A Performance Evaluation of Ontology-based Context Reasoning *
(Experience report)

Alessandra Agostini Claudio Bettini Daniele Riboni

DICo, University of Milan
via Comelico 39, I-20135 Milan, Italy
{*agostini,bettini,riboni*}*@dico.unimi.it*

Abstract

The CARE *middleware aims at supporting context-aware adaptation of Internet services in a mobile computing environment. The* CARE *hybrid reasoning mechanism is based on a loose interaction between ontological reasoning and efficient reasoning in a restricted logic programming language. In this paper we report recent experimental results on ontology-based context reasoning that support the hybrid approach.*

1 Introduction

In the last years, the research group of the DaKWE laboratory at the University of Milan has been working at the specification and implementation of a middleware – named *CARE*[1] – to support context-aware adaptation of Internet services for mobile users. Since Internet services can be possibly accessed by a huge number of users at a time, efficiency and scalability are mandatory. As a consequence, various efficient contextual reasoning procedures have been proposed for specific applications like telecommunication services (e.g., [11]) and e-commerce (e.g., [6]). The adaptation of these classes of services can be effectively performed taking into account context data such as the ones that describe the network, device capabilities, and categories of users' interests. This category of data – that we call *shallow* context data – can be naturally modeled by means of attribute/value pairs, adopting standard representation formalisms such as CC/PP [12].

On the other hand, mobility claims for the use of a wider set of context data, including complex data such as the user current activity, the set of persons and objects she can interact with, and her surrounding environment. A quite large consensus has been reached in the research community towards the use of expressive languages in order to represent and reason with this data, that we call *ontology-based* context data. Various frameworks for reasoning with this class of data have been recently proposed (e.g., [5, 7]) for applications requiring sophisticated adaptation, like ambient intelligence applications. Since the formalism of choice is typically OWL or some of its variations, the reasoning tasks are known to have high complexity. While some applications may not have strict requirements in terms of efficiency – since generally they serve a limited number of users at a time – the delay introduced by ontological reasoning is less problematic.

In the *CARE* framework we need to model both the above mentioned classes of context data. We have defined an efficient logic programming language for reasoning with shallow context data, while we adopt OWL-DL [10] as the language for representing and reasoning with ontology-based context data. Moreover, in order to have a uniform context representation language, we defined new CC/PP vocabularies to have a mapping between OWL concepts and CC/PP attributes. In *CARE*, policy rules defining how context values can be derived, can contain preconditions involving shallow as well as ontology-based context data that may have to be derived by ontological reasoning. Since efficiency is a main issue, a straightforward solution for avoiding the execution of ontological reasoning at the time of the service request is to perform reasoning asynchronously. However, in particular cases ontological reasoning must be performed at the time of the user request, after having populated the ontology with instances collected from the distributed context sources. This happens, for example, when the reasoning can only be performed after context data has been merged.

In this paper we report the results of some experiments we performed for assessing the feasibility of our approach. In Section 2 we briefly describe the *CARE* middleware, and we illustrate our hybrid reasoning approach with an exam-

*This work has been partially supported by Italian MUR (FIRB "WebMinds" project N. RBNE01WEJT_005).

[1] Context Aggregation and REasoning middleware.

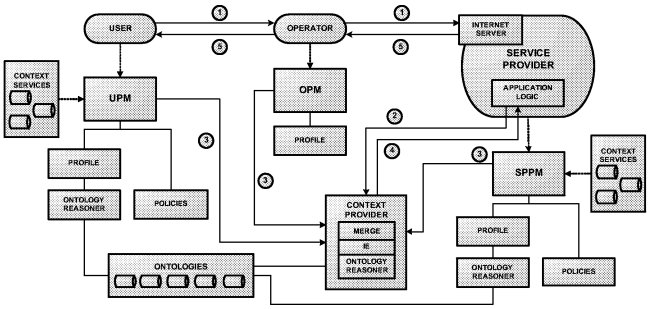

Figure 1. The *CARE* middleware architecture.

ple; In Section 3 we present the experimental results; Section 4 concludes the paper.

