On the Measurement of Semantic Reasoners in Ambient Assisted Living Environments

David Ausín, Federico Castanedo and Diego López-de-Ipiña
Deusto Institute of Technology, DeustoTech. University of Deusto.
Avda. Universidades 24. 48007 Bilbao, Spain.
{david.ausin, fcastanedo, dipina}@deusto.es
http://www.morelab.deusto.es/

***Abstract*—The increasing lifetime and population of elderly people leads to a great amount of interest in Ambient Assisted Living (AAL) environments and applications. An AAL environment could be modeled by ontologies and using a semantic reasoner as a way to infer knowledge about the underlying context. An important question is to clarify the feasibility of employing a semantic reasoner in AAL applications. For this reason, in this work we present our results of a simulated AAL environment using a knowledge driven approach. Our aim is to determine and clarify which combination of semantic reasoners and frameworks is the most convenient for this task. In particular, we are interested in the combination which provides a better computational performance and reasonable memory requirements. The obtained results show that the fastest choice under the employed dataset is to use the OWL API accessing the Pellet reasoner. On other hand, the less used and committed memory consumption experiment is provided by a combination of HermiT and custom Java rules.**

***Keywords*—Semantic reasoners, ontologies, AAL**

I. Introduction

It has been estimated that in the year 2025, 22.7 percent of the EU25 countries total population will be older than 65 years and this number will rise up to 30.3 percent in 2050. Intelligent environments located in user's homes provide a way of maintaining and increasing personal autonomy of elderly and disabled persons. A key component of an intelligent environment is how the user behavior is modeled and reasoned over time in order to assist the user in his activities of daily living (ADL) or execute the corresponding reactions. There are several ways to achieve this goal, ranging from statistics to logics and rule-based engines. Generally speaking, there are two broad different approaches for building an intelligent environment or activity recognition system. On one hand, the *data driven* approach focuses on inferring the activities from the acquired data without using any a priori knowledge model (that is without explicit knowledge). This approach is based on statistics and machine learning and their quality usually depends on the obtained data. The common way to deal with this approach is to have training data so that the algorithms iteratively learn and fit the model to the underlying data. This approach works well in several application domains but presents some drawbacks when it is necessary to have a greater expressiveness in the environment being modeled.

On other hand, the *knowledge driven* approach relies on logic based methods. Well-known Description Logics (DL) provides high expressiveness and when combined with appropriate computational properties are the underlying theory for ontologies. The main drawbacks of this approach are: (i) it requires a priori knowledge engineering task about the world model representation; (ii) their difficulty in handling uncertainty due to the violation of some properties that produce undecidable logics [1].

In this work, we focus on the knowledge driven approach, therefore an ambient assisted living environment is modeled using ontologies and different semantic reasoners are applied to a simulated stereotypical scenario obtaining corresponding benchmarking results. The aim of this work is to determine and clarify which combination of reasoners and framework is the most suitable for building knowledge-driven smart environments. In particular, we are interested in the combination which provides the best computational performance and reasonable memory requirements. Experiments are carried out using a modified version of the dataset published in [2]. The structure of this article is as follows. The next section provides an overview of related work in the literature. Then section III gives a brief description of the employed reasoners and semantic frameworks. Section IV describes the work done to simulate an ambient assisted living environment and the performed benchmark experiments. Finally, section V summarizes this work and points future work.

II. Related works

A Knowledge Base (KB) consists on two components: (i) the *TBox* component introduces the terminology (i.e. the vocabulary of the application domain) while the (ii) *ABox* contains assertions about named individuals in terms of this vocabulary. Benchmarks results with the goal of timing the classification of each ontology by each reasoner were presented in [3]. Since they focused on testing TBox reasoning (classification) and each reasoner may perform tasks in different order, they divided the tests into 5 different steps and timed each of them.

Other benchmarks are focused mainly in ABox reasoning. This is the case of Lehigh University Benchmark (LUBM) [4] which defines a procedure to compare Semantic Web Knowledge Base systems. However, there are authors [5] who claim that these benchmarks are not enough to help a developer in selecting a reasoner and suggest a set of

general benchmarking requirements for new OWL reasoner benchmarkings.

