Bridging the Application Knowledge Gap

Using Ontology-based Situation Recognition to Support Energy-Aware Resource Scheduling

Marcus Hähnel
Chair of Operating Systems
Institute of Systems Architecture
Technische Universität Dresden
mhaehnel@tudos.org

Julian Mendez, Veronika Thost, Anni-Yasmin Turhan
Chair for Automata Theory
Institute for Theoretical Computer Science
Technische Universität Dresden
lastname@tcs.inf.tu-dresden.de

ABSTRACT

Regarding energy efficiency, resource management in complex hard- and software systems that is based on the information typically available to the OS alone does not yield best results. Nevertheless, general-purpose resource management should stay independent of application-specific information. To resolve this dilemma, we propose a generic, ontology-based approach to resource scheduling that is context-aware and takes information of running applications into account. The central task here is to recognize situations that might necessitate an adaptation of resource scheduling. This task is performed by logical reasoning over OWL ontologies. Our initial study shows that current OWL 2 EL reasoner systems can perform recognition of exemplary situations relevant to resource management within 4 seconds.

Categories and Subject Descriptors

D.4.7 [**Operating Systems**]: Organization and Design; I.2.4 [**Artificial Intelligence**]: Knowledge Representation Formalisms and Methods

General Terms

Management, Design

Keywords

resource management,middleware,reasoning,energy

1. INTRODUCTION

Today's mobile, desktop and server systems become more adaptive and heterogeneous in their resources. Further, energy consumption is a growing concern, not only in mobile systems with limited power supply, but also in stationary systems, due to increasing energy prices and limits to the capability to provide energy to data-centers. This motivates our study on a context-aware resource scheduler that adapts to a more energy-efficient configuration depending on the current situation in the managed system.

ARM'14 December 9, 2014, Bordeaux, France

http://dx.doi.org/10.1145/2677017.2677020

1.1 Energy-Aware Resource Scheduling

To exploit the capabilities of heterogeneous architectures, resources must be managed according to application demand. In particular, a system's resource manager must find a configuration that provides the applications' required resources in the most energy-efficient way. However, the forecasting of the amount of resources an application can benefit from is non-trivial and usually involves knowledge which the application cannot easily communicate to lower-layer resource managers. In general, resource managers base their scheduling decisions on the current resource utilization or its recent history. However, a very high utilization of a resource does not necessarily mean that applications can benefit from more of it. In fact, it is often strongly application-dependent whether there is an actual benefit from additional resources. To cater to a changing demand for resources over time, the resource manager should take application knowledge into account. A similar problem arises with machine scheduling where, based on load, machines are turned off or on to save power. Turning on a new machine too late results in reduced service quality. Yet, turning it on without actually needing it wastes energy. In many cases, applications have information on whether the load increases in the near future (e.g., based on information on user types classified by their past behavior). This information should be used by the resource scheduler.

While for short-term decisions—such as resource configuration changes that are performed quickly and that are re-evaluated in short time intervals—the use of heuristics is essential to provide a responsive system, this is not the case for long-term decisions and configuration changes that may provide long-term energy-savings. Gathering and performing sophisticated analysis on application knowledge are here key to an energy-efficient system.

Since the resource manager should be as generic as possible and therefore not directly include application-specific information, it is a challenge to enable the system to exploit application knowledge while keeping the separation of concerns. To achieve this, an approach for bridging the gap from the resource manager to the application knowledge is needed that fulfills the following requirements:

- integration of information at different levels of detail and from heterogeneous sources into a coherent view.

- support the representation and recognition of *critical situations* under possibly incomplete and noisy information.

