Situation Recognition for Service Management Systems Using OWL 2 Reasoners

Waltenegus Dargie[†], Eldora[*], Julian Mendez[*], Christoph Möbius[†],
Kateryna Rybina[†], Veronika Thost[*], Anni-Yasmin Turhan[*]

[*], Chair for Automata Theory
Institute for Theoretical Computer Science
Technische Universität Dresden

email: *lastname*@tcs.inf.tu-dresden.de

[†], Chair of Computer Networks,
Institute of Systems Architecture
Technische Universität Dresden

email: *firstname*. *lastname*@tu-dresden.de

***Abstract*—For service management systems the early recognition of situations that necessitate a rebinding or a migration of services 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 provide a case study on the performance of state of the art OWL 2 reasoning systems for answering class queries and conjunctive queries modeling the relevant situations for service rebinding or migration in the differing OWL 2 profiles.**

I. Introduction

Service management systems (SMS) are systems that, broadly speaking, orchestrate the execution of complex services in distributed (cluster) computing environments. The prime goal of these systems is to ensure that computing resources are efficiently utilized while functional and non-functional requirements of individual services, expressed in so-called *service level agreements* (SLAs), are respected. One of the resources managed by an SMS is power.

Unfortunately, a significant portion of the power consumption of Internet-based servers is wasted due to underutilization [6]—often Internet-based servers are utilized only between 30 and 70% of their full capacity even though their idle power consumption amounts up to 60% of their peak power consumption [2], [1]. An SMS can be employed in a cluster environment to ensure that power is frugally consumed by servers. In this regard, a key aspect of the SMS is its ability to adapt the use of hardware resources according to the present and anticipated workload. Depending on the type of services that are being hosted and the priority and magnitude of the processed workload, an SMS can carry out different forms of adaptations to save power.

A. Service Management

One of the essential adaptation strategies is to switch off underutilized machines as often as possible. This can be achieved by consolidating services running on different machines onto a selected number of machines, which are then optimally configured. Service consolidation, in turn, can be achieved by either carrying out runtime service migration [10] or service rebinding [11]. During *service migration*, the main memory content of a service is transferred from one physical machine to another at runtime while the service is still executing. In virtualized environments, this can be achieved by encapsulating the service inside a virtual machine (VM) and then migrating the VM itself. If the service is stateless, an SMS may prefer the adaptation technique of *service-rebinding*. In this case, another instance (*service B*) of a service being run on an underutilized server (*service A*) will be started elsewhere, and all future requests directed to *service A* will be redirected to *service B*. The aim is to gradually terminate *service A* and switch off the server on which *service A* was running.

Clearly, adaptation techniques can be carried out if there are 'symptoms' indicating a need for adaptation. These symptoms may refer to potential SLA violations or to some utilization thresholds that are crossed or, to put it more generally, to some critical situations that can be sensed in the system. The time between recognizing a critical situation and finishing all necessary adaptation tasks can be considerably long—for example, the migration of a VM can take several seconds, or even minutes, depending on its size and the available network bandwidth [23]. During this delay, the performance of the service may degrade and the power consumption of both the target and the source server may increase.

Ideally, the SMS should be notified in advance about situations where a server is overloaded or underutilized and carry out the appropriate adaptations. To this end, it is desirable to describe contextual information that sufficiently characterizes the execution environment (i.e., context pertaining to SLAs, workloads, services, servers, etc.), then recognize situations in the actual system that potentially lead to the violation of one of the predefined thresholds, and decide on the suitable adaptation strategy to alleviate this violation.

In order to do so the situations need to be represented on differing levels of detail—a task Description Logic (DL) ontologies [5] are designed for. Moreover, the various aspects of the overall managed system's status are supplied by different information sources. For instance, the properties of the different implementations of services may be given

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

978-1-4673-5077-8/13/$31.00 ©2013 IEEE

by the software providers and can be stored in a database, while other information, such as the layout of the hardware and the current system parameters, can be retrieved from the OS directly. These information sources yield information of different levels of detail. Data integration can be performed by using DLs, see [9], [7]. Most importantly, DL systems can handle incomplete information gracefully, since they operate under the *open world assumption*, i.e., missing information is neither regarded as being *true* or *false*. In contrast, missing information is regarded as being *false*, in database systems.