In [6], authors propose an OSGi-based middleware platform enhanced with semantic context modeling and reasoning for intelligent environments. This platform integrates sensing, reasoning and actuation over heterogeneous equipment in an intelligent environment such as an ambient assisted living scenario. Instead of doing a benchmarking experiment, their aim was to evaluate the context model and their behavior over time. For a good survey on context modeling and reasoning techniques in pervasive computing, we refer the reader to [7].

Several ontologies have been proposed to represent complex human activities in ambient intelligence environments, such as SOUPA [8], CONON [9], CARE [10]. In [11], an ontology for smart homes is presented, data coming from sensors are mapped to ontological classes and properties and then added to the assertional part of the ontology. In [12], authors present an OWL2 software architecture to perform activity recognition in AAL environments. They present several experiments that evaluate the scalability in terms of computational costs with growing number of users (from 1 to 20) and sensors (from 20 to 120) using Pellet and HermiT. In contrast, in this work we focus on the computational performance of different semantic reasoners under similar conditions, that is using the same dataset.

In recent years, there has been a growing interest in combining *data driven* and *knowledge driven* approaches, such as [13] [14]. One issue that must be handled carefully in these kind of hybrid approaches is the combination of the open world assumption (ontologies) with the closed world (ie. rule-based languages). In the open world, no conclusion can be drawn from unknown assertions. This means that only the assertions declared in the KB are true. In contrast to open world, assertions which are not explicitly declared in the closed world are false.

III. Employed semantic frameworks and reasoners

In this section the employed semantic frameworks and several reasoners are described.

A. Semantic frameworks

A framework is a set of software libraries providing an abstraction functionality exposed by an Application Programming Interface (API). The main advantage of a semantic framework for developers is the common interface provided to interact with ontologies and reasoners. The most employed semantic frameworks nowadays are Jena, OWL-API and OWLlink API.

1) Jena: is a Java framework for developing semantic applications [15]. The current version of Jena (v2.8.8) supports Resource Description Framework (RDF) [16], RDF Schema (RDFS) [17] and OWL [18]. It also provides a rule-based inference engine and a SPARQL [19] query engine. Jena supports the following input/output formats: (i) RDF in RDF/XML, (ii) N3 and (iii) N-Triples.

One of the purpose of Jena's architecture is to provide an easy management of data in graphs and a straightforward view of the RDF graph in order to expose data as triples. The RDF graph is the basic component of Jena architecture. A graph is composed by triples and each triple is composed by nodes which can be a URI label, a blank node or a literal.

Jena architecture is composed by the following three layers:

1) *Graph layer*, which is based on the RDF Abstract Syntax. Jena includes several implementations of this layer providing different triple stores and some built-in inference mechanism.
2) *EnhGraph layer*, consists in the extension point on which to build APIs. Its design provides multiple views of graphs and nodes which can be used simultaneously.
3) *Model layer*, which offers methods to manipulate the graph and its nodes to the application *Resource* interfaces.

Not only does Jena include its own inference engines as is shown in subsection III-B, but also supports engines from other reasoners.

2) OWL API: was born from an attempt to provide a highly reusable component for the construction of different OWL based applications [20]. It provides a Java API and reference implementation for creating, manipulating and serializing OWL ontologies.

The current version of OWL API (v3) is aligned with the OWL 2 specification [21]. It supports parsing and printing ontologies in several formats, such as RDF/XML, OWL Functional Syntax or Turtle. The OWL API is completely based on OWL 2 Structural Specification. In contrast, other APIs (i.e. Jena) are based on RDF triples.

To perform OWL reasoning, the OWL API provides the OWLReasoner interface which must be implemented by the reasoners to allow access via OWL API. For instance, this interface is implemented by Fact++ [22], HermiT [23], Pellet [24], RacerPro [25] and CEL [26].

3) OWLlink: [27] is a reasoning protocol for exchanging information among OWL 2-aware applications and has been inspired by DIG [28] and KRSS [29] protocols. OWLlink offers a flexible straightforward way to communicate applications with reasoners. Due to this fact, the reasoning task can be delegated to a different computer using a distributed computational scheme.

