
&\quad \exists x, y.\mathsf{TranscoderComponent}(x) \wedge \mathsf{requires}(x, z_1) \wedge \mathsf{MedCPUFreq}(z_1) \wedge \\
&\qquad \mathsf{isHostedBy}(x, z_2) \wedge \mathsf{UnderUtilizedServer}(z_2) \wedge \mathsf{hosts}(z_2, y) \wedge \\
&\qquad \mathsf{TranscoderComponent}(y) \wedge \mathsf{requires}(y, z_3) \wedge \mathsf{HighCPUFreq}(z_3) \wedge \\
&\qquad \mathsf{hasCPUFrequency}(z_2, z_4) \wedge \mathsf{HighCPUFreq}(z_4) \wedge \\
&\qquad \mathsf{hasLoadAvgTrend}(z_2, z_5) \wedge \mathsf{DecreasingLAT}(z_5)
\end{aligned}$$

$$\begin{aligned}
&q_{\mathsf{SwitchTranscoderSituation}} = \\
&\quad \exists x, y.\mathsf{TranscoderComponent}(x) \wedge \mathsf{requires}(x, z_1) \wedge \mathsf{MedCPUFreq}(z_1) \wedge \\
&\qquad \mathsf{isHostedBy}(x, z_2) \wedge \mathsf{UnderUtilizedServer}(z_2) \wedge \mathsf{hosts}(z_2, y) \wedge \\
&\qquad \mathsf{TranscoderComponent}(y) \wedge \mathsf{requires}(y, z_3) \wedge \mathsf{HighCPUFreq}(z_3) \wedge \\
&\qquad \mathsf{hasCPUFrequency}(z_2, z_4) \wedge \mathsf{HighCPUFreq}(z_4)
\end{aligned}$$

Fig. 1. Situations when to switch a transcoder implementation captured as query concept and conjunctive queries.

Although our modeling is in large parts rather abstract, it captures the nature of component systems. We conjecture that the ontology grows primarily in size rather than in complexity, in case real applications are considered. Also, the modeling should be similar for component systems in other domains, since the focus is generally on modeling the components with their contracts and the topological structure of the system's hardware environment.

3 Evaluation of Reasoner Performance w.r.t. the OWL 2 Variants

The goal of our evaluation is to see whether current OWL 2 reasoners are yet appropriate for situation recognition to initiate system reconfiguration for self-adaptive software. However, to adopt DL reasoning for this kind of scenario, the reasoners have to be able to detect situations by processing realistic amounts of data within short time. We consider OWL 2 and the two profiles OWL 2 EL and OWL 2 QL in our evaluation. Since the syntactic restrictions of the lightweight profiles allow only for coarser modeling than full OWL 2, some information cannot be modeled. An interesting question is whether this leads to missing inferences in our scenario.

3.1 Test Data and Reasoning Systems

Test TBoxes. Our base TBox from Section 2.1 contains about 200 concepts and 40 roles and uses $\mathcal{ALCI}$, a sub-logic of OWL 2. For both lightweight profiles, we built variations of the base TBox manually—keeping as much information as possible.

Reasoner	Version	Query type		Profile		
		Concept	Conjunctive	OWL	EL	QL
FaCT++ [24]	v1.6.1	x		x	x	x
HermiT [14]	v1.3.6	x		$\mathcal{SHOIQ}$	x	x
Pellet [19]	v2.3.0	x	x	$\mathcal{SHOIN}(\mathcal{D})$	x	x
RacerPro [8]	v2.0	x	x	$\mathcal{SHIQ}(\mathcal{D})$	x	x
ELK [11]	v0.3.1	x			$\mathcal{EL}^+$	
jCel [13]	v0.18.0	x			$\mathcal{EL}^+$	
Quest [16]	v1.7-alpha		x			x

Table 1. Reasoners and their supported queries and profiles.

Test ABoxes. The ABoxes model a modular video platform running on four servers and providing the services described in Section 2. Since the concept assertions use only named concepts, the ABoxes do not vary for the profiles.

We consider two different ABoxes modeling two different states of the system. In order to reflect realistic scenarios, the test ABoxes do not only contain information about the situations to be detected, but model the overall system state. For example, we modeled other users requesting video services. This roughly doubles the sizes of both ABoxes. Each of the test ABoxes contains about 450 individuals, 485 concept, and 350 role assertions.