We follow a common approach to ontology-based situation recognition: we use an ontology describing the managed system and servers and employ DL reasoning to identify situations of interest. This approach has been employed in several domains for context-aware applications, for instance, in the intelligent home domain [27], [22] and air surveillance [3]. In [19], [3] it was demonstrated that the expressivity and reasoning capabilities of DL systems suffice to model the domain at hand faithfully.

B. OWL 2 for Situation Recognition

DLs are the logical formalism underlying the W3C standard OWL. In OWL categories from the application domain can be described by *class expressions* and binary relations by so-called *(object) properties*. For example, the class Server, which is hardware that has a CPU and a memory can be characterized by the expression:[1]

$$\mathsf{Server} \equiv \mathsf{Hardware} \sqcap (\exists \mathsf{hasPart}.\mathsf{CPU}) \sqcap (\exists \mathsf{hasPart}.\mathsf{Memory}).$$

The above definition assigns to the named class Server the complex class expression on the right-hand side and uses the property hasPart and the other named classes Hardware, CPU and Memory. Now, based on Server we can define an IdleServer as a server that has the power state 'idle' by writing:

$$\mathsf{IdleServer} \equiv \mathsf{Server} \sqcap \exists \mathsf{hasPowerState}.\mathsf{Idle}$$

Such definitions of classes are stored in the *TBox*. In addition, characteristics of properties can be stated in the TBox, e.g., that the property hasPart is transitive or that the property isPartOf is its inverse.[2]

The *ABox* stores concrete facts from the application, expressed by *class assertions*, which state that an individual belongs to a (possibly complex) class or *property assertions* that relate two individuals via a property.

Example 1: We can state in our ABox $\mathcal{A}_1$ that the individual named $Server1$ belongs to the class Server and that its related power state is individual $State2$, which belongs to the class IdleState, by writing the statements:

$$\mathcal{A}_1 = \{\ \mathsf{Server}(Server1), \mathsf{hasPowerState}(Server1, State2), \mathsf{IdleState}(State2)\ \}.$$

The TBox and the ABox together constitute the *ontology*. For DL systems, there are several *reasoning services* that can infer from the information given explicitly in the ontology the implicitly captured facts. *Subsumption* can compute super- and sub-class relationships for the classes defined in the TBox. For example, it can be derived that the class IdleServer is a subclass of the class $\exists$hasPart.CPU. *Class queries* compute for a given (complex) class $\mathsf{C_q}$ and an ontology all the individuals in the ABox that belong to the class $\mathsf{C_q}$. For the query class IdleServer and $\mathcal{A}_1$ we can derive the individual $Server1$. A more powerful way to query the ABox are conjunctive queries. A *conjunctive query* is a conjunction of assertions that may also contain variables, of which some can be existentially quantified. For example, the conjunctive query

$$q_{ex} = \exists x, y.\mathsf{Server}(x) \wedge \mathsf{hasPart}(x, z) \wedge \mathsf{uses}(y, z) \wedge \mathsf{Process}(y)$$

asks for all pairs of servers and processes, where the process uses some part of the server. In contrast to class queries, conjunctive queries can return a tuple of individuals from the ABox.

We model the basic categories and relations of the SMS domain, such as the hardware or the managed services, in a TBox. The current state of the system managed by the SMS is then captured at runtime in an ABox, similarly to [27], [22], [3]. To recognize the relevant situations for the SMS we employ answering of class or conjunctive queries w.r.t. the ABox. Once such a situation is detected for a (tuple of) ABox individual(s), the SMS invokes the appropriate adaptations on the returned individuals to ensure energy efficiency for the overall system.