We propose an ontology-based approach to bridge the gap and that acts as a dedicated glue layer between the knowledge from different kinds of applications and the resource manager itself. That is, we employ logical reasoning over an ontology that captures the status of the system to recognize *critical situations* in which the resource manager might want to adapt. More precisely, knowledge of running applications and information on the current system state is captured in an ontology, and the situations to be detected are formulated as queries over this ontology. Then, an ontology reasoner evaluates the queries over the ontology and notifies the resource manager in case it recognizes critical situations. The latter alert is issued via a simple, generic interface—for example, in case the resource manager should power up an additional server. This results in a generic approach for reflective resource scheduling and could thus be used for other non-functional requirements such as robustness or security.

1.2 Situation Recognition by Standard OWL 2 EL Reasoning

We model ontologies that capture our hard- and software components in the ontology language OWL 2 EL, one of the *lightweight* profiles [10] of the Web ontology language OWL[14]. Ontologies offer a graceful way of integrating multiple information sources on different levels of abstraction. Moreover, OWL has formal semantics, which are a prerequisite for well-defined *reasoning services* that allow to infer implicitly captured facts from the given ones. In our case, the answering of class queries or conjunctive queries are of interest. However, these reasoning services are of high computational (worst case) complexities. For example, in OWL 2 EL, which corresponds to the DL $\mathcal{EL}^{++}$, class query answering is in P [3], 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 ontology. This may suggest that employing ontology-based reasoning for the resource scheduling problem at hand cannot be done within a useful time interval. Thus, it is interesting to find out whether the task of situation recognition can be supported by today's OWL reasoning systems. We apply class and conjunctive query answering to query the ontology for the situations to be recognized.

In OWL, in general, categories from the application domain can be described by *classes* and binary relations by so-called *properties*. The class Server, e.g., can be characterized as hardware with a part that is of class CPU and a part that is of class memory by the expression:

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

This definition assigns to the named class Server the complex class expression on the right-hand side using the property hasPart and the other named classes HW, CPU, and Memory. Based on Server, we can define an OffServer as a server that has power state 'off':

$$\mathsf{OffServer} \equiv \mathsf{Server} \sqcap \exists \mathsf{hasPowerState}.\mathsf{Off}$$

Definitions of classes are stored in the *TBox*, which is one of the two parts of an ontology. The second part, called *ABox*, stores concrete facts about the application—either as *class assertions* that state class membership of an individual or as *property assertions* that relate two individuals via a property. For instance, in ABox $\mathcal{A}_1$, below, we state that the individual *S1* belongs to the class Server and that its related power state is individual *PS2*, which belongs to the class Off, by writing the statements:

$$\mathcal{A}_1 = \{\mathsf{Server}(S1),\ \mathsf{hasPowerState}(S1, PS2),\ \mathsf{Off}(PS2)\}.$$

For OWL, there are several *reasoning services* that can infer from the explicitly given information in the ontology implicitly captured facts. *Class queries* compute, for a given class (expression) $\mathsf{C_q}$ and an ontology, all the individuals in the ABox that belong to the class $\mathsf{C_q}$. For example, for the query class OffServer, the above class definitions, and $\mathcal{A}_1$, the instance *S1* is derived. A more powerful way to query the ABox are conjunctive queries. A *conjunctive query* is a conjunction of assertions where variables can be used instead of individuals. For example,

$$q_{ex}(x_1, x_2) = \exists y.\mathsf{Server}(x_1) \wedge \mathsf{Process}(x_2) \wedge \mathsf{uses}(x_2, y) \wedge \mathsf{hasPart}(x_1, y)$$

is a conjunctive query asking for all pairs of servers and processes where the process uses some part of the server.

To apply OWL reasoning to support energy-efficient resource scheduling, we model the system's basic categories and relations such as the hardware and application-specific knowledge in a TBox. The current state of the system is then captured at runtime in an ABox, similarly to [2, 7, 8, 12]. To recognize critical situations for the resource manager, we employ the answering of class and conjunctive queries. Once such a situation is detected for a (tuple of) ABox individual(s), the resource manager is notified and can (decide to) invoke appropriate adaptations.