Test Queries. We modeled 9 critical situations to be recognized as OWL 2 concepts. For OWL 2 EL, these concept definitions had to be only slightly adapted. In particular, we removed the disjunction by replacing it with adequate new concepts and corresponding inheritance relations.

Since the OWL 2 QL profile does not support truly complex concept descriptions, the situations cannot be modeled as concepts; but they can be captured with conjunctive queries. Therefore, we only evaluate conjunctive query answering for the QL profile.

Note that for modeling the modular video platform and situations for reconfiguration, we do not need universal quantification. For the 9 situations, we formulated corresponding conjunctive queries for our test suite. The concept queries have a size of about 10–counting the concept and role names. The conjunctive queries are formulated in the query languages SPARQL and nrql. They contain 15 conjuncts on average.[1]

Reasoning Systems. The tests were run for seven DL reasoners, which differ w.r.t. the DL they support and the reasoning services they provide. Table 1 depicts the tested reasoners, the used version, and the closest DL of the respective profile they implement ('x' stands for full coverage).

Next to the tableaux-based reasoners for expressive DLs in the first group of Table 1, we tested reasoners specialized on lightweight profiles, which are listed in the second and third group of the table. Quest can be used for ontology-based

[1] Our complete test data is available from
`http://lat.inf.tu-dresden.de/~thost/download/exp_data_owled13.zip`

		Concept queries				Conjunctive queries			
		Load.	Reason.	Avg/Q.	Total	Load.	Reason.	Avg/Q.	Total
OWL	FaCT++	0.164	0.223	0.019	0.387				
	HermiT	0.200	0.156	0.008	0.356				
	Pellet	0.162	0.897	0.086	1.059				
	RacerPro	0.198	1.798	0.145	1.996	0.539	1.441	0.144	1.980
EL	ELK	0.173	0.078	0.004	0.251				
	FaCT++	0.175	0.060	0.003	0.235				
	HermiT	0.222	0.115	0.005	0.338				
	jCel	0.177	0.271	0.021	0.448				
	Pellet	0.204	0.585	0.054	0.789	0.207	0.776	0.078	0.983
	RacerPro	0.163	1.999	0.165	2.162	0.536	0.971	0.097	1.507
QL	Pellet					0.199	1.355	0.136	1.554
	Quest					0.723	5.401	0.540	6.124
	RacerPro					0.429	0.748	0.075	1.177

Table 2. Runtimes for answering concept queries and conjunctive queries in seconds.

data access (i.e., a data base functions as ABox and can be queried directly). However, we used Quest with classical ABoxes, here.

3.2 Evaluation

The tests were carried out on an Intel Core 2 Duo workstation with 2 GB RAM using Java 1.6.0. on Ubuntu. Besides recording the mere runtimes, we checked whether the reasoners delivered the same results for a query. For our concept queries, all reasoners agreed on the result individuals. For the conjunctive queries, we noticed that Pellet did not compute the expected answer sets. On the one hand, it did not deliver all answer tuples for 8 queries while reasoning in OWL 2 QL. On the other, it returned too many result tuples for one query in OWL 2 EL. The other reasoners delivered the expected answers. This nicely demonstrates that the lightweight profiles suffice for modeling and reasoning about our modular video platform—and thus about the hard- and software of complex systems, in general.

Performance for Concept Queries. For concept queries, we ran tests for full OWL 2 and the OWL 2 EL profile. We used the OWL API (version 3.4.1) to access the reasoners. The results are displayed in the left half of Table 2 sorted by profiles. The first column depicts the time spent on loading the ontology. The next one displays the time for answering *all* the concept queries. The average runtime per query is displayed next. The last column contains the runtime for the overall process and thus is the most interesting for our application of situation recognition. As expected, it shows that the overall runtime is higher for OWL 2 than for the OWL 2 EL profile, especially for FaCT++ and Pellet. Nevertheless, the gain when using OWL 2 EL is not big. Interestingly, RacerPro performs slightly better for OWL.

OWL 2: Apart from RacerPro, which took about 2 seconds, all reasoners delivered a full situation recognition within roughly 1 second.

OWL 2 EL: With an overall runtime of about 0.25 seconds, ELK and FaCT++ outperform the other systems. For these systems the overall runtime is dominated by the time for loading the ontology. HermiT and jCel can still test for occurrence of a critical situation within half a second. All systems can perform situation recognition within about 2 seconds.

Performance for Conjunctive Queries. For the conjunctive queries, the results for all of the three profiles are displayed in right half of Table 2. As for answering concept queries, reasoning in the lightweight profiles is a little faster.