2 Hybrid reasoning in *CARE*

The *CARE* middleware and its underlying technical solutions have been presented in [1, 4]. The main components of the *CARE* architecture are shown in Figure 1. In our framework the contextual data, being by nature distributed, is managed by different entities (i.e., the user, the network operator, and the service provider). We call *profile* the subset of context data collected and managed by a certain entity. Each entity has a dedicated profile manager for handling its own context data (called UPM, OPM, and SPPM, respectively). Profiles include both shallow context data and ontology-based context data which is expressed by means of references to ontological classes and relations. Both the user and the service provider can declare policies in the form of rules over context data in order to derive higher-level context data starting from more simple ones, and to determine the adaptation parameters of the service. A dedicated module (called CONTEXT PROVIDER) is in charge of building the aggregated context data for the application logic, evaluating adaptation policies and solving possible conflicts. The architecture includes modules for performing ontological reasoning.

As anticipated in the introduction, we adopt hybrid rule-based/ontological reasoning for modeling context. As a matter of fact, policy rules can contain preconditions involving context data derived through ontological reasoning. In order to illustrate the hybrid mechanism with an example, suppose that a user declared a policy rule asking to set her status to *busy* when involved in a business meeting:

$$\textbf{If } \textit{CurrentActivity} = \textit{`BusinessMeeting'} \quad (1)$$
$$\textbf{then } \textit{Status} = \textit{`Busy'}$$

The semantics of the rule precondition is that the condition holds if the instance *CurrentActivity* (that represents the activity currently performed by the user) belongs to the *BusinessMeeting* ontological concept. Hence, ontological reasoning must be performed before evaluating the rule, in order to check whether the instance *CurrentActivity* belongs to the concept *BusinessMeeting* or not. As an example, consider this definition of the *BusinessMeeting* activity:

$$\textit{BusinessMeeting} \equiv \textit{Activity} \sqcap \geq 2\,\textit{Actor} \sqcap$$
$$\forall\,\textit{Actor.Employee} \sqcap \exists\,\textit{Location.WorkBuilding}$$

Based on this definition, in order to check whether the user is involved in a business meeting it is necessary to have information about the people she is with (possibly derived by the user profile manager analyzing her agenda) and her current location (possibly provided by the network operator). These data must be retrieved by the CONTEXT PROVIDER from the UPM and OPM, respectively, and is added to the assertional part of the ontology (i.e., that part of the ontology – called *ABox* – that contains the individuals of the addressed domain).

We have investigated different approaches for overcoming the computational issues of ontological reasoning. The solution we adopt consists in keeping the terminological part of the ontology (i.e., that part of the ontology – called *TBox* – that contains the definition of classes and relations of the addressed domain) static, in order to be able to perform the TBox classification [3] in advance to the service request. In this way it is possible to save a good amount of computational time while serving user requests, since the ontology classification task is particularly expensive.

Furthermore, the assertional part of the ontology can be filled in advance to the service request with those instances that are known *a priori*, i.e., before retrieving context data from the distributed profile managers. This data obviously depends on the particular domain addressed by the ontology. In the case addressed by our example, the ABox should be populated with a huge number of instances, including those that correspond to the employees of the user organization, and to particular locations (e.g., rooms belonging to the organization). After having populated the ontology with these instances, it is possible to perform the ABox realization [3] in advance to the service request. The realization of the ABox consists in computing, for each individual of the ontology, the most specific classes that it instantiates.

Once again, ABox realization is an expensive reasoning task; hence, it is unsuitable to perform realization at the time of the service request when the ontology contains a huge number of instances. At the time of the user request, the ABox is filled only with those instances that are retrieved from the profile managers and have a mapping to ontological classes and relations. Considering the ontology definition of our example, the instances to be inserted into the ontology correspond to a new activity *currentActivity* – the one performed by the user – and to the relations that

link that activity to its actors and location. Adopting this approach, the only reasoning task that must be performed at the time of the service request is the realization of the single *currentActivity* instance.

3 Experimental evaluation

In order to assess the feasibility of this approach, we performed some experiments on executing ontological reasoning with the OWL-DL ontology we defined for modeling the socio-cultural environment of mobile users presented in [2]. The reasoning task we performed corresponds to the realization of an instance *CurrentActivity* belonging to the *Activity* class.

3.1 Experimental setup

This experimental setup simulates the case in which ontological reasoning is used to derive the specific activity that is currently performed by the user. As explained before, our approach consists in executing ontological reasoning mostly in advance to the service request. For keeping computational times acceptable for interactive services, we perform ontological reasoning at the time of the service request only on small subsets of the whole ontology, populating the ABox with only those instances that are necessary for deriving new context data. For instance, in Experiments A and B we use the subset of our ontology – composed by 12 classes – that is sufficient to define a particular activity we are interested into. Ontological reasoning is performed by the Racer [9] ontology reasoner, on a two-processor Xeon 2.4 GHz workstation with 1.5GB of RAM, using a Linux operating system. Results are calculated as the average of ten runs. Standard deviation is shown in plots.