OWLlink is composed by a protocol core and a set of bindings. On the one hand, the core is able to query the reasoning engine for its properties and settings, which can be modified by it. Besides, the core also offers primitives to manage the KB and the way the reasoner is accessed. On the other hand, bindings establish the transport protocol and the content serialization. Although custom bindings can be developed, the OWLlink specification defines three bindings: (i) XML over HTTP (the primary binding), (ii) Functional Syntax over HTTP and (iii) S-Expression over HTTP.

OWLlink has been implemented by several frameworks like OWLlink API or OntoLisp and reasoners such as RacerPro [30].

B. Semantic reasoners

Semantic reasoners are a key component in an intelligent semantic system. This component infers logical consequences from a set of asserted facts, usually stored in the knowledge base. In this work we briefly summarize the following reasoners: (1) Jena, (2) HermiT and (3) Pellet.

1) Jena reasoners: Apart from being a semantic framework, Jena default distribution includes several inference engines. According to Jena documentation, it includes reasoners for RDFS, OWL, DAML and a transitive reasoner.

In addition, Jena also includes a general purpose rule-based reasoner. This reasoner is the base of the RDFS and OWL reasoners and contains two internals rule engines: a forward chaining RETE [31] engine and a tabled datalog engine. It is possible to use them separately or as an hybrid rule engine via the provided class implementation: *GenericRuleReasoner*. In the hybrid rule engine both rule engines cooperate to answer a query. Firstly, the forward engine (RETE) processes the data and then the results are supplied to the backward engine (tabled datalog engine) which finally answers the query.

A set of built-in primitives are provided with Jena and can be called form the rules (Jena rules). Moreover, new built-ins can be created and registered to enrich the Jena rules constructions. Primitives can be used in the rule head, the rule body (list of antecedents) or both. Primitives used in the rule body act as a condition which must be fulfilled in order to execute the rule head. However, primitives used in the rule head are only used for their side effects. Custom inference rules can be combined with some RDFS and OWL inference, but with some limitations. The way to achieve this is using reasoners in cascade or loading a combination of RDFS / OWL rules and custom rules into a *GenericRuleReasoner*.

2) HermiT: [23] is a description reasoner for $\mathcal{SROIQ}$ distributed under the terms of LGPL which can be accessed through OWLlink or OWL API. The main difference against other reasoners is that the reasoning algorithm employed, which is based on the hypertableau calculus algorithm (a complete calculus for first order clausal logic) [32].

HermiT supports DL Safe SWRL rules, but it does not include built-in implementations. Therefore, rules that call built-in atoms can not be executed under HermiT. Besides, it has the problem that property chains, transitivity axioms and complex properties in the body of the rules produces incomplete reasoning.

3) Pellet: [24] is an OWL 2 reasoner for Java under dual license terms: AGPLv3 for open source projects and a special one for closed source applications. Pellet can be accessed through Jena, OWL API or OWLlink.

It uses a a tableaux algorithm which is able to handle $\mathcal{SROIQ}(D)$ logic and includes several optimizations such as back-jumping, simplification or absorption. Pellet has also techniques for incremental reasoning, such as incremental consistency checking and incremental classification.

In addition, Pellet supports SPARQL, but only to query ABox. In order to query TBox, Pellet can be used with Jena.

In this case, non ABox queries will be delegated to Jena. Besides, it implements DL-Safe SWRL rules support. All the built-ins for SWRL rules described in [33] are provided, with the exceptions of built-ins for *List* and some built-ins for date, time and duration. Anyway, custom built-ins can be developed and registered into the reasoner in order to be called from SWRL rules.

IV. Benchmark experiments and obtained results

The goal of the experiments is to measure the performance in pervasive environments of various semantic reasoners combined with different semantic frameworks. To achieve this, a simulated scenario in which an elderly person (e.g. Abraham Simpson), who needs help to remember when he must take his medicines, has been modeled. Two types of medicament doses are taken into account: (i) periodic treatments and (ii) meal treatments.

On one hand, meal treatments are the treatments in which drugs have to be taken after, before or during a meal (breakfast, lunch or dinner). On the other hand, periodic treatments are treatment in which drug must be taken with a frequency of *n* hours.