OWL 2: Here, RacerPro needs about 2 seconds overall runtime. It thus takes about the same time as for the concept queries. Note that its reasoning time for the conjunctive queries is lower than for concept queries, but the loading of the same ontology takes more than twice using the RacerPro's interface for conjunctive querying.

OWL 2 EL: Pellet and RacerPro answer all queries in less than 2 seconds. However, Pellet didn't deliver the correct result, whereas RacerPro did.

OWL 2 QL: RacerPro performs even slightly better for this profile. While Pellet performs worse than in the OWL 2 EL case giving an incomplete answer. By taking 6 seconds, Quest shows a significantly worse performance. We conjecture that this is attributed to running a first alpha version of Quest and using a traditional ABox instead of a database, for which it has been designed for.

All in all, the experiments show that most state of the art reasoners can be applied for situation recognition in our component-based video platform, since response times of 1 to 2 seconds are acceptable if one considers the long time video services such as transcoding usually run. Especially by the use of the lightweight profiles, we achieve very good runtimes for reasoning. Our study revealed two surprising effects. First, comparing the runtimes for concept query answering and conjunctive query answering, it showed that the run times do not differ largely, which wasn't to be expected. Second, the loss of information when using a lightweight profile turned out to be only marginal for our video platform use case, and, more importantly, it did not influence the completeness of the results—apart those of Pellet.

4 Conclusions and Future work

We have supplied a study on employing OWL 2 and DL reasoners to perform situation recognition for runtime variability management applied to a video platform. To the end of automatically recognizing complex situations that might

invoke the switching of software variants in order to achieve energy efficiency, the domain was modeled in an OWL 2 ontology. In this ontology the ABox reflected realistic situations in the application. The actual recognition of critical situations, was realized by concept query answering or conjunctive query answering. Our experiments w.r.t. the different OWL 2 profiles gave evidence that the performance of today's DL systems is sufficient to detect complex situations fast enough. In particular, for the OWL 2 EL and the OWL 2 QL profile it can be done within 2 seconds for each of the inferences employed. Thus, there still seems to be room for growth—considering larger ontologies in practice.

Future work on the practical side should evaluate whether the gain of reasoning time given by the lightweight profiles grows, too, with larger ontologies. Clearly, it would be interesting to run Quest in the ODBA mode and to realize the whole situation recognition more tightly coupled to a DB, such that the data collected there can be queried directly, instead of generating and loading an ABox. Further, the actual energy gain should be considered. On the theoretical side, we would like to lift the limitation of OWL regarding the modeling of fuzzy or even temporal information by investigating query answering for sequences of ABoxes, which contain this kind of information.

References

1. F. Baader, A. Bauer, P. Baumgartner, A. Cregan, A. Gabaldon, K. Ji, K. Lee, D. Rajaratnam, and R. Schwitter. A novel architecture for situation awareness systems. In M. Giese and A. Waaler, editors, *Proc. of the Int. Conf. on Automated Reasoning with Analytic Tableaux and Related Methods (TABLEAUX 2009)*, vol. 5607 of *LNCS*, p. 77–92. Springer, 2009.
2. F. Baader, S. Brandt, and C. Lutz. Pushing the $\mathcal{EL}$ envelope further. In K. Clark and P. F. Patel-Schneider, editors, *In Proc. of the OWLED Workshop*, 2008.
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. D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. Tractable reasoning and efficient query answering in description logics: The DL-Lite family. *Journal of Automated Reasoning*, 39(3):385–429, 2007.
5. B. H. Cheng, R. Lemos, H. Giese, P. Inverardi, J. Magee, J. Andersson, B. Becker, N. Bencomo, Y. Brun, B. Cukic, G. Marzo Serugendo, S. Dustdar, A. Finkelstein, C. Gacek, K. Geihs, V. Grassi, G. Karsai, H. M. Kienle, J. Kramer, M. Litoiu, S. Malek, R. Mirandola, H. A. Möller, S. Park, M. Shaw, M. Tichy, M. Tivoli, D. Weyns, and J. Whittle. Software engineering for self-adaptive systems: A research roadmap. In *Software Engineering for Self-Adaptive Systems*, vol. 5525 of *LNCS*, p. 1–26. Springer, 2009.
6. P. Collet. Taming complexity of large software systems: Contracting, self-adaptation and feature modeling. Habilitation, December 2011.
7. S. Götz, C. Wilke, S. Cech, and U. Aßmann. *Sustainable ICTs and Management Systems for Green Computing*, chapter Architecture and Mechanisms for Energy Auto Tuning, p. 45–73. IGI Global, 2012.
8. V. Haarslev, K. Hidde, R. Möller, and M. Wessel. The racerpro knowledge representation and reasoning system. *Semantic Web Journal*, 3(3):267–277, 2012.

