Semantic Reasoning in Context-Aware Assistive Environments to Support Ageing with Dementia

Thibaut Tiberghien[1,2,3], Hamdi Aloulou[1,2,3], Mounir Mokhtari[1,2], and Jit Biswas[4]

[1] CNRS, Image & Pervasive Access Lab (IPAL UMI), Singapore
{thibaut.tiberghien, hamdi.aloulou, mounir.mokhtari}@ipal.cnrs.fr
[2] Institut Mines Télécom, France
[3] Université Pierre & Marie Curie, France
[4] Institute for Infocomm Research (I^{2}R/A*STAR), Singapore
biswas@i2r.a-star.edu.sg

Abstract. Robust solutions for ambient assisted living are numerous, yet predominantly specific in their scope of usability. In this paper, we describe the potential contribution of semantic web technologies to building more versatile solutions — a step towards adaptable context-aware engines and simplified deployments. Our conception and deployment work in hindsight, we highlight some implementation challenges and requirements for semantic web tools that would help to ease the development of context-aware services and thus generalize real-life deployment of semantically driven assistive technologies. We also compare available tools with regard to these requirements and validate our choices by providing some results from a real-life deployment.

Keywords: Ambient Assisted Living, Context Awareness, Knowledge Modelling, Semantic Web, Inference Engine

1 Introduction

Ambient Assisted Living (AAL) consists of a set of ubiquitous technologies embedded in a living space to provide pervasive access to context-aware assistive services. It can for example enhance ageing in place by helping elderly people with their Activities of Daily Living (ADL). The available solutions in this field are numerous and, in most cases, robust. However, their scope of usability, mostly focused on security aspects, is generally very narrow [7, 15]. To help the generalization of such systems, it would be useful to integrate them in an interoperable way. This would decrease their cost by sharing hardware or even software resources. Leveraging the system in place, we could then provide other context-aware services like reminders or ADL assistance at a lower cost and start to tackle the Quality of Life (QoL) aspects [12]. The novelty of our approach lies in the complete redesign of the semantic reasoning engine, able to adapt to people with unpredictable behaviours and evolving needs. This engine aims at providing real-time assistive services in a context-aware manner.

In Sect. 2, we present the potential use of semantic web technologies to drive the interoperability of the system. In a nutshell, semantic descriptions can be used to separate application logic from underlying models in order to avoid writing application specific code [11]. The numerous semantic web tools available have very disparate characteristics and performances. Moreover, benchmarks for such tools have limitations and a more qualitative observation on the requirements is needed to give useful hints to developers [19]. As explained by Luther et al. [11], "choosing the appropriate combination of a reasoning engine, a communication interface and expressivity of the utilized ontology is an underestimated complex and time consuming task". We spent the last year putting in place an appropriate test-case in order to give useful hints to AAL researchers and developers. Sect. 3 describes our conditions on reasoners and ontology/rules formalization to be efficiently integrated in AAL systems. Finally, Sect. 4 provides a comparison of some reasoners with regard to the suggested requirements and some results from our validation process through real-life deployment. The authors recommend to readers who are unfamiliar with AAL systems to read first the description of our prototype in Sect. 4.3 in order to get a good idea of the systems described in the coming sections.

2 Contributions of Semantic Web Technologies to AAL

The Internet of Things (IoT) describes a world where machines and physical objects are seamlessly integrated into the information network, and communicate together to exchange and process information. Tim Berners-Lee's Linked Data [2] is possibly a syntax for this exchange that encloses semantic modelling and annotation in its heart, thus improving the run-time adaptability of the communication. The powerful combination of IoT and Linked Data drives pervasive computing away from predefined bindings and static communication protocols. AAL is only an application-domain of this combination, whose specificities are being analysed here. Semantic technologies are used to perform context-aware service provision in smart environments, and have a multi-faceted role in the platform. Indeed, we referenced four main purpose to using these technologies in our use-case: 1. the modelling of assistance in smart spaces, including non-predictable behaviours, 2. the integration of all entities of the system, with an environment discovery and configuration mechanism, 3. the collaboration between modules of the system based on a shared model and lexical, 4. the reasoning to create the system's intelligence, based on the three previous points. Of course, this paper does not cover all these aspects and if a general introduction is given in this section, it will later focus only on reasoning.

2.1 Enhancing Modularity & Flexibility

To build AAL spaces or smart spaces in general, one must integrate a line-up of entities: sensor network, reasoning engine, environment actuators, interactive devices and services. By enhancing the modularity and flexibility of the system,

we could go towards a larger scale deployability without decreasing the customizability of solutions. The Service Oriented Architecture (SOA) has a beneficial contribution [10] as it provides mechanisms for the deployment and maintenance of entities as well as for the communication between them. We have augmented it with a SOA-based discovery protocol and the automatic generation of bundles (SOA software resources) in order to add a plug & play support for sensors, actuators and devices [1]. However, this only puts in place a *mechanical* plug & play where entities discover each other and start exchanging data. Entities actually do not know about each other's bindings with the environment. E.g. where has this motion detector been deployed? Who is carrying this handphone? Being able to parse data received from a new unknown entity is not enough; you need to be given its semantic. We have imagined, and described in a previous publication [1], a *semantic* plug & play where entities — services, sensors, actuators or devices — provide their semantic profile when "shaking hands" with the platform. This profile can be edited during the development, the deployment, or updated at run-time by users or even other entities. Doing so, a real plug & play behaviour is created where entities are able to genuinely *understand* each other to collaborate. Our solution is represented in Fig. 1.