A. Developed ontology and rules

The employed ontology (the same for all the experiments) are a set of the TALISMAN+ ontologies, an OWL2 DL ontology. The TALISMAN+ core ontology is the main ontology of the TALISMAN+ project [34]. A core and extension ontology has been developed. The extension has also been undertaken to support the activities described in the employed dataset and contains the necessary individuals for the experiments and their relationships. The following are the most important owl classes defined:

- *DisabledPerson*. It models a person who needs care giving.
- *MedicalTreatment*. It represents a drug and its prescription.
- Subclasses of *Activity*. They describe activities. The most important for this scenario are the subclasses of *HavingMainFood*, the subclasses of *TakingDrug*, *PickingUpTableFood* and *PreparingFood*.

To describe that a *DisabledPerson* is taking a *MedicalTreatment* the *medicalTreatmet* object property is used and to specify what is his current activity the *isDoing* object property is employed.

In addition to the ontology, a set of rules has been created. These rules check if the user has taken the corresponding medicament, and advises him when he forgets to take it or when he takes the wrong one. These rules do not try to cover all the possible situations that can be produced, but do at least, the most important cases. Rules have been implemented using SWRL rules and Jena rules format. A custom built-in to convert Unix time to the number of seconds in a day has been developed.

As rules execution are only available out-of-the-box in Pellet and Jena reasoners, custom Java rules have been developed

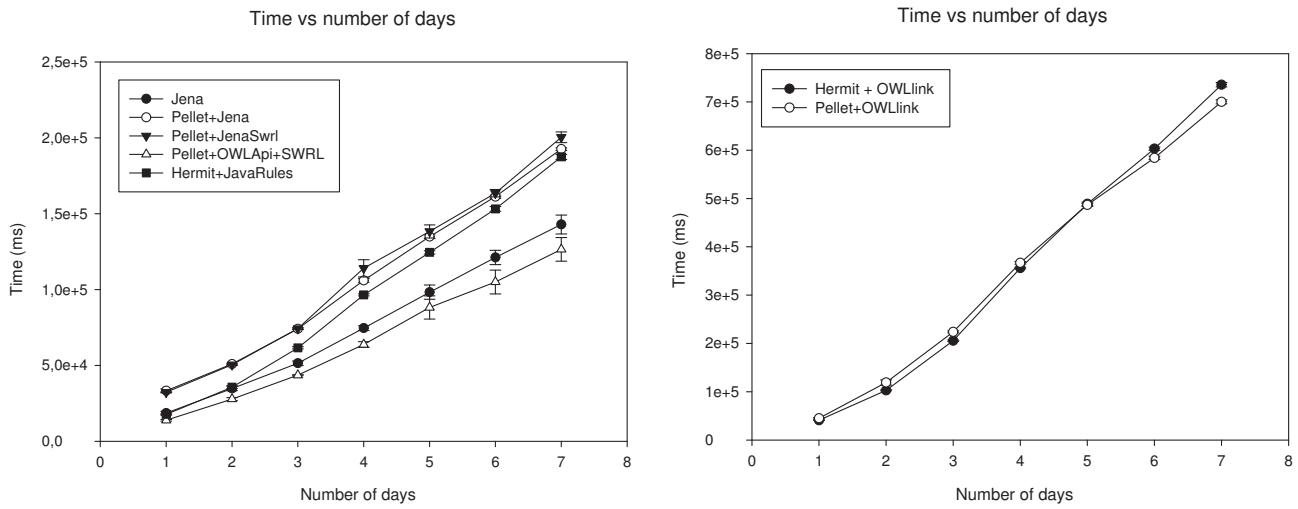

Fig. 1. Obtained results of reasoners execution time as we increase the number of days. The left graph shows the results of the reasoners used as libraries whereas the one on the right side shows the results of OWLlink protocol access to reasoners (Client/Server approach).