3.2 Experiment A: Ontological reasoning with an increasing number of instances obtained from the aggregated profile

This experiment aimed at evaluating the feasibility of on-line ontological reasoning with a growing number of instances added to the ABox at the time of the service request. Since these instances are gathered from the aggregated profile, they are not known a-priori. Hence, realization must be performed for assigning these new instances to the classes they belong to.

TBox The TBox consists of the 12 classes that are used to define the *UnimiInternalMeeting* concept.

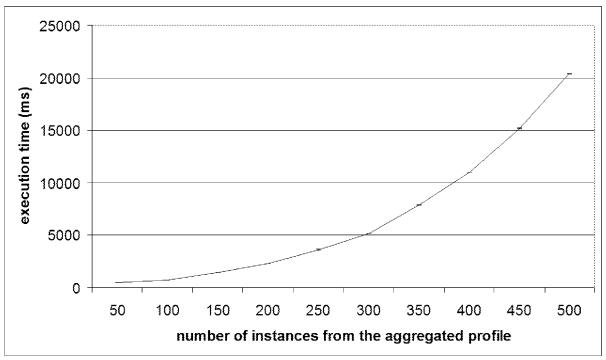

Figure 2. Results of Experiment A

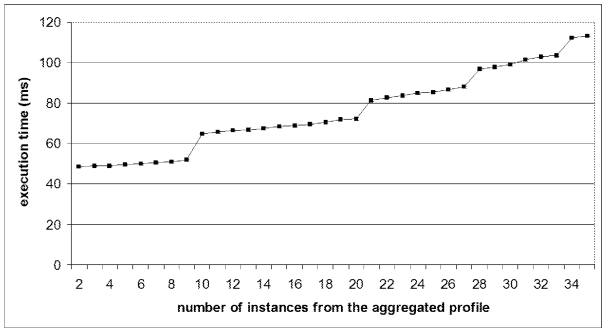

Figure 3. Results of Experiment A with a small number of instances

ABox Before the service request, the ABox contains:

- a variable number k of *UnimiEmployee*s; k corresponds to the values of the x axis of plots in Figures 2 and 3;
- 1 *UnimiBuilding*.

At the time of the service request, the ABox is filled with:

- the *CurrentActivity* instance;
- 1 relation that links *CurrentActivity* to the building in which it is performed;
- k relations that link *CurrentActivity* to its actors.

Results Experimental results are shown in Figures 2 and 3. Execution times grow exponentially with the number k of instances added to the ABox at the time of the service request (see Figure 2). As shown in Figure 3, execution times with this subset of our ontology are acceptable (i.e., less than 100ms) only when the number of instances added at the time of the service request is small.

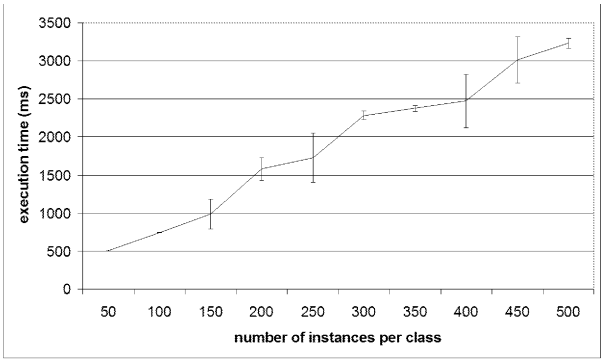

Figure 4. Results of Experiment B: Ontological reasoning with increasing ABox size

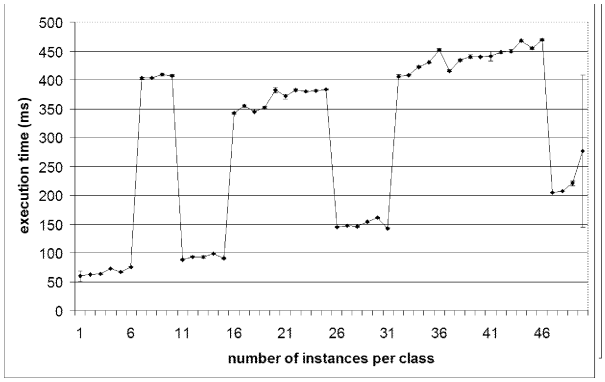

Figure 5. Results of Experiment B with a small number of instances

3.3 Experiment B: Ontological reasoning with an increasing number of instances known *a priori*

This experiment aimed at evaluating the feasibility of on-line ontological reasoning with a growing number of instances added to the ABox *before* the service request. Since these instances are in the ABox before the service request, they can be realized in advance. Hence, in this case the only reasoning task to be performed at the time of the service request is the realization of the *CurrentActivity* instance.