In this paper, we take a video platform as a use case to demonstrate the feasibility of our ontology-based approach to support resource scheduling decisions. Our contributions are as follows:

- Identification of situations characterized by application-specific knowledge and whose recognition would improve the system's energy-efficiency.
- Modeling of an OWL 2 EL ontology $\mathcal{O}_{\mathrm{VP}}$ which integrates knowledge about the OS and application-specific knowledge.
- Evaluation of the performance of several state-of-the-art OWL 2 EL reasoners w.r.t. the recognition of critical situations modeled as queries over $\mathcal{O}_{\mathrm{VP}}$.

This paper complements the picture on how highly adaptive and complex hard- and software systems can be supported by situation recognition through OWL reasoning. In the past, we studied situation recognition to achieve energy-efficiency for different applications—by switching between software variants [8] and for service management systems [7]. While these studies were targeted more towards coarser adaptations and therefore explored *which OWL 2 profiles* to use, the study at hand investigates whether adaptations on the OS level can be supported by systems implementing logical reasoning at all. Given different levels of a complex hard- and software system, each modeled by OWL 2 ontologies and performing situation recognition by logical reasoning, it is not hard to see that our approach can be extended

to implement cross-layer integration with the ontology as central platform. Though, in such a setting, further issues need to be addressed. A first idea for resolving conflicting adaptations between the layers is, for example, to prioritize the alerts raised when a situation is recognized.

The remainder of the paper is structured as follows: Section 2 covers related work, and Section 3 describes the ontology for the video platform use case and the modeling of the relevant situations. Section 4 then presents an empirical evaluation of how current OWL 2 EL reasoners perform in our scenario, by considering class and conjunctive query answering over our ontology.

2. RELATED WORK

There have been several attempts to incorporate reasoning approaches into resource management. In [6] Chantaraskul et al. have shown a system to use agents to recognize previously observed traffic patterns in W-CDMA networks using case-based reasoning and configuring management policies accordingly. Further Attard et al. [1] have shown an ontology-based approach to recognize personal, recurring situations of mobile device users and adapt their behavior accordingly. Pandis et al. [11] work with ontology-based methods to manage dynamic resource registration and invocation. Wang et al. [16] have built a system the enables efficient management of host and guest resources in VM scenarios with the help of cross-layer in-vm application knowledge. This knowledge is then used to feed an adaptive learning component that can model the VMs resource usage characteristics. This resource usage is directly exported to the resource manager. Neal [15] has proposed a method for stakeholder-directed resource allocation where application knowledge is used in the form of priorities and weights to guide memory allocation schemes.

While all these approaches try to manage resources using application knowledge, none of them can cope with heterogeneous resources, picking the best fit based on user requirements and application specific resource usage information. To the best of our knowledge, we are the first to use ontology-based reasoning to bridge the gap from generic resource management for highly adaptive, heterogeneous hardware to application knowledge using an ontology-based approach that does not incur direct changes in the resource manager. Resource Containers by Banga et al. provide fine-grained resource management primitives letting the OS assign resources to an application, separating resource principals from protection domains [4]. This allows the application to manage resources on its own. Unlike more direct and simple approaches for resource scheduling, our approach offers the opportunity to realize cross-layer integration as well.

In more general settings, the idea of employing logical reasoning and ontology-based approaches for situation recognition is certainly not new. There are investigations on a variety of domains from the last decade. However, when it comes to studies on energy-efficient computing in complex hard- and software systems, then our earlier studies [7, 8] are the only ones we are aware of.