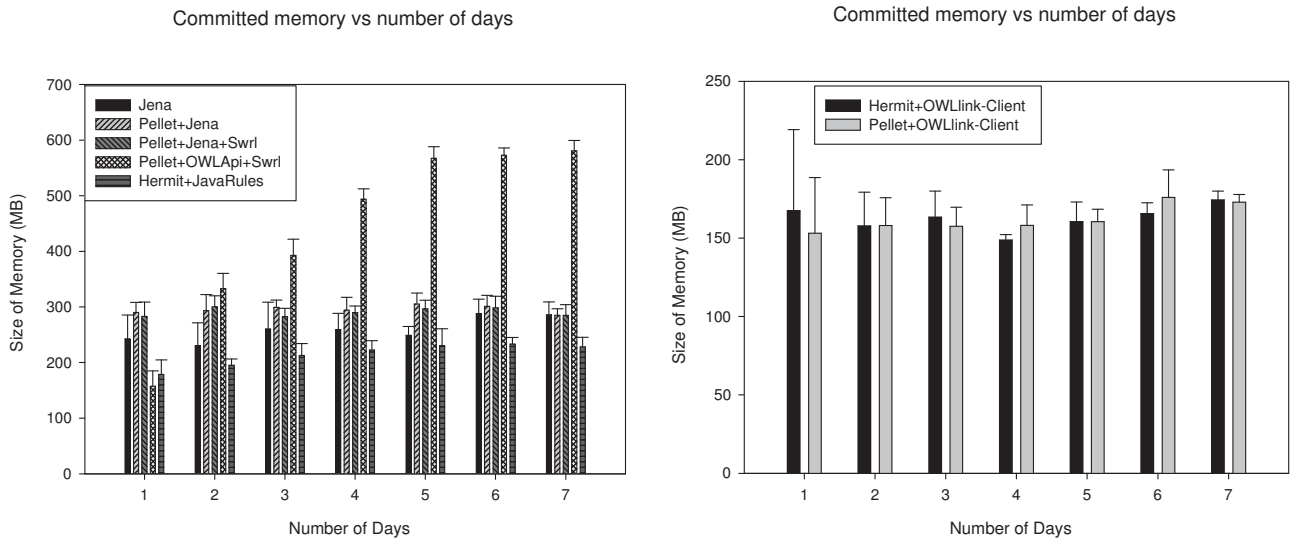

Fig. 2. Committed memory results when we increase the number of days, i.e. the data grows. Commited memory is the total amount of memory allocated for use by the JVM. The left graph shows the results of the reasoners libraries whereas the one on the right side shows the results of OWLlink protocol access to reasoners (Client side memory).

for HermiT and reasoners used them via OWLlink protocol. These Java rules aim to provide the same inference results as SWRL and Jena rules.

B. Employed dataset

As has been previously mentioned, a modified version of a dataset created by [2] has been employed. Not only does our version add new activities in order to describe that a user has taken a drug, it also simplifies the dataset using only three fields: (i) activity, (ii) routine and (iii) time. This dataset is composed of seven days of user activities. We refer the reader to [2] for more information about this dataset.

C. Experiments

The experiments have been carried out using an Acer Aspire 4830TG laptop whose main features are an Intel Core i5-2410M and 8 GB of RAM. The operating system is Ubuntu 11.10 for 64 bits, a distribution of GNU/Linux, which runs the Java 1.6.0_23 version provided by the OpenJDK Runtime Environment (IcedTea6 1.11pre) is employed.

The program firstly loads the ontologies into a reasoner. Then, it reads the simulated activities that the user is supposed to be doing from the dataset and updates the user context. Frameworks and reasoners combinations are summarized in the table I. Each combination has been executed ten times and the data presented below shows the average and the standard deviation. Figure 1 shows the time needed by the reasoners