TBox The TBox consists of the 12 classes that are used to define the *UnimiInternalMeeting* concept.

ABox Before the service request, the ABox contains:

- 5 *UnimiEmployee*s;
- 1 *UnimiBuilding*;
- k instances for each one of the remaining 10 classes; k corresponds to the value of the x axis of plots in Figures 4 and 5.

At the time of the service request, the ABox is filled with:

- the *CurrentActivity* instance;
- 1 relation that links *CurrentActivity* to the building in which it is performed;
- 5 relations that link *CurrentActivity* to its actors.

Results Experimental results are shown in Figures 4 and 5. Execution times grow linearly with the number of instances added to the ABox before the service request (see Figure 4). The results obtained when adding a small number of instances exhibit a strange behaviour (see Figure 5), which was confirmed when executing experiments on a different machine. These results are probably due to the internal behaviour of the Racer reasoner. Even if the time of ontological reasoning execution increases with the cardinality of the population of the ABox, performance is not too badly affected.

3.4 Experiment C: Ontological reasoning with a realistic ontology and increasing ABox size

This experiment aimed at evaluating the feasibility of online ontological reasoning with a realistic ontology and an increasing number of instances populating the ABox. In this experiment we tried to reproduce a realistic ontology, which describes more than 100 activities, and other 400 concepts (with a maximum depth of 5 in the subclasses hierarchy). The ABox contains 2000 instances belonging to classes that are not involved in reasoning (i.e., classes that are not subclasses of *Activity*), and a growing number of instances that belong to classes involved in the ontological reasoning. Since these instances are added to the ABox before the service request, their realization can be performed off-line. The only reasoning task to be performed at the time of the service request is the realization of the *CurrentActivity* instance.

TBox The TBox consists of:

- the 12 classes that are used to define the *InternalMeeting* concept;
- other 100 subclasses of *Activity* (similar to the previous ones, but with different cardinality restrictions);

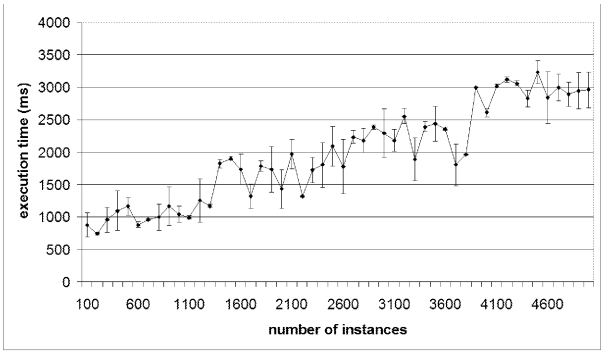

Figure 6. Results of Experiment C: Ontological reasoning with a realistic ontology and increasing ABox size

- 400 classes that are not involved in the definition of *Activity* and its subclasses.

ABox Before the service request, the ABox contains:

- 2000 instances belonging to classes not involved in the reasoning task;
- 5 *Person*s;
- 1 *WorkBuilding*;
- k instances belonging to classes involved in the reasoning task; k corresponds to the values of the x axis of the plot in Figure 6.

At the time of the service request, the ABox is filled with:

- the *CurrentActivity* instance;
- 1 relation that links *CurrentActivity* to the building in which it is performed;
- 5 relations that link *CurrentActivity* to its actors.

Results Experimental results are shown in Figure 6. Execution times grow linearly with the number of instances that belong to classes involved in the reasoning task. When the TBox contains a large number of classes (more than 500 in this setup) and the ABox is filled with a rather large number of instances (more than 2000), ontological reasoning execution time is in the order of seconds, even if few instances belong to classes involved in the reasoning task.

3.5 Remarks

Various performance evaluations and experiments about executing reasoning with ontological languages have been presented before (see, e.g., [8]), which confirm that scalability is a main issue.