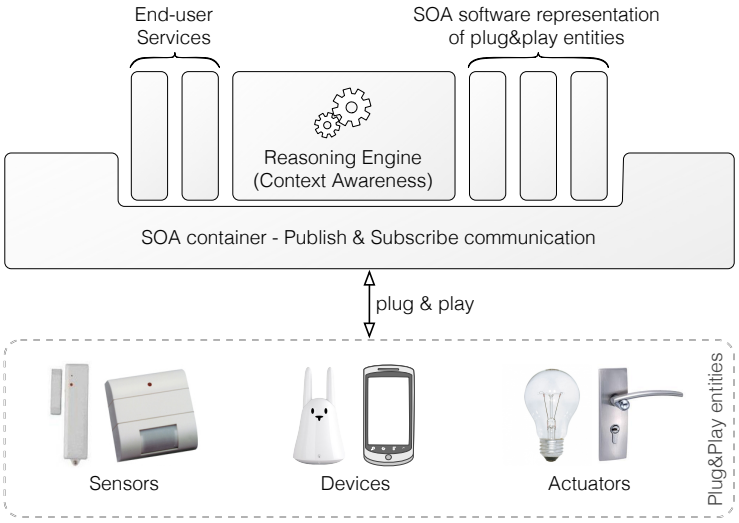

Fig. 1. High-level representation of our context-aware service platform

In the literature, pervasive systems often utilize a layer providing a level of abstraction common to all entities, helping communication, discovery and collaboration using protocols and data formats [10]. Our alternative approach is to use semantic web technologies to bring down to each entity the possibility to understand newly discovered other entities, thus decreasing the overhead on this layer, which is then solely in charge of a higher level system coordination.

2.2 An Adaptable Reasoning Engine for Context-Awareness

With the hardware plug & play and the software modularity in place, we had to reduce our use of specific application code. Indeed, adopting an imperative approach to implement context sensitive applications is very robust and requires only a short design phase. However, it brings deep constraints in term of reusability in personalized environments and adaptability in dynamic environments. As introduced in Sect. 1, a declarative approach allows for a more efficient separation between application logic and underlying models describing the use-case and peculiarities of the environment. Although this choice represents an important trade-off on the system's robustness and the effort to be put at the design phase, it seems unavoidable when targeting a deployment of more than just a dozen of homes. Although reasoners are the heart of AAL solutions, they do not need to be extremely powerful or complex. Their true requirement is to reach a consistent result in a limited time, which can be implemented using semantic reasoners. Moreover, this choice integrates well with the semantic description of the environment and its entities needed for the semantic plug & play presented above. The selection of relevant services and interaction modalities is then performed using semantic matching between the knowledge about users' context derived from sensor events and formalized into an ontology, and respectively services' and devices' semantic profiles. Finally, and as detailed further in Sect. 3.4, semantic technologies are perfectly adapted to model contextual information, along with its specificities.

3 Requirements on Semantic Inference Engines for AAL

In this part, we try to highlight the "must have" features of a semantic reasoner in order to be used efficiently into an AAL deployment.

3.1 Retractability of Knowledge

In assisted living spaces, contextual information is evolving and a detected situation is valid for a short period of time. The most needed feature for a reasoner to be used in AAL is the possibility and ease to retract information, both asserted and inferred. It has not been ignored that removing pieces of knowledge from an ontology is traditionally not a good practise. However, there are several reasons to support this choice in the targeted use-case. Most importantly, we do not want to overload the triple store with deprecated triples having an older time-stamp. We would also prefer to avoid dedicating processing time to select triples with the newest time-stamp. To support this choice, a mechanism has been designed such that the existence of a "thing" is never removed from the ontology. In other words, triples defining a new class, property or individual will never be removed. In an ontological graph, nodes are therefore anchored, while branches can be changed freely to represent the current contextual information available. E.g. if a resident walks to another room, the triple *ns:resident aal:locatedIn ns:kitchen*

is replaced by *ns:resident aal:locatedIn ns:bedroom*, whereas the "existential" triples *ns:kitchen rdf:type aal:Room* and *ns:bedroom rdf:type aal:Room* remain untouched.

We would like as well to retract inferred triples easily, when the conditions necessary to their inference are not fulfilled anymore. Using a graphical analogy, let us consider an asserted piece of knowledge as a node, and the knowledge inferred partly from this node as new nodes branching downwards (unidirectional relation) from it. The expected behaviour is that if a node is removed, which means the represented piece of knowledge is withdrawn, all nodes branching downwards from it should be removed as well. Although it is easy to use SPARQL queries, among others, to update the asserted triples in the ontology, the automatic removal of inferred triples as described above is more complex. Due to the monotonicity assumption of the Resource Description Framework (RDF), and the Semantic Web Rule Language (SWRL) being built on top of RDF, SWRL rules can be written to add new triples into an ontology but not to retract triples from it [14]. If one tries to assert a new value for a property, two values will then be coexisting. Optionally, the property can be characterised as *functional* to indicate that only one value is possible. However, this does not mean that the property will be updated but rather that the knowledge will become inconsistent when the two values are coexisting.