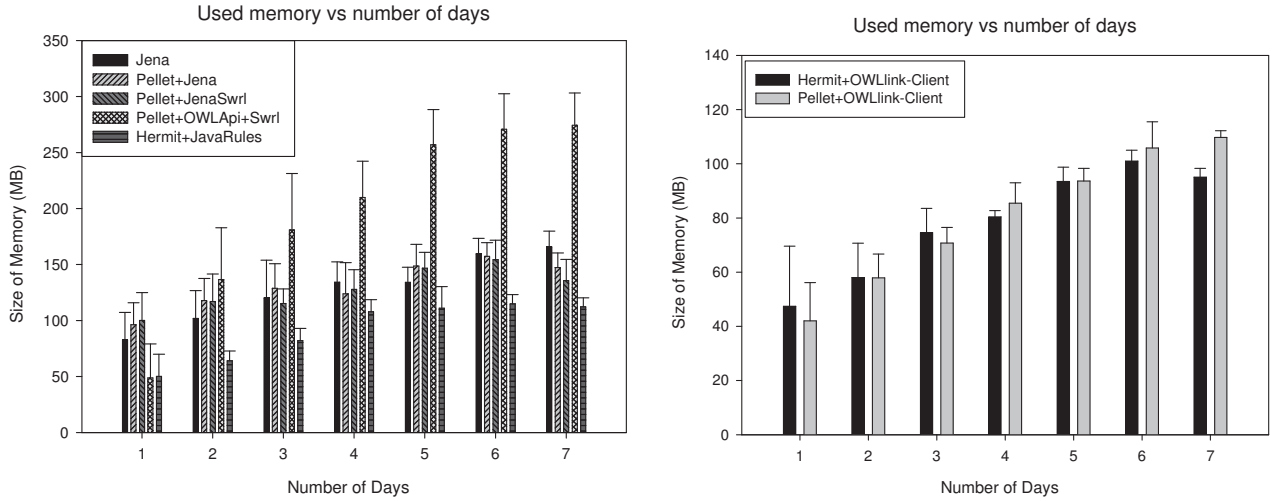

Fig. 3. Used memory results when we increase the number of days, i.e. the data grows. Memory used includes the memory occupied by all objects including both reachable and unreachable objects. The left graph shows the results of the reasoners libraries whereas the one on the right side shows the results of OWLlink protocol access to reasoners (Client side memory).

whereas figure 3 the memory consumption.

	OWLlink API 1.2.2	*OWL API 3.1.0*	*Jena 2.6.4*
Jena 2.6.4	-	-	(1) Jena rules
Hermit 1.3.5	(2) Java rules	(3) Java rules	-
Pellet 2.3.0	(4) Java rules	(5) SWRL rules	(6) Jena and (7) SWRL rules

TABLE I
DIFFERENT COMBINATIONS OF REASONERS AND SEMANTIC FRAMEWORKS USED IN THE BENCHMARK EXPERIMENTS AND THE KIND OF RULES EMPLOYED.

As figure 1 shows the fastest choice is to use the OWL API to access to the Pellet reasoner. This result is pretty meaningful, because this approach is a 12% better than the second fastest configuration. However, the fastest configuration has also the biggest memory consumption, as figure 3 shows. In general, these figures do not show a relationship between the necessary time to perform a task and the memory consumption. Another important issue to remark is that in general the reasoner used in combination with the Jena framework (Pellet+Jena, Pellet+JenaSwrl in figure 1 left) is slower than others. However the time performance is improved when the reasoner provided by Jena is employed (Jena in figure 1). This issue could be the result of a more lightweight reasoning support for OWL in Jena reasoners. We also measure the amount of used and committed memory for the different combination. Used memory includes the memory occupied by all objects including both reachable and unreachable objects. Committed memory is the amount of memory guaranteed to be available for use by the JVM. The used memory results (see figure 3 left) show that the best memory consumption combination is to use HermiT+JavaRules through OWL API (case 3 of table I), which is also valid for the committed memory (see figure 2). In the particular case of Pellet+OWLApi+SWRL (case 5 of table I) we can notice a inverse relation between the memory consumption and the execution time, since it has the smallest execution time and the highest memory consumption. In the case of using the OWLlink protocol to access reasoners capabilities, a high increment of execution time is noticed (see figure 1 right). This is because of the required marshalling/unmarshalling of the data and the added abstraction layer. The obtained results of memory consumption shows an equal amount of memory required in the client side (see figure 3 right) as we expected.

V. CONCLUSIONS AND FUTURE WORK

In this work we have measured the performance of several semantic reasoners combined with semantic frameworks in a simulated AAL scenario. The results show that the performance of a reasoner can vary meaningfully depending on the semantic framework used. Regarding memory consumption,

	Jena	*HermiT*	*Pellet*
Description logic support	It depends on reasoner	$\mathcal{SROIQ}$	$\mathcal{SROIQ}$
OWL API	No	Yes	Yes
OWLlink	No	Using OWLlink API	Using OWLlink API
Jena framework	Yes	No	Yes
Rule support	Jena rules	SWRL without built-ins	SWRL,some built-ins are not provided

TABLE II
COMPARISON OF REASONERS AND THEIR CAPABILITIES.

we have noticed that the combination of HermiT and Java Rules (see Figure 3 and 2 left) provides the best memory consumption. On the other hand, Pellet+OWLApiSRWL provide the worst one as can be shown in the previous figures. We suppose that this effect is due to the fact that Pellet implementation needs to transform the internal data from OWLApi

format to the internal Pellet representation. In contrast, we suppose that HermiT provides better memory consumption results because it internally employs OWLApi format.