The OWL 2 standard for ontology languages comprises so-called *OWL 2 profiles* which differ w.r.t. expressivity [28]. Depending on the profile, more class constructors and property statements are allowed for the TBox.

- OWL 2 is the most expressive ontology language in the W3C standard. Reasoning in the corresponding DL $\mathcal{SROIQ}$ is 2NExpTime-complete [13], [14], i.e., class queries can take more than double exponential time.
- OWL 2 EL corresponds to the DL $\mathcal{EL}^{++}$, where class query answering is in P [4], i.e., it can always be done in polynomial time. 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 allows only for very limited class 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 [8].

The motivation for the different profiles are the good computational properties of the lightweight DLs $\mathcal{EL}^{++}$ and *DL-Lite*$_\mathcal{R}$ for answering class queries or conjunctive queries, respectively.

There are non-commercial, optimized reasoners for answering class or conjunctive queries. Although the computational complexity of the implemented algorithms 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

[1] We give the class expressions in DL syntax for better readability.

[2] For the exact syntax and semantics of DLs we refer the reader to [5].

SMSs. While [19], [3] argued that the reasoning capabilities of DL systems suffice to recognize complex situations, little is known about whether the performance of the implementation of DL reasoners is good enough for this kind of task. This question was addressed in the study in [27] back in 2006 for class queries, where it turned out that for fairly 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 that measures the performance of today's reasoning systems for class queries and conjunctive queries for the different OWL profiles. The application is to recognize situations for an SMS that manages a video platform such that it runs in an energy efficient way.

The paper is structured as follows: Section II describes the ontology for the video platform use case and the modeling of the relevant situations. Section III presents the empirical evaluation, i.e., how current DL reasoners perform on class and conjunctive queries w.r.t. the different OWL profiles.

II. Use Case: Managing a Video Platform Server

For a proof of concept for our DL-based approach for SMS, we consider a video platform as application scenario, i.e., a distributed application over several servers, which allows users to search for, upload, and download videos. Internally, services for ranking and transcoding of videos (i.e., conversion of video encodings) are executed. The up- and downloading of videos are complex and resource-intensive processes. For that reason, we apply an elaborate service management to effectively exploit the available resources.

The two techniques considered to reduce the energy consumption of the video server platform are service migration and service rebinding. Service rebinding is performed in case one server is not optimally utilized while another server still has available resources. Consider a server providing a downloading service. If several users request this service at the same time, the server becomes overloaded. To balance the load, this downloading service can be rebound—by starting an instance of this service on another server that has available resources and by 'redirecting' future requests to the new instance.

To recognize situations where the application of such adaptation techniques can be beneficial, we create ontologies capturing information about the system and then apply DL reasoners to detect situations apt for optimization. More precisely, at design time the general domain knowledge about video platforms (e.g., the kinds of services provided) and notions of SMS (e.g., when a server has available resources) are described in the TBox. The relevant situations to be recognized are modeled as query classes or conjunctive queries—depending on the reasoning task to be employed. The TBox and the queries are assumed to be fixed over the runtime of the SMS.

The ABox describes the architecture of the specific application managed by the SMS (e.g., available servers) and its current state (e.g., load of the servers, executed implementations, etc.). Most of the data in the ABox has to be collected at runtime. Due to the highly dynamic nature of the system, the ABox is refreshed several times a minute. Every ABox can be generated by several sources, such as sensor data delivered by the OS or a database describing the implementations available to the SMS. For the task of converting numerical data (e.g., sensor data) *preprocessors* are applied to convert the numeric data into named classes (following the approach used in [3], [24]). For example, if the load measured for a server $Server1$ has been constantly very low, the assertions

$$\mathsf{hasLoadAverage}(Server1, Load2),\ \mathsf{UnderUtilized}(Load2)$$

are added to the ABox created for the past interval. Once the ABox is refreshed, the DL reasoner performs the answering of the class or conjunctive queries provided at design time.