9. I. Horrocks, O. Kutz, and U. Sattler. The even more irresistible $\mathcal{SROIQ}$. In *Proc. of the 10th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR-06)*, p. 57–67. AAAI Press, 2006.
10. Y. Kazakov. $\mathcal{RIQ}$ and $\mathcal{SROIQ}$ are harder than $\mathcal{SHOIQ}$. In G. Brewka and J. Lang, editors, *Proc. of the 11th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR-08)*, p. 274–284. AAAI Press, 2008.
11. Y. Kazakov, M. Krötzsch, and F. Simančík. ELK reasoner: Architecture and evaluation. In *Proc. of the OWL Reasoner Evaluation Workshop (ORE'12)*, vol. 858 of *CEUR Workshop*, 2012.
12. C. Lutz. The complexity of conjunctive query answering in expressive description logics. In A. Armando, P. Baumgartner, and G. Dowek, editors, *Proc. of the 4th International Joint Conference on Automated Reasoning (IJCAR2008)*, number 5195 in LNAI, p. 179–193. Springer, 2008.
13. J. Mendez. jCel: A modular rule-based reasoner. In *In Proc. of the 1st Int. Workshop on OWL Reasoner Evaluation (ORE'12)*, vol. 858 of *CEUR Workshop*, 2012.
14. B. Motik, R. Shearer, and I. Horrocks. Optimized Reasoning in Description Logics using Hypertableaux. In F. Pfennig, editor, *Proc. of the 23th Conf. on Automated Deduction (CADE-23)*, LNAI, p. 67–83, 2007. Springer.
15. B. Neumann and R. Möller. On scene interpretation with description logics. In H. Christensen and H.-H. Nagel, editors, *Cognitive Vision Systems: Samping the Spectrum of Approaches*, number 3948 in LNCS, p. 247–278. Springer, 2006.
16. M. Rodriguez-Muro and D. Calvanese. Quest, an OWL 2 QL reasoner for ontology-based data access. In *Proc. of the 9th Int. Workshop on OWL: Experiences and Directions (OWLED 2012)*, vol. 849 of *CEUR Workshop*, 2012.
17. S. Röttger and S. Zschaler. CQML+: Enhancements to CQML. In *In Proc. of the 1st International Workshop on Quality of Service in Component-Based Software Engineering*, p. 43–56. Cépadués-Éditions, 2003.
18. M. Salehie and L. Tahvildari. Self-adaptive software: Landscape and research challenges. *ACM Trans. Auton. Adapt. Syst.*, 4:14:1–14:42, 2009.
19. E. Sirin and B. Parsia. Pellet system description. In B. Parsia, U. Sattler, and D. Toman, editors, *Description Logics*, vol. 189 of *CEUR Workshop*, 2006.
20. T. Springer and A.-Y. Turhan. Employing description logics in ambient intelligence for modeling and reasoning about complex situations. *Journal of Ambient Intelligence and Smart Environments*, 1(3):235–259, 2009.
21. C. Szyperski. *Component Software: Beyond Object-Oriented Programming (ACM Press)*. Addison-Wesley Professional, 1997.
22. K. Taylor and L. Leidinger. Ontology-driven complex event processing in heterogeneous sensor networks. In *Proc. of 8th Extended Semantic Web Conference (ESWC 2011)*, vol. 6644 of *LNCS*, p. 285–299. Springer, 2011.
23. S. Tobies. *Complexity Results and Practical Algorithms for Logics in Knowledge Representation*. PhD thesis, RWTH Aachen, 2001.
24. D. Tsarkov, I. Horrocks, and P. F. Patel-Schneider. Optimising terminological reasoning for expressive description logics. *Journal of Automated Reasoning*, 2007.
25. 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, *Proc. of the Workshop on Context Modeling and Reasoning (CoMoRea'06)*. IEEE Computer Society, March 2006.
26. W3C OWL Working Group. OWL 2 web ontology language document overview. W3C Recommendation, 27th October 2009. `http://www.w3.org/TR/2009/REC-owl2-overview-20091027/`.