We have noticed that in the case of Jena framework, committed memory results provide a more stable consumption while the number of days are increased.

One particular issue that should be considered for AAL scenarios is that OWL 2 does not provide native support to handle uncertainty about the data being modeled. We believe there is a huge room to improve current semantic reasoners through reasoning support under uncertainty and there are few works covering this topic [1].

In the future, we would like evaluate the same scenario using ontologies described in different OWL2 profiles, such as OWL2 EL. An interesting future work is to develop a specific benchmark for AAL applications where different semantic reasoners could be tested.

ACKNOWLEDGEMENTS

This work is supported by the Spanish MICINN project TALIS+ENGINE (TIN2010-20510-C04-03).

REFERENCES

[1] T. Lukasiewicz and U. Straccia, "Managing uncertainty and vagueness in description logics for the semantic web," *Web Semantics: Science, Services and Agents on the World Wide Web*, vol. 6, no. 4, pp. 291–308, 2008.

[2] T. Huynh, M. Fritz, and B. Schiele, "Discovery of activity patterns using topic models," in *Proceedings of the 10th international conference on Ubiquitous computing*. ACM, 2008, pp. 10–19.

[3] T. Gardiner, I. Horrocks, and D. Tsarkov, "Automated benchmarking of description logic reasoners," *Proc. of DL*, vol. 189, 2006.

[4] Y. Guo, Z. Pan, and J. Heflin, "Lubm: A benchmark for owl knowledge base systems," *Web Semantics: Science, Services and Agents on the World Wide Web*, vol. 3, no. 2-3, pp. 158–182, 2005.

[5] T. Weithöner, T. Liebig, M. Luther, and S. Böhm, "Whats wrong with owl benchmarks," in *Proc. of the Second Int. Workshop on Scalable Semantic Web Knowledge Base Systems (SSWS 2006)*. Citeseer, 2006, pp. 101–114.

[6] D. López-de Ipiña, A. Almeida, U. Aguilera, I. Larizgoitia, X. Laiseca, P. Orduña, A. Barbier, and J. Vazquez, "Dynamic discovery and semantic reasoning for next generation intelligent environments," in *2008 IET 4th International Conference on Intelligent Environments*. IET, 2008, pp. 1–10.

[7] C. Bettini, O. Brdiczka, K. Henricksen, J. Indulska, D. Nicklas, A. Ranganathan, and D. Riboni, "A survey of context modelling and reasoning techniques," *Pervasive and Mobile Computing*, vol. 6, no. 2, pp. 161–180, 2010.

[8] H. Chen, T. Finin, and A. Joshi, "An ontology for context-aware pervasive computing environments," *The Knowledge Engineering Review*, vol. 18, no. 03, pp. 197–207, 2003.

[9] T. Gu, X. Wang, H. Pung, and D. Zhang, "An ontology-based context model in intelligent environments," in *Proceedings of Communication Networks and Distributed Systems Modeling and Simulation Conference*, vol. 2004. Citeseer, 2004, pp. 270–275.

[10] A. Agostini, C. Bettini, and D. Riboni, "Hybrid reasoning in the care middleware for context awareness," *International journal of Web engineering and technology*, vol. 5, no. 1, pp. 3–23, 2009.

[11] L. Chen, C. Nugent, M. Mulvenna, D. Finlay, and X. Hong, "Semantic smart homes: towards knowledge rich assisted living environments," *Intelligent Patient Management*, pp. 279–296, 2009.

[12] D. Riboni and C. Bettini, "Owl 2 modeling and reasoning with complex human activities," *Pervasive and Mobile Computing*, 2011.

[13] X. Hong, C. Nugent, M. Mulvenna, S. McClean, B. Scotney, and S. Devlin, "Evidential fusion of sensor data for activity recognition in smart homes," *Pervasive and Mobile Computing*, vol. 5, no. 3, pp. 236–252, 2009.

[14] D. Riboni and C. Bettini, "Cosar: hybrid reasoning for context-aware activity recognition," *Personal and Ubiquitous Computing*, vol. 15, no. 3, pp. 271–289, 2011.