A. Modeling the OWL 2 Video Platform Ontology

Our TBox contains basic notions of the video platform domain, such as characteristics of a DownloadingService and notions specific for SMS (e.g., AvailableResourceServer), written in the DL $\mathcal{ALCIQ}$, a proper sub-logic of OWL 2. For this DL, testing class queries is PSpace-complete [25], while conjunctive query answering is even 2ExpTime-complete [16]. Our ABox contains assertions describing the architecture of the video platform and its current state based on the available sensor data.

Example 2: Let's assume that $State1$ from ABox $\mathcal{A}_1$ has changed to 'operating' in the last interval. Now, the characterization of $Server1$, its resources, and states at runtime can be captured by:

$$\begin{array}{ll} \mathsf{Server}(Server1), & \\ \mathsf{CPU}(CPU1), & \mathsf{hasPart}(Server1, CPU1), \\ \mathsf{Memory}(Memory1), & \mathsf{hasPart}(Server1, Memory1), \\ \mathsf{Operating}(State1), & \mathsf{hasPowerState}(Server1, State1), \\ \mathsf{UnderUtilized}(Load1), & \mathsf{hasLoadAverage}(Server1, Load1) \end{array}$$

It turned out that even the expressivity of the lightweight profiles allows to describe at least the main characteristics of the domain knowledge of our application scenario. This is because the TBox primarily captures the conceptual model of the application, which is exactly the use case *DL-Lite* has been developed for. If needed, complex class definitions, which cannot be represented in the lightweight profiles, can be captured alternatively using fine-granular conjunctive queries for modeling the rebinding situations.

Example 3: Consider the class definition for underutilized servers, which have an average load that is underutilized or that have a part that is an underutilized CPU or NIC:

$$\begin{aligned} \mathsf{UnderUtilizedServer} \equiv\ & \exists \mathsf{hasLoadAverage}.\mathsf{UnderUtilized} \sqcup \\ & \exists \mathsf{hasPart}.(\mathsf{UnderUtilizedCPU} \sqcup \mathsf{UnderUtilizedNIC}) \end{aligned}$$

It cannot be expressed in an OWL 2 EL/QL ontology, due to disjunction ($\sqcup$). Thus, such a query concept would have to consist of the right-hand side of the definition.

B. Modeling the Rebinding Situations

To recognize critical situations, we apply either answering of class or of conjunctive queries. For the former, the situations

$$\begin{aligned}
&\mathsf{RebindingDownloadingServiceSituation} = \\
&\quad \exists \mathsf{hasServer}.(\mathsf{AvailableResourceServer} \sqcap \exists \mathsf{runs}.\exists \mathsf{hosts}.\mathsf{DownloadingImplementation}) \sqcap \\
&\quad \exists \mathsf{hasServer}.(\neg \mathsf{OptimallyUtilizedServer} \sqcap \exists \mathsf{runs}.\exists \mathsf{hosts}.\exists \mathsf{bindsTo}.\mathsf{DownloadingService}) \\
&\mathsf{RebindingServiceSituation} = \\
&\quad \exists \mathsf{hasServer}.(\mathsf{AvailableResourceServer} \sqcap \exists \mathsf{runs}.\exists \mathsf{hosts}.\mathsf{Implementation}) \sqcap \\
&\quad \exists \mathsf{hasServer}.(\neg \mathsf{OptimallyUtilizedServer} \sqcap \exists \mathsf{runs}.\exists \mathsf{hosts}.\exists \mathsf{bindsTo}.\mathsf{Service})
\end{aligned}$$