Some reasoners — e.g. Pellet [17], Euler [6] — have a rule syntax that is not expressive enough to allow the retraction of knowledge. One must annotate a part of the knowledge as deprecated and write external queries (e.g. with SPARQL) to filter it out. Others — e.g. Jena [4], RacerPro [8] — use rules that can remove triples. In both cases, it is needed to manually retract knowledge inferred from the asserted-then-retracted "nodes". We did implement some inference rules dedicated to cleaning the ontology after a retraction happened. Although it is working well, this increases the complexity at design level and naturally decreases the performance of the system. We finally realized when experimenting on Euler that even though its expressiveness did not allow retraction of knowledge; the reasoner having *no live state*, the knowledge previously inferred from now-deprecated data is simply not inferred anymore. The live state of a reasoner is the state in which the reasoner remains in between two inference process. It is used so as to keep in memory the inferred state of an ontology, thus inferred knowledge does not need to be inferred again. In our use-case, we prefer to use a rule engine with no live state (i.e. no memory), as it is then only needed to care about information being asserted or retracted, and the rest is handled automatically, similarly to the "downwards branching nodes" approach described above. To summarize, reasoners often implement complex mechanisms to infer knowledge with incremental updates; but we found more suitable, in the AAL use-case, to use a *naive-only inference* like what is provided by Euler.

3.2 Processing Efficiency

Taking into account more common applications of the semantic web in the cloud, one can easily imagine having reasonable resources to process knowledge. How-

ever, in the AAL use-case, it is necessary to embed the reasoner into a low processing power and low power consumption device, so that this device can be easily integrated anywhere in a house. E.g. the reasoner used for our deployment runs on a tiny debian machine whose CPU turns at 500MHz with 500Mb of RAM, and consuming only 5W. Moreover, the data inferred is highly dynamic; unlike web data which is updated by human users with a low frequency, context information is derived from ambient sensors activations representing people's behaviour in real-time, therefore the update rate is way below the minute. Finally, the inference is used to compute which services should be provided in the environment and with which interaction modality; this is decided depending on people's action so users should have the feeling of an almost instant response time. Therefore the minimum inference frequency has to be set very low, which forms the requirement on the processing time, thus on processing efficiency.

3.3 Scalability of Inference

The conceived service platform for assistive living, partly described in this paper, is usually tested in a single room or at most an apartment. But it is difficult to estimate now the extent of the monitored/serviced space in which it will be deployed once AAL technologies get a larger impact. Let us consider that we are deploying at the scale of a whole building: should we plan to have one reasoner per room, per apartment or even per building? With regard to the Linked Data philosophy [2] and due to the interconnection of events inside the building, it makes sense to think of a reasoner per building to be able to draw relations between the data from all apartments. Or even considering the smart cities initiatives — suggesting the extension of smart spaces to the city level [3] — the number of triples to be considered at the reasoning step might suffer a genuine explosion. Thus we have to include a requirement on the scalability of the system, e.g. reasoners inferring with linear cost should be prioritized compared to those running with quadratic cost. Although we expressed in the introduction our reservations towards semantic reasoners' benchmarks, we give in Sect. 4 some figures to compare selected reasoners in this perspective.

3.4 An Opening on Uncertainty & Quality of Information

The main peculiarities of context information lie in its high interrelation, which is leveraged through the linked data approach; and its imperfection, inconsistent or incomplete information being common due to faulty hardware, delays between production and consumption of the information, or even networking problems. Although this is obvious to the engineer, ontological knowledge is naturally processed as an absolute truth if no notion of uncertainty or quality of information is introduced in the semantic model or if the reasoner is not conceived to consider these notions. A semantic modelling language can cope with this by introducing classes of information and associations in accordance with their persistence and source. The adopted description must also allow a range of temporal characterizations as well as alternative context representations at different levels of

abstraction. Introducing such concept into the reasoning engine remains very challenging, this is why we entitled this part *opening on*. Although we do not have a strong contribution here, we could not write about these requirements without mentioning the QoI aspect. The idea of the classifying associations by Henricksen et al. [9] appears to be an important key towards QoI-based semantic reasoning and although they did not explicitly refer to the semantic web paradigm, the model proposed was obviously close to it and its semantic implementation would be straight forward. We also believe that the uncertainty aspect will not be tackled by the engine itself, rather that it is the way the engine is used and wrapped that can ever address this aspect.

4 The Appropriate Reasoner

We have presented in the previous section the requirements we gathered for a practical semantic reasoner in the AAL use-case. Some are immediate necessities like the retractability of knowledge or the processing efficiency, others are key challenges enabling larger scale deployments like the scalability, or a better reliability like the quality of information. Below, we give some feedback on 4 available reasoners that we have selected and tried over the last year.