[15] J. Carroll, I. Dickinson, C. Dollin, D. Reynolds, A. Seaborne, and K. Wilkinson, "Jena: implementing the semantic web recommendations," in *Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters*. ACM, 2004, pp. 74–83.

[16] J. J. Carroll and G. Klyne, "Resource description framework (RDF): Concepts and abstract syntax," W3C, W3C Recommendation, Feb. 2004, http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/.

[17] R. V. Guha and D. Brickley, "RDF vocabulary description language 1.0: RDF schema," W3C, W3C Recommendation, Feb. 2004, http://www.w3.org/TR/2004/REC-rdf-schema-20040210/.

[18] W3C, "OWL 2 web ontology language document overview," W3C, Tech. Rep., 2009, http://www.w3.org/TR/2009/REC-owl2-overview-20091027/.

[19] E. Prud'hommeaux and A. Seaborne, "SPARQL query language for RDF," W3C, W3C Recommendation, 2008, http://www.w3.org/TR/2008/REC-rdf-sparql-query-20080115.

[20] S. Bechhofer, R. Volz, and P. Lord, "Cooking the semantic web with the owl api," *The Semantic Web-ISWC 2003*, pp. 659–675, 2003.

[21] M. Horridge and S. Bechhofer, "The owl api: a java api for working with owl 2 ontologies," in *Proc. OWLED 2009 Workshop on OWL: Experiences and Directions. CEUR Workshop Proceedings*, vol. 529, 2009.

[22] D. Tsarkov and I. Horrocks, "Fact++ description logic reasoner: System description," *Automated Reasoning*, pp. 292–297, 2006.

[23] R. Shearer, B. Motik, and I. Horrocks, "Hermit: A highly-efficient owl reasoner," in *Proceedings of the 5th International Workshop on OWL: Experiences and Directions (OWLED 2008)*, 2008, pp. 26–27.

[24] E. Sirin, B. Parsia, B. Grau, A. Kalyanpur, and Y. Katz, "Pellet: A practical owl-dl reasoner," *Web Semantics: science, services and agents on the World Wide Web*, vol. 5, no. 2, pp. 51–53, 2007.

[25] V. Haarslev, K. Hidde, R. Möller, and M. Wessel, "The racerpro knowledge representation and reasoning system," *Semantic Web*, 2011.

[26] F. Baader, C. Lutz, and B. Suntisrivaraporn, "Cel— a polynomial-time reasoner for life science ontologies," in *Proceedings of the 3rd International Joint Conference on Automated Reasoning (IJCAR'06)*, ser. Lecture Notes in Artificial Intelligence, U. Furbach and N. Shankar, Eds., vol. 4130. Springer-Verlag, 2006, pp. 287–291.

[27] T. Liebig, M. Luther, O. Noppens, and M. Wessel, "Owllink," *Semantic Web – Interoperability, Usability, Applicability*, vol. 2, no. 1, pp. 23–32, 2011.

[28] S. Bechhofer and R. Möller, "The dig description logic interface: Dig/1.1," in *Proceedings of the 2003 Description Logic Workshop (DL 2003)*, 2003.

[29] P. Patel-Schneider and B. Swartout, "Description-logic knowledge representation system specification from the krss group of the arpa knowledge sharing effort," *KRSS group of the ARPA*, 1993.

[30] RacerPro, "Racerpro users guide and reference manual version 1.9.2," 2007.

[31] C. Forgy, "Rete: A fast algorithm for the many pattern/many object pattern match problem," *Artificial intelligence*, vol. 19, no. 1, pp. 17–37, 1982.

[32] P. Baumgartner, U. Furbach, and I. Niemelä, "Hyper tableaux," *Logics in Artificial Intelligence*, pp. 1–17, 1996.

[33] I. Horrocks, P. Patel-Schneider, H. Boley, S. Tabet, B. Grosof, and M. Dean, "Swrl: A semantic web rule language combining owl and ruleml, 2004," *W3C Submission*, 2006.

[34] D. Ausín, D. López-de Ipiña, J. Bravo, M. Valero, and F. Flórez, "Talisman+: Intelligent system for follow-up and promotion of personal autonomy," *Ambient Assisted Living*, pp. 187–191, 2011.