$$\begin{aligned}
&q_{\mathsf{RebindingDownloadingServiceSituation}} = \\
&\quad \exists x, y.\ \mathsf{AvailableResourceServer}(x) \wedge \mathsf{runs}(x, z_1) \wedge \mathsf{hosts}(z_1, z_2) \wedge \mathsf{DownloadingImplementation}(z_2) \wedge \\
&\qquad \mathsf{bindsTo}(z_2, z_3) \wedge \mathsf{DownloadingService}(z_3) \wedge \\
&\qquad \neg \mathsf{OptimallyUtilizedServer}(y) \wedge \mathsf{runs}(y, z_4) \wedge \mathsf{hosts}(z_4, z_5) \wedge \mathsf{DownloadingImplementation}(z_5) \wedge \\
&\qquad \mathsf{bindsTo}(z_5, z_6) \wedge \mathsf{isBoundTo}(z_6, z_5) \wedge \mathsf{DownloadingService}(z_6) \\
&q_{\mathsf{RebindingServiceSituation}} = \\
&\quad \exists x, y.\ \mathsf{AvailableResourceServer}(x) \wedge \mathsf{runs}(x, z_1) \wedge \mathsf{hosts}(z_1, z_2) \wedge \mathsf{Implementation}(z_2) \wedge \\
&\qquad \mathsf{bindsTo}(z_2, z_3) \wedge \mathsf{Service}(z_3) \wedge \\
&\qquad \neg \mathsf{OptimallyUtilizedServer}(y) \wedge \mathsf{runs}(y, z_4) \wedge \mathsf{hosts}(z_4, z_5) \wedge \mathsf{Implementation}(z_5) \wedge \mathsf{bindsTo}(z_5, z_6) \wedge \\
&\qquad \mathsf{isBoundTo}(z_6, z_5) \wedge \mathsf{Service}(z_6)
\end{aligned}$$

Fig. 1. The situation when to rebind a (downloading) service captured as query classes and conjunctive queries.

need to be specified as classes, while for the latter, the situations need to be described by conjunctive queries.

Example 4: A situation apt for rebinding a downloading service considers two servers, one with available resources and one that is not optimally utilized. The former hosts the corresponding implementation and the latter hosts the same implementation as is currently bound by the service. The resulting query class is displayed in Figure 1 in the upper half as the class RebindingDownloadingServiceSituation and the corresponding conjunctive query $q_{\mathsf{RebindingDownloadingServiceSituation}}$ in the lower half of the figure. Note that the fact that the servers use the same implementation cannot be expressed by a class description, since they only allow to describe tree-like structures. Furthermore, conjunctive queries retrieve tuples from the ABox, while a query concept can only retrieve a single individual.

In Figure 1, a situation that generalizes the above one is characterized in the query class RebindingServiceSituation and in the query $q_{\mathsf{RebindingDownloadingServiceSituation}}$, respectively. In this situation the service and the implementation are not further specified. 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 it is stated in the TBox that every DownloadingServer is a Server and that every DownloadingImplementation is an Implementation. Furthermore, assume that for a particular downloading implementation it cannot be retrieved that it is an implementation of that kind, but only that it is an implementation (of some kind). Thus, the next ABox is incomplete. In such a case, a situation that might necessitate the rebinding of a downloading service cannot be recognized. More precisely, the class RebindingDownloadingServiceSituation does not have an instance in the current ABox and the query $q_{\mathsf{RebindingDownloadingServiceSituation}}$ yields no tuples. However, the more general class RebindingServiceSituation would have an instance and the query $q_{\mathsf{RebindingServiceSituation}}$ would yield a result tuple. Thus, a counter measure could be invoked at least for this kind of situation.

Class and conjunctive queries differ in the expressive power for specifying the situations. While the former are limited by the expressivity of the ontology language, the latter can, in addition, make use of the variables to describe arbitrary structures to describe the details of the situations. This addition comes at the cost of higher computational complexity.

III. Evaluation for the OWL 2 profiles

The goal of our evaluation is to see whether current OWL 2 reasoners are appropriate for situation recognition in SMSs. 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. However, the syntactic restrictions of the lightweight profiles allow only for coarser modeling than full OWL 2—some information simply cannot be modeled. An interesting question is whether this leads to missing inferences in our scenario.