4.1 Comparing Reasoners' Usability

Jena: the Predominant Reasoner. In the AAL community, the Jena framework [4] is predominantly used. This might partly be due to the unawareness about its alternatives as well as its apparent ease of use compared to other reasoning engines. Indeed, Jena has a few advantages compared to its rivals with probably the most complete Java API for building semantic applications. Unlike most of the other alternatives, Jena has been designed to be used on Java and its way of programming is therefore more natural for a lambda programmer getting a first hand on semantic web technologies. Actually, taking into account the possibility to implement Java built-ins to be called directly from an inference rule, one might not even realize the differences brought by the declarative reasoning paradigm. Moreover, Jena comes fully-featured with, among others, an API to build, populate and modify ontologies, an inference engine using its own rule format, and an integrated SPARQL query point.

Despite all the above, we are having mixed feelings about our experience developing with Jena and would like to express some reservations about it. In fact, without having to load the ontology much, we could observe some inconsistencies in the reasoning when trying to use several rules to collaborate on one decision. When searching for an explanation to this flimsy behaviour, we found out that Jena was actually having an incomplete integrated inference engine [16] and that using Pellet [17] as an external reasoner was advised. Consequently we started to compare the features of available semantic reasoners and their ease of use in our peculiar use-case. Our motivation towards writing this paper grew as we met researchers in the community interested in finding an appropriate reasoner.

Pellet: the Famous Alternative. Since Jena has an option to use Pellet [17] as an external reasoner, it allows to change reasoner while keeping the system infrastructure, like the modules updating the ontology depending on sensors inputs. Moreover, Pellet's rules are using SWRL, the Semantic Web Rule Language, which makes the rules compatible with some other reasoners. Logically we decided to try Pellet but as explained in Sect. 3.1, SWRL does not allow retraction of triples and makes it difficult to be used in AAL use-cases.

RacerPro: the Fully-Featured Commercial Option. We then searched a reasoner able to tackle the knowledge retractability issue and found RacerPro [8], a commercial reasoner with add-ons to the W3C recommendations. Essentially, its expressiveness allows for the retraction of triples from the ontology standing in memory. Though it is necessary to write rules dedicated to retract triples to clean the ontology, the system is at least functional. Other than being a closed-source shareware, RacerPro has its own downsides due to its own powerful rule/query language. This language is actually the most complex one we have used, which did bring a heavier workload on the implementation phase.

Euler: the Lightweight, Naive Reasoner. While facing implementation difficulties with RacerPro, we found an alternative solution with Euler [6], more specifically the EYE implementation by De Roo et al. from AGFA Healthcare. Euler is notably using Notation3 (N3), the most human-readable RDF syntax. It has the advantage to be among the fastest reasoners we found that had a full OWL-DL entailment, and it is also the most lightweight of the reasoners we selected. However, we faced the same retractability limitation as with Pellet. Despite this, we found out that Euler providing a *naive* inference, it was not problematic as explained in Sect. 3.1. Our current choice remains Euler and the validation results presented in Sect. 4.3 have been obtained using EYE.

Synthetic Comparison. We have described above the process we went through and highlighted the pros and cons of each selected reasoner. The Table 1 summarizes the aspects taken into consideration with some key specifications for each of the four reasoners. Its first half provides a good representation of the engines' expressiveness, the ease of use being purely qualitative, subjective, and based on our hands-on experience. It is interesting to note that languages purely implementing the semantic web specifications are the ones we found the most straight forward to use. Based on this first half, Pellet and Euler appear to be the two best options. We refined then our analysis with more quantitative specifications, addressing in a way the concerns raised in Sects. 3.2 & 3.3. Despite the reservations we hold about such benchmarks, we realize with the response time figures the superiority of Euler. Finally, the qualitative characterisation of engines' scalability is given subjectively, taking into account the response time profile, the ease of use and the inference completeness (OWL-DL entailment). These multiple aspects and requirements taken into consideration, this is why

Table 1. A comparative table of semantic reasoners

	Jena	**Pellet**	**RacerPro**	**EYE**
OWL-DL entailment	incomplete	full	full	full
Rule format	own, basic & built-ins	SWRL	own, powerful	N3 & built-ins
Retractability	yes	can emulate stateless	yes	stateless
Ease of use	average	easy	complex	easy
Response time for 100 triples[1]	783ms	442ms	∼503ms	4ms
Response time for 1,000 triples[1]	29,330ms	38,836ms	∼44,166ms	40ms
Response time for 10,000 triples[1]	out of memory	out of memory	out of memory	436ms
Scalability	Very limited	Average	Limited	Good
Size (download)	22.3Mb	24.3Mb	60.3Mb	12.9Mb
Licensing	freeware, open source	freeware, open source	shareware, closed source	freeware, open source

we chose to use Euler. To be specific about the scope of this choice, the authors would like to highlight that Euler has two advantages applicable to any use-case: its scalability, due to its optimized implementation based on YAP-Prolog, and its human readable formalization language using N3. However, Euler is a naive (memoryless) reasoner, which is crucial from our perspective but might be counter-productive in many applications. Here lies the main trade-off in our choice.