3. USE CASE: MANAGING A VIDEO PLATFORM

To investigate whether resource management can be supported by OWL reasoning, we consider a video platform focusing on feature films as our example scenario. The platform uses several servers, which are connected by a heterogeneous network setup as suggested in [9]. The application running on this platform is distributed over several servers and can expand and shrink based on the required resources. The video platform allows users to search for, upload, and stream videos. It further uses transcoding to provide the uploaded videos in various quality levels and applies a ranking algorithm to determine popular videos. In this system, the up- and downloading of videos is resource-intensive for the network, while the transcoding is CPU- and memory-intensive and possibly requires fast storage. Users of the platform need to log in before being able to up- or download a film. Furthermore, they may use the services for free, but only get guaranteed service quality (i.e., w.r.t. transcoding times, streaming bandwidth, etc.) if they pay a fee. This guaranteed service quality is also known as a *service level agreement* (SLA) between the provider of the service and the user. If the agreed quality is not provided, this is a *SLA violation*.

Next to providing the video services in the agreed quality, the main goal of the system's resource manager is to achieve this in a cost-effective way. That is, energy consumption should be kept low while SLA violations are being prevented.

3.1 The Application Knowledge Gap

We identified several cases where generic, energy-aware resource scheduling can benefit from application knowledge that is present on the video platform. For instance, upcoming streaming jobs can be predicted early by taking knowledge on login events (e.g., that a user wants to stream) or about user context (e.g., that she uses a high-speed connection, which is known from earlier requests) into account. Hence, application knowledge can help to influence the decision on when to proactively

- power-up servers without interruption or slowdown of service.
- in- or decrease the available network bandwidth as, e.g., required by the network energy management [9].
- identify processes to be throttled (i.e., lowered in priority) or paused temporarily—for example, if prioritized users need computing power, but no other resources are left.

In the first two cases, the usual approach, a naïve threshold-based decision considering performance metrics such as network or CPU utilization, may lead to increased energy consumption or even to reduced service quality. In case three, the decisions depending on priority require information on the kind of processes and their importance for the application. Moreover, decisions about priority cannot be made at all without explicit information on the kind of processes and their importance for the application. Due to space restrictions, we focus on the proactive power-up of servers, in the following. The scheduling of additional servers is usually performed when a certain threshold of required bandwidth is crossed (e.g., a utilization of 90 %). Once the system detects such a threshold crossing, it powers up an additional server to guarantee service quality.

This is a very crude metric. Using this method we can neither determine in advance whether the additional band-

$$\mathsf{PowerUpServerSituation} \equiv \exists \mathsf{hasRequests}.(\mathsf{StreamingReqsNotProcessedYet} \sqcap \exists \mathsf{mayRequestResources}.\mathsf{NotProvidableResourceConfig}) \sqcap \exists \mathsf{hasServer}.(\mathsf{StreamingServer} \sqcap \exists \mathsf{hasPowerState}.\mathsf{Off})$$

$$q_{\mathsf{PowerUpServerSituation}}(x_1, x_2) = \exists y_1, y_2, y_3.\ \mathsf{VideoPlatform}(y_1) \wedge \mathsf{hasRequests}(y_1, x_2) \wedge \mathsf{StreamingReqsNotProcessedYet}(x_2) \wedge \mathsf{mayRequestResources}(x_2, y_2) \wedge \mathsf{NotProvidableResourceConfig}(y_2) \wedge \mathsf{hasServer}(y_1, x_1) \wedge \mathsf{StreamingServer}(x_1) \wedge \mathsf{hasPowerState}(x_1, y_3) \wedge \mathsf{Off}(y_3)$$

Figure 1: The situation to 'power up an additional server' captured as a class query and conjunctive query.

width will actually be useful (considering the user's bandwidth capacity), nor assess if it will be required at all. This is of importance since a threshold closer to 100 % would enable more energy savings and less needless powering of servers. However, it would also lead to the danger of late server power-ups and thus to brief reductions in service quality.