A. Test Data and Reasoning Systems

a) Test ontologies: Our base TBox from Section II-A contains 113 class and 66 property definitions and uses $\mathcal{ALCIQ}$, a sub-logic of OWL 2. For both lightweight profiles, we built variations of the base TBox manually—keeping as much information as possible. Since the OWL 2 QL profile does not support truly complex class descriptions, the situations in the OWL 2 QL TBox cannot be modeled as classes. However, the necessary information can be captured in the conjunctive queries. Thus, for the QL profile, we only apply conjunctive query answering.

The ABoxes model a video platform running on four servers and providing the services described in Section II. Since the class assertions use only named classes, 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 situation to be detected, but model the overall system state. We added data about other users requesting video services, which are carried out on other servers. This roughly doubles the sizes of both ABoxes. Each of the test ABoxes contains about 380 individuals, more than 770 class, and more than 545 property assertions.

b) Test Queries: We modeled 13 situations as OWL 2 classes. Since OWL 2 EL does not offer universal quantification, only 11 of them are modeled as OWL 2 EL classes. For these 11 situations we formulated the corresponding conjunctive queries included in our test suite. The class queries have a size of about 10 counting the class and property names. The conjunctive queries are formulated in the query languages SPARQL and nrql. They contain on average 15 disjuncts of conjunctions with 8 conjuncts each.

c) Reasoner Systems: The tests were run for seven DL reasoners, which differ w.r.t. the DL they support and the reasoning services provided. Table I 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 I, 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 data access (i.e., a data base functions as ABox and can be queried directly). We used QUEST with classical ABoxes, here.

B. 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 class and conjunctive queries, all reasoners agree on the result tuples. However, when comparing the results for conjunctive queries w.r.t. differing expressiveness of the profiles, it shows that RACERPRO detects all of the (expected) tuples for OWL 2, while less tuples are returned for the lightweight profiles. As to be expected, this is due to the loss in expressivity when using a lightweight profile. We observed the same effect for the lightweight profiles in the results of PELLET and QUEST.

Reasoner	Version	Query type		Profile		
		Class	Conj.	OWL	EL	QL
FACT++ [26]	v1.6.1	x		x	x	x
HERMIT [18]	v1.3.6	x		$\mathcal{SHOIQ}$	x	x
PELLET [21]	v2.3.0	x	x	$\mathcal{SHOIN}(\mathcal{D})$	x	x
RACERPRO [12]	v2.0	x	x	$\mathcal{SHIQ}(\mathcal{D})$	x	x
ELK [15]	v0.3.1	x			$\mathcal{EL}^+$	
JCEL [17]	v0.18.0	x			$\mathcal{EL}^+$	
QUEST [20]	v1.7-alpha		x			x

TABLE I
REASONERS AND THEIR SUPPORTED QUERIES AND PROFILES.

		Load.	Reason.	Avg/Query	Total
OWL	HERMIT	0.180	1.832	0.148	2.012
	PELLET	0.203	0.434	0.033	0.637
	FACT++	0.208	0.225	0.017	0.433
	RACERPRO	0.199	24.163	1.985	24.362
EL	ELK	0.228	0.078	0.004	0.306
	FACT++	0.245	0.063	0.002	0.308
	HERMIT	0.212	0.120	0.004	0.332
	JCEL	0.230	0.199	0.012	0.429
	PELLET	0.197	0.576	0.045	0.773
	RACERPRO	0.342	1.675	0.112	2.018

TABLE II
RUNTIMES FOR CLASS QUERIES IN SECONDS.

		Load.	Reason.	Avg/Query	Total
OWL	RACERPRO	1.302	40.035	3.336	41.336
EL	PELLET	0.522	2.344	0.195	2.866
	RACERPRO	0.503	6.773	0.564	7.277
QL	PELLET	0.541	1.918	0.160	2.460
	QUEST	0.453	93.208	7.767	93.661
	RACERPRO	0.349	6.604	0.550	6.953

TABLE III
RUNTIMES FOR CONJUNCTIVE QUERIES IN SECONDS.