At this point, one might also wonder about the level of reasoning chosen in our implementation. Using Euler for the inference, developers are able to use any subset of rules catering to their specific needs. Our implementation is now using a subset of OWL2-RL, but we might choose to use rules from a higher level of reasoning if needed in the future.

4.2 Design: Integration of the Reasoner into a Service Platform

In order to validate our ideas and choices, Euler has been integrated into a context-aware service platform, insuring the selection and provision of appropriate services to end-users depending on their profile and situational needs [18]. As represented in Fig. 2, the platform is based on the OSGi specification, specifically the Apache Felix implementation, which materializes the SOA approach

[1] Figures extracted from [13] for Jena, Pellet and Euler. Cross-integration of RacerPro through a comparison with Pellet [11].

and facilitates the deployment of AAL technologies as explained in Sect. 2.1. Inside the Felix container, the platform is composed of several modules (called bundles in the OSGi lexical) that can be installed, updated and removed at runtime without interrupting the platform's operation. We use this aspect of the OSGi specifications to implement the plug & play behaviour introduced in Sect. 2.1. A Wireless Sensor Management Service (WSMS) bundle has been developed to handle the ZigBee communication between sensors and the platform: once a sensor is turned on in the environment, this bundle automatically generates a new bundle representing and describing the sensor in the platform. A similar mechanism ensuring devices plug & play is currently being developed with a bundle supporting heterogeneous communication layers (e.g. Wi-Fi, Bluetooth, 3G). The dynamic aspect of OSGi bundles also permits the integration of new end-user services (e.g. reminder services, home control) at runtime.

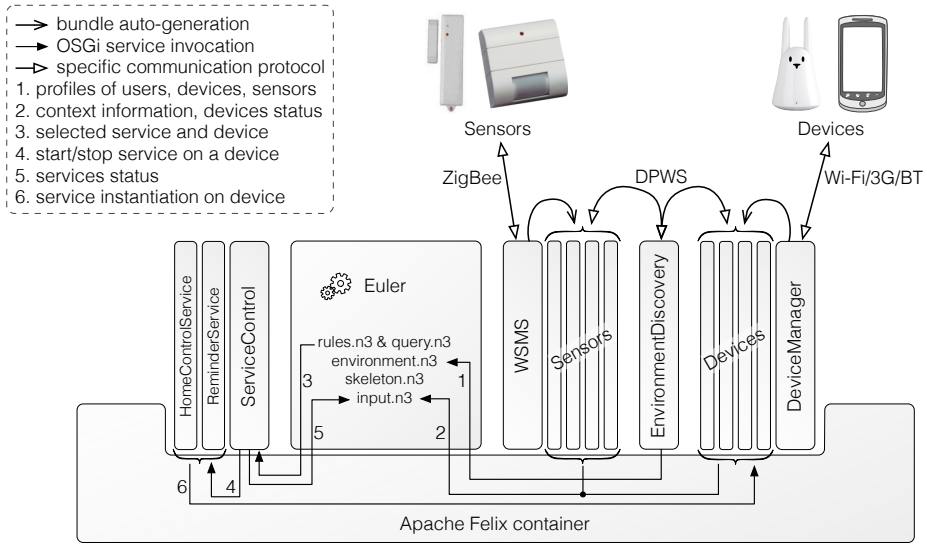

Fig. 2. Detailed architecture of our context-aware service platform

To use Euler as the core reasoner of the platform, it had to be integrated on OSGi. We took the Euler open source Java API and modified it slightly to make it compatible with the dynamic class loading feature overridden by OSGi. In the platform, Euler infers continuously in an independent system thread. The ontology is build from a set of files written in N3 and containing different kinds of information; its update is reduced to files parsing and modification. There is a file (skeleton.n3) constituted of the classes and properties that can be instantiated in the whole system to represent the current contextual information. It is the T-box of our ontological model, the highest level of modelling used in our system, containing notably models of the physical environment, the users and their

behaviour, as well as the available categories of services. Another few files (let us consider a merged example named environment.n3) contain the knowledge coming from the environment discovery phase: e.g. actual users and their profile, or sensors, devices and services along with their semantic profile. Two files (rules.n3 and query.n3) contain the rules and queries necessary for the inference process, thus centralize the application logic, which is the system context-aware decision-making. Finally, a file named input.n3 is updated at run-time through a dedicated interface to reflect the changes in the environment: real-time context information, services or devices status, etc.

In order to ensure discovery and events exchange between the different bundles in the platform, we are using the Device Profile Web Service (DPWS) protocol. DPWS uses several standards from the web services specification — namely WSDL, WS-Discovery, WS-Eventing and SOAP — in order to advertise and discover bundles, as well as for events exchange. Once a bundle representing an entity in the environment is generated, it uses DPWS protocol to advertise itself and send a description of his capabilities. A DPWS client bundle (EnvironmentDiscovery) is in charge of discovering these bundles and updating the environment.n3 file with a semantic description of their corresponding entity. Interested bundles can then subscribe to events coming from the entity, for example to update the input.n3 file. Euler parses all given N3 files, infer a high level representation of users context, and then infer which services need to be started or stopped, as well as on which devices they should be instantiated. This decision is finally executed transparently through the ServiceControl bundle.