The shortcomings of this naïve approach in the previous example can be traced back to insufficient situation-awareness and especially to the lack of application knowledge, an obstacle that we call the *Application Knowledge Gap*. While existing approaches can take application specific knowledge into account, they are mostly limited to a direct interface between the resource manager and the application. These interfaces must either use a unifying abstraction such as a currency [17] or must use application specific mechanisms that can cope with direct input from the application. The resource managers, nevertheless, should not implement such application-specific interfaces because this would result in a loss of their generality and require an extension for each new application. Other approaches such as using a currency force application developers to use (seemingly arbitrary) values that have no meaning in their domain of expertise.

To bridge the Application Knowledge Gap, we create ontologies capturing information about the system, explicitly describe situations where the (non-)application of adaptation may be beneficial, apply an OWL 2 EL reasoner to process the application-specific knowledge (of multiple applications), and that notify the OS once such a critical situation is recognized. This enables, for example, the early power-up of an additional server if a user known to perform heavy downloads is logging into the application when the system is already nearly fully utilized.

3.2 Ontology-based Situation Recognition

At design time, the relevant situations to be recognized are modeled as (query) classes or conjunctive queries. These queries, in turn, use classes referring to domain knowledge and need to be described in the TBox. In our case, general domain knowledge about video platforms (e.g., characteristics of a StreamingServer), and notions specific for resource management (e.g., conditions for a requested resource configuration to be a ProvidableResourceConfiguration) are described in the TBox. In particular, the TBox captures application-specific classes relevant to resource scheduling, for instance,

$$\mathsf{ThrottleableProcess} \equiv \exists \mathsf{exec}.(\mathsf{Request} \sqcap \exists \mathsf{issuedBy}.\mathsf{NotPayingUser}).$$

The ABox describes the architecture of the specific video platform instance handled by the resource manager (e.g., available servers) and the current state of its components (e.g., servers on power, network utilization, etc.) as well as application knowledge (e.g., users logged onto the platform). Most of the data in the ABox is dynamic and therefore needs to be collected at runtime. Due to the highly dynamic nature of the system, the ABox is refreshed several times a minute (e.g., every second). Further, ABoxes are generated from several sources such as sensor data delivered by the OS or databases with application-specific knowledge. For the task of converting numerical (sensor) data into a symbolic representation (i.e., named classes), *preprocessors* are applied to convert the numeric data into named classes (as in [2, 13]). For example, if the power state of $S1$ in ABox $\mathcal{A}_1$ from Section 1.2 has changed to 'on' in the last interval, the assertions: $\mathsf{hasPowerState}(S1, PS2)$ and $\mathsf{On}(PS2)$ are added to the ABox that is created for the past interval.

Once the ABox is updated, the reasoner performs the answering of the class queries or conjunctive queries to recognize the critical situations. Under the assumption that bandwidth details are available for at least some requests, a situation apt for powering up an additional server could be the following: The yet unprocessed streaming requests may demand for resources, say bandwidth, that cannot be provided currently, and there is a server available, that is off power and dedicated to streaming. The resulting class query is displayed in the upper part of Figure 1 as class PowerUpServerSituation and the corresponding conjunctive query $q_{\mathsf{PowerUpServerSituation}}(x_1, x_2)$, is given in the lower part. Please note that, with a conjunctive query, we can directly query for the available server and the request in question. In principle, conjunctive queries allow to specify arbitrary relational structures by the use of variables, while concept queries in OWL 2 EL allow to specify only relational structures of tree shape. Thus, conjunctive queries are more expressive than concept queries, but also computationally more expensive.

4. EXPERIMENTS

The goal of our experiments is to test whether current OWL 2 EL reasoners can be applied for situation recognition to support resource management. The reasoners have to be able to detect situations while processing realistic amounts of data within short time. To this end, we specified 6 rather different situations as class and as conjunctive queries in OWL 2 EL and let them be answered by current reasoner systems over $\mathcal{O}_{\mathrm{VP}}$.

4.1 Test Data and Reasoning Systems

4.1.1 Test ontologies

The TBox of $\mathcal{O}_{\text{VP}}$ described in Section 3.2 contains about 200 class and 50 property definitions. Since our domain is of hierarchical nature (i.e., when it comes to the part-of property of the hardware), we can model it in OWL 2 EL—despite the limited expressivity of this profile.