With respect to the problem of reasoning with context data, our approach has some similarity with the one proposed by Wang and colleagues in [13]. While they perform experiments with ontologies whose size is comparable with the one used in our experiments, the execution times they obtain grow exponentially with the ontology size. On the contrary, by performing part of the processing in advance to the service request, we can improve scalability, since ontological reasoning execution times grow linearly with respect to the ontology size (see Experiments B and C).

4 Conclusions

In this paper we presented an experimental evaluation of the feasibility of the hybrid reasoning approach adopted by the *CARE* framework. Experimental results with a complex ontology (having more than 500 classes and more than 2000 instances) show that the execution time of reasoning tasks like instance realization is in the order of seconds. As a consequence, ontological reasoning at the time of the service request is unfeasible for most Web applications and services, and should be executed asynchronously with respect to the service requests. On the other hand, ontological reasoning at the time of the service request is feasible when executed on simple ontologies populated by a small number of instances. These small ontologies can be profitably used for executing particular reasoning tasks about a specific class of context data.

References

[1] A. Agostini, C. Bettini, N. Cesa-Bianchi, D. Maggiorini, D. Riboni, M. Ruberl, C. Sala, and D. Vitali. Towards Highly Adaptive Services for Mobile Computing. In *Proceedings of IFIP TC8 Working Conference on Mobile Information Systems (MOBIS)*, pages 121–134. Springer, 2004.

[2] A. Agostini, C. Bettini, and D. Riboni. Loosely Coupling Ontological Reasoning with an Efficient Middleware for Context-awareness. In *Proceedings of the Second Annual International Conference on Mobile and Ubiquitous Systems: Networking and Services (MobiQuitous 2005)*, pages 175–182. IEEE Computer Society, 2005.

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

[4] C. Bettini and D. Riboni. Profile Aggregation and Policy Evaluation for Adaptive Internet Services. In *Proceedings of The First Annual International Conference on Mobile and Ubiquitous Systems: Networking and Services (Mobiquitous)*, pages 290–298. IEEE Computer Society, 2004.

[5] H. Chen, F. Perich, T. W. Finin, and A. Joshi. SOUPA: Standard Ontology for Ubiquitous and Pervasive Applications.

In *Proceedings of the 1st Annual International Conference on Mobile and Ubiquitous Systems (MobiQuitous 2004), Networking and Services*, pages 258–267. IEEE Computer Society, 2004.

[6] B. Grosof. Prioritized Conflict Handling for Logic Programs. In *Proceedings of the International Logic Programming Symposium (ILPS)*, pages 197–211, 1997.

[7] T. Gu, X. H. Wang, H. K. Pung, and D. Q. Zhang. An Ontology-based Context Model in Intelligent Environments. In *Proceedings of Communication Networks and Distributed Systems Modeling and Simulation Conference*, 2004.

[8] Y. Guo, Z. Pan, and J. Heflin. An Evaluation of Knowledge Base Systems for Large OWL Datasets. In *Proceedings of the Third International Semantic Web Conference (ISWC 2004)*, volume 3298 of *Lecture Notes in Computer Science*, pages 274–288. Springer, 2004.

[9] V. Haarslev and R. Möller. RACER System Description. In *Proceedings of Automated Reasoning, First International Joint Conference (IJCAR 2001)*, pages 701–706. Springer, 2001.

[10] I. Horrocks, P. F. Patel-Schneider, and F. van Harmelen. From SHIQ and RDF to OWL: The Making of a Web Ontology Language. *Journal of Web Semantics*, 1(1):7–26, 2003.

[11] R. Hull, B. Kumar, D. Lieuwen, P. Patel-Schneider, A. Sahuguet, S. Varadarajan, and A. Vyas. Enabling Context-Aware and privacy-Conscius User Data Sharing. In *Proceedings of the 2004 IEEE International Conference on Mobile Data Management*, pages 187–198. IEEE Computer Society, 2004.

[12] G. Klyne, F. Reynolds, C. Woodrow, H. Ohto, J. Hjelm, M. H. Butler, and L. Tran. Composite Capability/Preference Profiles (CC/PP): Structure and Vocabularies 1.0. W3C Recommendation, W3C, January 2004. http://www.w3.org/TR/2004/REC-CCPP-struct-vocab-20040115/.

[13] X. H. Wang, T. Gu, D. Q. Zhang, and H. K. Pung. Ontology Based Context Modeling and Reasoning using OWL. In *Proceedings of Second IEEE Annual Conference on Pervasive Computing and Communications Workshops*, pages 18–22. IEEE Computer Society, 2004.