4.3 Validation: Prototype & Deployment

After a first implementation of the platform, a validating deployment process was organized in collaboration with Peacehaven nursing home in Singapore. Peacehaven is hosting elderly residents with mild dementia who need of a continuous assistance from caregivers in order to perform their ADLs. The deployment of our platform in the nursing home assists the residents with reminders to increase their independence, as well as the caregivers by raising targeted notifications when an abnormal situation is detected. Initially, a proof of concept deployment was realized in May 2011 ending with a demonstration to the nursing home staff and management. We received good feedback about the features and performance, and filed shortly after this an Ethics Approval application for a real-world deployment with genuine residents. In August 2011, we received the Ethics Approval and put in place a sub-part of the system for technological real-world experimentation. This system's pre-trial period started in November 2011, the specificity being that interaction is instantiated only with caregivers so as to test the system without affecting residents with eventual false alarms.

Prototype Description The platform, described in Fig. 3, is running on a tiny (115 x 115 x 35 mm, 505g) fanless debian machine, mounted with a 500MB RAM/500Hz CPU, a 4GB Compact Flash drive, and a power consumption of

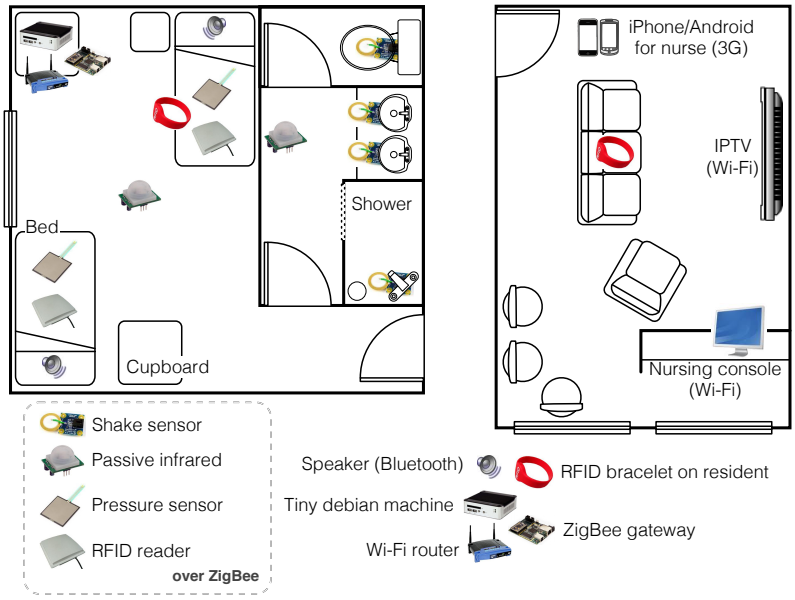

Fig. 3. Hardware infrastructure & use-case of the deployed solution

only 5W. Sensors are using the ZigBee communication protocol on a wireless sensor network based on Crossbow's IRIS mote platform. A Crossbow node is connected via serial port to the debian machine, serving as gateway. The communication with other devices in the environment uses bluetooth for residents' embedded speakers, a client-server communication based on Jabber over Wi-Fi for the residents' IPTV and the nursing console (Windows7 machine with touchscreen) or 3G for the nurses' smartphones (Samsung Galaxy S2 with Android 2.3 and Apple iPhone 4 with iOS 5). In this phase, we monitor two residents wearing an RFID bracelet for identification, bluetooth speakers are deactivated to avoid eventual trouble to the residents in case of a system malfunction. The activities in the bathroom are monitored using shake sensors (accelerometers) placed on the pipes to detect taps/shower usage. A shake sensor is also embedded in the soap dispenser to detect soap usage during the shower. Motion sensors (passive infrared) are positioned on the ceiling of both the bedroom and the bathroom to detect presence and measure the amount of activity. The bedroom also has pressure sensors (force sensing resistors) under the mattresses and an RFID reader allowing the detection of residents and their identification. In the following phases, it is planned to increase the number of rooms monitored to reach a number of 10 residents, there are only 2 now.

In Figure 4, we illustrate our conception with a graphical representation of our model (at T-box level). It is only a part of the whole ontological model in use, simplified for readability and conciseness, thus containing only high-level

concepts related to the use-case and not to the internal reasoning process. This model will then be populated with knowledge about the actual environment of deployment, users and their profile, services activated, hardware deployed and real-time knowledge derived from the sensor data concerning the activities of the residents. In average, the T-box and A-box together constitute around 150 triples to be processed by the reasoner. Depending on the activated features, this processing consists in 10 to 15 rules and queries. The rules used are similar to the examples given below:

$$\forall \textit{ Service s, Resident r, Location l, Device dc, Activity a, Deviance da}$$
$$\textit{(r hasContext da)} \wedge \textit{(s helpsWith da)} \Rightarrow \textit{(s runningFor r)}$$
$$\textit{(s runningFor r)} \wedge \textit{(r locatedIn l)} \wedge \textit{(dc deployedIn l)} \Rightarrow \textit{(s onDevice dc)}$$
$$\textit{(r hasContext a)} \wedge \textit{(a needHands true)} \wedge \textit{(dc handheld true)} \Rightarrow \textit{(dc fitted false)}$$

The inferred knowledge is not to be used by any front-end application; it is rather used for back-end decision making to provide services seamlessly. For the time-being, context information is inferred from sensor stream by another module using a business rule engine and then used to perform service and device selection semantically. However, the ontological model is currently being extended to infer contextual knowledge using our semantic engine, as defended by Chen et al. [5].