We consider three different ABoxes modeling three different states of the system. Each ABox models a mid-sized video platform running on 20 servers and providing the services described in Section 3. For one system state, we modeled, for instance, about 1,000 streaming, 40 uploading, and 200 transcoding jobs being currently processed. This leads to test ABoxes each containing more than 1,000 individuals, more than 1,500 class, and more than 104,000 property assertions.

4.1.2 Test Queries

We have modeled 5 situations as OWL 2 EL classes. Interestingly, one situation could not be represented as an OWL 2 EL class, since it already referred to parts of the adaptation the resource manager would need to perform. For such adaptations the open world semantics of OWL is not appropriate. Without the adaptation aspects this concept query would have been trivial, and thus we did not include it in our tests. In contrast, all 6 situations could be described by conjunctive queries, because the query languages offer the necessary additional features. The average size for both the class queries (counting class and property names) and the conjunctive queries (counting number of conjuncts) is seven. The conjunctive queries are formulated in the query languages SPARQL[1] and nrql[2]. The rather small size of the queries is due to the fact that most raw data in the OS context is numerical and needs to be preprocessed to obtain symbolic OWL classes. Such named classes then capture rather much information (e.g., ProvidableResourceConfiguration). Since concept queries offer less expressivity than conjunctive queries, we had to use more advanced preprocessing to generate the ABoxes for the corresponding tests.

4.1.3 Reasoner Systems

The tests were run for 7 reasoners that differ w.r.t. the OWL profiles and the reasoning services supported (i.e., not all of them support both types of queries). Note that, apart from the reasoners ELK and jCel (both are specialized for a sublanguage of OWL 2 EL), all reasoners support full OWL 2 EL and are freely available.

4.2 Results

The tests were carried out on an Intel(R) Xeon(R) E5-2640 2.50GHz machine with 96 GB RAM using Java 1.7.0_51 and running Linux 3.2.0-4-amd64. To access the reasoners in a uniform way, we used the OWL API (v 3.5.0) if possible[3]. Besides the runtimes, we checked whether the reasoners delivered the same results for the respective queries. This was the case for all but TrOWL, which returned too many answers for two queries.

4.2.1 Performance for Class Queries

Reasoner	Version	Load.	Reason.	Avg/Q	Total
ELK [4]	v0.4.1	3.614	1.302	0.063	4.916
FaCT++ [5]	v1.6.3	2.392	1.188	0.109	3.580
HermiT [6]	v1.3.8	2.151	1.774	0.210	3.925
jCel [7]	v0.20.0	3.656	10.475	1.304	14.131
Pellet [8]	v2.3.1	1.229	32.515	5.328	33.743
RacerPro [9]	v2.0.0	5.234	1.958	0.326	7.193
TrOWL [10]	v1.4.0	1.554	0.247	0.025	1.800

Table 1: Runtimes for class queries in seconds.

Reasoner	Load.	Reason.	Avg/Q	Total	Total'
Pellet	1.792	39.699	4.017	41.478	32.542
RacerPro	5.123	11.643	1.902	16.766	7.715
TrOWL	1.585	32.769	5.460	34.354	1.956

Table 2: Runtimes for conjunct. queries in seconds.

The runtimes measured for class-query answering are displayed in Table 1 (in seconds). The rows of the table show the time spent on loading the ontology, answering *all* the queries, the average runtime per query, and the total runtime of the situation recognition. The last column shows the total time situation recognition would take in our scenario. Apart from RacerPro, all reasoners took less than 4 seconds for loading $\mathcal{O}_{\text{VP}}$. There are however major differences between the reasoners w.r.t. the reasoning times. Yet, the fastest reasoners performed the situation recognition within 4 seconds.