1) Performance for Class Queries: For class queries, we ran tests for the 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 Table II sorted by profiles. The first column depicts the time spent on loading the ontology. The next one displays the time for answering *all* the 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 6-10 times higher for OWL 2 than for the OWL 2 EL profile, with the exception of PELLET, which performs slightly better for OWL.
OWL 2: Apart from RACERPRO, which took about 25 seconds, all reasoners delivered a full situation recognition within 2 seconds.
OWL 2 EL: With an overall runtime of about 0.3 seconds, ELK, FACT++, and HERMIT outperform the other systems. All systems can perform situation recognition within 0.8 seconds, apart fromRACERPRO, which, again, takes considerably more time.

2) Performance for Conjunctive Queries: For the conjunctive queries, the results for all of the three profiles are displayed in Table III. As for class querying, reasoning in the lightweight profiles is much faster.
OWL 2: Here, RACERPRO needs about 41 seconds overall runtime. Interestingly, it takes nearly twice as long as for the corresponding class query, due to one outlier query.
OWL 2 EL: PELLET answers all queries in less than 3 seconds, but takes about four times as long as for the class queries. With 7.2 seconds, RACERPRO takes more than twice as long than PELLET.
OWL 2 QL: The times of PELLET and RACERPRO are similar to the OWL 2 EL case. QUEST, in contrast, shows

a significantly worse performance by taking more than 1.5 minutes. We conjecture that this is attributed to running a first alpha version of QUEST and using a traditional ABox (i.e., instead of a database).

All in all, the experiments show that most state of the art reasoners can be applied for situation recognition in our SMS application, since response times of half a minute would be acceptable. Especially by the use of the lightweight profiles, we achieve very good runtimes for reasoning. Surprisingly, the loss of information when using a lightweight profile turned out to be only marginal for our video platform use case.

IV. CONCLUSIONS AND FUTURE WORK

We have supplied a study on employing state of the art DL reasoners to perform situation recognition for service management applied to a video platform. The task was to recognize complex situations that might invoke rebinding of services in order to achieve energy efficiency. To solve this task, the domain was modeled in an OWL 2 ontology, where the ABox reflected realistic situations in the application. The actual recognition of critical situations was realized by class and 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 3 seconds.

Future work on the practical side includes 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. 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 that contain this kind of information.

REFERENCES

[1] D. Abts, M. R. Marty, P. M. Wells, P. Klausler, and H. Liu. Energy proportional datacenter networks. *SIGARCH Comput. Archit. News*, 38(3):338–347, 2010.

[2] F. Ahmad and T. N. Vijaykumar. Joint optimization of idle and cooling power in data centers while maintaining response time. In *Proc. of the fifteenth edition of ASPLOS on Architectural support for programming languages and operating systems*, ASPLOS '10, p. 243–256, USA, 2010. ACM.

[3] 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 *Proc. of the 18th Int. Conf. on Automated Reasoning with Analytic Tableaux and Related Methods (Tableaux'09)*, vol. 5607 of *LNCS*, p. 77–92. Springer, 2009.

[4] F. Baader, S. Brandt, and C. Lutz. Pushing the $\mathcal{EL}$ envelope further. In K. Clark and P. F. Patel-Schneider, eds. *In Proc. of the OWLED Workshop*, 2008.

[5] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. Patel-Schneider, eds. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2003.

[6] L. Barroso and U. Hölzle. The Case for Energy-Proportional Computing. *Computer*, 40(12):33–37, Dec. 2007.

[7] A. Borgida, M. Lenzerini, and R. Rosati. Description logics for databases. In *[5]*, p. 462–484. Cambridge University Press, 2003.

[8] 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. *J. of Automated Reasoning*, 39(3):385–429, 2007.

[9] D. Calvanese, G. De Giacomo, M. Lenzerini, D. Nardi, and R. Rosati. Knowledge representation approach to information integration. In *Proc. of AAAI Workshop on AI and Information Integration*, p. 58–65. AAAI Press/The MIT Press, 1998.