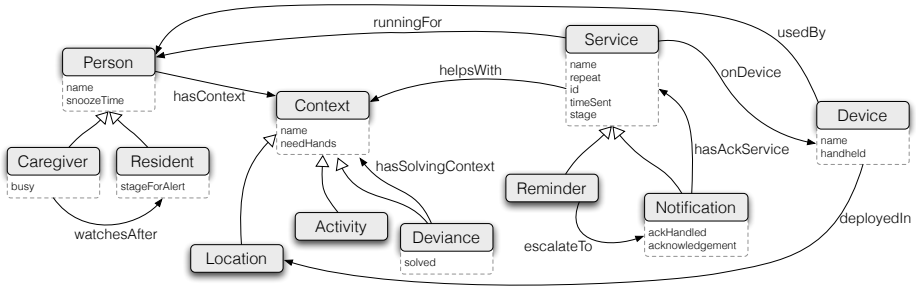

Fig. 4. Ontological sub-model for elderly assistance in a smart space

In Peacehaven nursing home, three services have been running for six months at the time of writing. These services have been designed in collaboration with the nursing staff to respond to the specific needs of the residents who agreed to test the system. These services are monitoring deviances (i.e. problematic behaviours) that are the most likely to lead to a fall. On one side, there are bathroom activities where the space is narrow and ground wet, with notifications being raised when a resident has been showering for an unusual time or when he forgets to turn off the tap of the basin. And on the other side, we raise a notification when a resident is detected to be wandering during the night.

Prototype Performance The first aspect in which we wish to judge the system performance is regarding its uptime. In this aspect, we learnt a lot from our deployment in Peacehaven and highlighted the main areas, summarized in Figure 5, in which the platform had to be improved. We worked hard on improving these aspects and were able to improve the average uptime, from 3 days in December 2011 to 11 days in May 2012. The more technical errors were considerably reduced and the challenge today concerns the 12% of reasoning failure, mostly due to our implementation, rather than the reasoner itself. We are currently reimplementing Euler's module, to improve on this aspect.

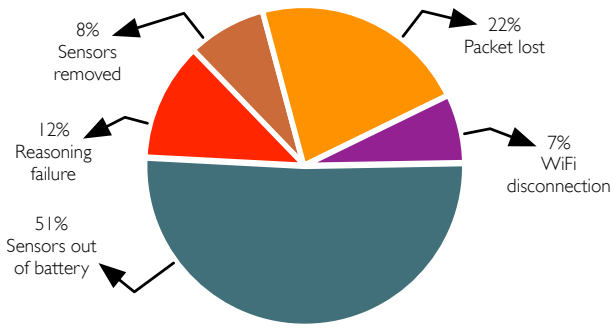

Fig. 5. Pie chart for system crash reasons in Peacehaven

During the trial period, we compute some figures to validate the platform's performance in term of accuracy and timeliness. Results are given based on the analysis of the logs for the two bathroom services described above, during an uptime period of nine days. We consider *atomic* events first — e.g. the use of taps and shower — which happened 34 times a day in average with a recognition accuracy of 71%. This was obtained by comparing the system log files with a manual record of activities provided by the nurses each time they had an intervention in the room. *Complex* events — that correspond to deviances and services provision — happened 7 times a day in average, with an accuracy of service delivery of 70%. This accuracy characterises the ratio between the number of times a service was delivered over the number of times it was needed. As complex events are derived from atomic events, we conclude that little error is introduced by the event mining (reasoning) itself. Finally, the system's reaction time, calculated between the time a service is needed and the time it is delivered in the environment, has an average of 2.713s, which has been refined in 1.226s for Euler module's processing itself, 0.735s for the communication between modules and 0.752s for the processing due to other miscellaneous bundles.

The last aspect considered to estimate the performance is the time needed to set-up the system into a new environment. Indeed, our goal was to build a more flexible system that can adapt to different environments and needs. Therefore,

we have analysed the time needed to adapt the operational platform to a new use-case, counting on a team of 2 engineer-researchers. With the imperative approach used before our adoption of semantic technologies, the first reasoner was written in 5 days and its subsequent adaptation took 3 days. We then needed several months to build the first semantic version of the platform. As we were not experienced, we had to discover the existing tools, as well as build the required models. The subsequent adaptation to a new deployment with its specificities took us only 2 to 3 hours, mostly to adapt the model. In our semantic platform, rules and thus the system logic are kept unchanged.