4.2.2 Performance for Conjunctive Queries

The task of situation recognition was more difficult for the conjunctive queries, since less heavily preprocessed concepts were used in $\mathcal{O}_{\text{VP}}$ and because the answering of conjunctive queries has a higher computational complexity; this is reflected in the higher times it took for reasoning, displayed in Table 2. Note that, since the situation that had not been representable as class query also lead to a rather unnatural conjunctive query, Table 2 shows an additional total runtime for a run considering only the five other situations. And though the task for answering conjunctive queries was more complex, TrOWL succeeds in performing the whole task within only 2 seconds.

To sum up, it turned out that OWL 2 EL can model the knowledge about our video platform faithfully (i.e., without preprocessed concepts that are seemingly unnatural) and even better so if we apply conjunctive query answering. Moreover, our experiments show that state-of-the-art OWL 2 EL reasoners can perform situation recognition for exemplary resource scheduling situations within 4 seconds, which is acceptable to invoke mid-term adaptations (e.g., power on servers) or adaptations that are done proactively, in advance to the actual situation requiring the adaptation. Note that the very good performance of TrOWL should be treated with caution given the fact that its answers were not sound in all our test cases.

5. CONCLUSIONS AND FUTURE WORK

[1] http://www.w3.org/TR/sparql11-query/
[2] http://racer.sts.tuhh.de/ [3] Note that not all of the reasoners do support this interface, yet.
[4] http://code.google.com/p/elk-reasoner/
[5] http://code.google.com/p/factplusplus/
[6] http://hermit-reasoner.com/
[7] http://jcel.sourceforge.net/ [8] http://clarkparsia.com/pellet/
[9] http://racer.sts.tuhh.de/ [10] http://trowl.eu/

The case study presented here investigated how to support resource scheduling by application knowledge via ontology-based situation recognition. To this end, we specified exemplary situations where (non)adaptation leads to better energy-efficiency, but where the actual recognition of the situation depends on application knowledge. We further employed OWL 2 EL ontologies to integrate the system knowledge and used query answering to recognize such situations. All in all, our experiment underlined that several state-of-the art reasoning systems can process realistic amounts of data and recognize a small set of situations in acceptable time. Thus, we showed that the application knowledge gap can be closed using a generic method that does not require to adapt the resource scheduler to each new application.

While this study focused on situation recognition to accomplish energy-efficiency, our approach can easily be applied to achieve other non-functional requirements, such as security or robustness, as well. Furthermore, our approach bears the potential to be extended for cross-layer integration and hence assist to resolve conflicting adaptations.

In the future, we want to implement our approach in a real-world setting—especially, to investigate the amount of saved energy as well as the influence on service quality. Further, we plan to consider situations that include temporal information, which allows us to specify many real-world situations more accurately. We have already conducted theoretical research in this direction [5]. In particular, we considered a combination of conjunctive queries over *DL-Lite* ontologies and operators of the propositional linear temporal logic LTL. Since these queries can be translated into queries over databases, this approach suits many practical settings.

Acknowledgments

This work was partially funded by the German Research Council (DFG) through the Collaborative Research Center CRC 912 "Highly-Adaptive Energy-Efficient Systems" (HAEC) and the cluster of excellence "Center for Advancing Electronics Dresden".

References

[1] J. Attard, S. Scerri, I. Rivera, and S. Handschuh. Ontology-based situation recognition for context-aware systems. In *Proc. of the 9th International Conference on Semantic Systems (I-SEMANTICS '13)*, I-SEMANTICS'13, pages 113–120. ACM, 2013.

[2] 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 2009)*, volume 5607 of *LNCS*, pages 77–92. Springer, 2009.

[3] F. Baader, S. Brandt, and C. Lutz. Pushing the el envelope further. In K. Clark and P. F. Patel-Schneider, editors, *Proc. of the OWLED 2008 DC Workshop on OWL: Experiences and Directions*, 2008.