[10] C. Clark, K. Fraser, S. Hand, J. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield. Live migration of virtual machines. In *Proc. of the 2nd Conference on Symposium on Networked Systems Design & Implementation-Vol. 2*, p. 273–286. USENIX Association, 2005.

[11] W. Dargie, A. Strunk, and A. Schill. Energy-aware service execution. In *Proc. of the 36th Annual IEEE Conference on Local Computer Networks*, 2011.

[12] V. Haarslev, K. Hidde, R. Möller, and M. Wessel. The RacerPro knowledge representation and reasoning system. *Semantic Web Journal*, 3(3):267–277, 2012.

[13] I. Horrocks, O. Kutz, and U. Sattler. The even more irresistible $\mathcal{SROIQ}$. In P. Doherty, J. Mylopoulos, and C. Welty, eds. *Proc. of the 10th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR-06)*, p. 57–67. AAAI Press, 2006.

[14] Y. Kazakov. $\mathcal{RIQ}$ and $\mathcal{SROIQ}$ are harder than $\mathcal{SHOIQ}$. In G. Brewka and J. Lang, eds. *Proc. of the 11th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR-08)*, p. 274–284. AAAI Press, 2008.

[15] Y. Kazakov, M. Krötzsch, and F. Simančík. ELK reasoner: Architecture and evaluation. In I. Horrocks, M. Yatskevich, E. Jimenez-Ruiz, editor, *Proc. of the OWL Reasoner Evaluation Workshop (ORE'12)*, vol. 858 of *CEUR*, 2012.

[16] C. Lutz. The complexity of conjunctive query answering in expressive description logics. In A. Armando, P. Baumgartner, and G. Dowek, eds. *Proc. of the 4th International Joint Conference on Automated Reasoning (IJCAR'08)*, nr. 5195 in LNAI, p. 179–193. Springer, 2008.

[17] 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*, 2012.

[18] 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, Germany, 2007. Springer.

[19] B. Neumann and R. Möller. On scene interpretation with description logics. In H. Christensen and H.-H. Nagel, eds. *Cognitive Vision Systems: Samping the Spectrum of Approaches*, nr. 3948 in LNCS, p. 247–278. Springer, 2006.

[20] M. Rodriguez-Muro and D. Calvanese. Quest, an OWL 2 QL reasoner for ontology-based data access. In *Proc. of the 9th Int. WS on OWL: Experiences and Directions (OWLED'12)*, vol. 849 of *CEUR*, 2012.

[21] E. Sirin and B. Parsia. Pellet system description. In B. Parsia, U. Sattler, and D. Toman, eds. *Description Logics*, vol. 189 of *CEUR*, 2006.

[22] T. Springer and A.-Y. Turhan. Employing description logics in ambient intelligence for modeling and reasoning about complex situations. *J. of Ambient Intelligence and Smart Environments*, 1(3):235–259, 2009.

[23] A. Strunk and W. Dargie. Does live migration of virtual machines cost energy? In *Proc. of the 27th IEEE International Conference on Advanced Information Networking and Applications (AINA-2013)*, 2013.

[24] K. Taylor and L. Leidinger. Ontology-driven complex event processing in heterogeneous sensor networks. In *Proc. of 8th Extended Semantic Web Conference (ESWC'11)*, vol. 6644 of *LNCS*, p. 285–299. Springer, 2011.

[25] S. Tobies. *Complexity Results and Practical Algorithms for Logics in Knowledge Representation*. PhD thesis, RWTH Aachen, 2001.

[26] D. Tsarkov, I. Horrocks, and P. F. Patel-Schneider. Optimising terminological reasoning for expressive description logics. *J. of Automated Reasoning*, 2007.

[27] 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, eds. *Proc. of the Workshop on Context Modeling and Reasoning (CoMoRea'06)*. IEEE Computer Society, 2006.

[28] 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/.