[4] G. Banga, P. Druschel, and J. C. Mogul. Resource containers: A new facility for resource management in server systems. In *Proceedings of the Third Symposium on Operating Systems Design and Implementation*, OSDI '99, pages 45–58, Berkeley, CA, USA, 1999. USENIX Association.

[5] S. Borgwardt, M. Lippmann, and V. Thost. Temporal query answering in the description logic *dl-lite*. In P. Fontaine, C. Ringeissen, and R. A. Schmidt, editors, *Proceedings of the 9th International Symposium on Frontiers of Combining Systems (FroCoS 2013)*, volume 8152 of *Lecture Notes in Computer Science*, pages 165–180, Nancy, France, 2013. Springer-Verlag.

[6] S. Chantaraskul and L. Cuthbert. Using case-based reasoning in traffic pattern recognition for best resource management in 3g networks. In *Proc. of the 7th ACM International Symposium on Modeling, Analysis and Simulation of Wireless and Mobile Systems (MSWiM'04)*, MSWiM '04, pages 252–259. ACM, 2004.

[7] W. Dargie, Eldora, J. Mendez, C. Möbius, K. Rybina, V. Thost, and A.-Y. Turhan. Situation recognition for service management systems using OWL 2 reasoners. In *Proc. of the 10th IEEE Workshop on Context Modeling and Reasoning (CoMoRea'13)*, pages 31–36. IEEE Computer Society, 2013.

[8] S. Götz, J. Mendez, V. Thost, and A.-Y. Turhan. OWL 2 reasoning to detect energy-efficient software variants from context. In K. Srinivas and S. Jupp, editors, *Proc. of the 10th OWL: Experiences and Directions Workshop (OWLED 2013)*, 2013.

[9] M. Hähnel, B. Döbel, M. Völp, and H. Härtig. ebond: Energy saving in heterogeneous r.a.i.n. In *Proc. of the Fourth International Conference on Future Energy Systems (e-Energy '13)*, e-Energy'13, pages 193–202. ACM, 2013.

[10] B. Motik, B. Cuenca Grau, I. Horrocks, Z. Wu, A. Fokoue, and C. Lutz. OWL 2 web ontology language profiles. W3C Recommendation, October 2009. `http://www.w3.org/TR/2009/REC-owl2-profiles-20091027/`.

[11] I. Pandis, J. Soldatos, A. Paar, J. Reuter, M. Carras, and L. Polymenakos. An ontology-based framework for dynamic resource management in ubiquitous computing environments. In *Embedded Software and Systems, 2005. Second International Conference on*, 2005.

[12] 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.

[13] K. Taylor and L. Leidinger. Ontology-driven complex event processing in heterogeneous sensor networks. In G. Antoniou, M. Grobelnik, E. Simperl, B. Parsia, D. P. De Leenheer, and J. Pan, editors, *The Semanic Web: Research and Applications*, volume 6644 of *LNCS*, pages 285–299. Springer, 2011.

[14] W3C OWL Working Group. OWL 2 web ontology language document overview. W3C Recommendation, October 2009. `http://www.w3.org/TR/2009/REC-owl2-overview-20091027/`.

[15] N. H. Walfield. Viengoos: A framework for stakeholder-directed resource allocation. Technical report, The Johns Hopkins University, October 2008.

[16] L. Wang, J. Xu, and M. Zhao. Application-aware cross-layer virtual machine resource management. In *Proc. of the 9th International Conference on Autonomic Computing (ICAC '12)*, ICAC'12, pages 13–22, New York, NY, USA, 2012. ACM.

[17] H. Zeng, C. S. Ellis, A. R. Lebeck, and A. Vahdat. Currentcy: A unifying abstraction for expressing energy management policies. In *Proc. of the Annual Conference on USENIX Annual Technical Conference (ATEC '03)*, ATEC'03, pages 4–4, Berkeley, CA, USA, 2003. USENIX Association.