Of course, the aim here is not to compete with commercial systems on the real-time performance but to validate that more versatile solutions driven by semantic web technologies are an option with a sufficient performance, as observed in a real-life deployment. In this aspect, the results obtained are judged satisfactory for a first, unoptimized implementation.

5 Conclusion

In this paper, we have highlighted how AAL solutions can leverage semantic web technologies in order to enhance their modularity and versatility. AAL can indeed be driven towards a more flexible deployability by using semantic web technologies with a double role of inference engine and integrating abstraction layer. Reflecting on a year of trying out existing inference engines, we have given our take on the requirements for a semantic reasoner to be used efficiently in AAL use-cases. Some reasoners that we tested have then been compared with regard to the mentioned requirements and our choices justified. Finally, the conception and validation phases for the integration of the reasoner have been described and some results provided.

Although the tools available are not always fitting well with the AAL use-case, we observed the important contributions of semantic web technologies to this field. We are still designing and implementing some features made possible through the semantic web in order to enhance the flexibility of our solution. Among others we are planning to create a smart-space composer helping at the deployment step to configure the system, on one hand by making possible a rich semantic characterization of the deployed entities, and on the other hand by generating a specific inference rule-set from a set of abstracted meta-rules and the entire semantic characterization of the specific environment.

Acknowledgements

This work has been funded under AMUPADH project through A*STAR Home 2015 research program (SERC) in Singapore, and by CNRS ModCo project in France. The authors would like to express their gratitude to Peacehaven nursing home in Singapore and the residents involved; to Racer Systems for according us a free educational license of RacerPro; and to Jos De Roo from AGFA Healthcare Belgium for his support on Euler and his inputs to this paper.

References

1. Aloulou, H., Mokhtari, M., Tiberghien, T., Biswas, J., Lin, J.H.K.: A semantic plug&play based framework for ambient assisted living. Lecture Notes in Computer Science, vol. 7251, pp. 165–172. Springer (2012)
2. Berners-Lee, T.: Design issues: Linked data. Published online at http://www.w3.org/DesignIssues/LinkedData.html (2006)
3. Caragliu, A., Del Bo, C., Nijkamp, P.: Smart cities in Europe. Vrije Universiteit, Faculty of Economics and Business Administration (2009)
4. Carroll, J., Dickinson, I., Dollin, C., Reynolds, D., Seaborne, A., Wilkinson, K.: Jena: implementing the semantic web recommendations. In: Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters. pp. 74–83. ACM (2004)
5. Chen, L., Nugent, C., Wang, H.: A knowledge-driven approach to activity recognition in smart homes. IEEE Transactions on Knowledge and Data Engineering (2011)
6. De Roo, J.: Euler proof mechanism - eye. http://eulersharp.sourceforge.net/ (2005)
7. Floeck, M., Litz, L., Rodner, T.: An ambient approach to emergency detection based on location tracking. Lecture Notes in Computer Science, vol. 6719, pp. 296–302. Springer (2011)
8. Haarslev, V., Möller, R.: Description of the racer system and its applications. Description Logics 49 (2001)
9. Henricksen, K., Indulska, J., Rakotonirainy, A.: Modeling context information in pervasive computing systems. Pervasive Computing pp. 79–117 (2002)
10. Hong, J., Landay, J.: An infrastructure approach to context-aware computing. Human-Computer Interaction 16(2), 287–303 (2001)
11. Luther, M., Liebig, T., Böhm, S., Noppens, O.: Who the heck is the father of bob? The Semantic Web: Research and Applications pp. 66–80 (2009)
12. Mokhtari, M., Aloulou, H., Tiberghien, T., Biswas, J., Racoceanu, D., Yap, P.: New trends to support independence in persons with mild dementia. In: International Journal of Gerontology (2012)
13. Osmun, T., De Roo, J.: http://eulersharp.sourceforge.net/2009/12dtb/ (deep taxonomy benchmark - sources)
14. Plas, D.J., Verheijen, M., Zwaal, H., Hutschemaekers, M.: Manipulating context information with swrl. Tech. rep., Ericsson Telecommunicatie B.V. (2006)
15. Rougier, C., Auvinet, E., Rousseau, J., Mignotte, M., Meunier, J.: Fall detection from depth map video sequences. Lecture Notes in Computer Science, vol. 6719, pp. 121–128. Springer (2011)
16. Singh, S., Karwayun, R.: A comparative study of inference engines. In: Seventh International Conference on Information Technology: New Generations (ITNG). IEEE (2010)
17. Sirin, E., Parsia, B., Grau, B., Kalyanpur, A., Katz, Y.: Pellet: A practical owl-dl reasoner. Web Semantics: science, services and agents on the World Wide Web (2007)
18. Tiberghien, T., Mokhtari, M., Aloulou, H., Biswas, J., Zhu, J., Lee, V.: Handling user interface plasticity in assistive environment: Ubismart framework. Lecture Notes in Computer Science, vol. 6719, pp. 256–260. Springer (2011)
19. Weithoner, T., Liebig, T., Luther, M., Bohm, S.: Whats wrong with owl benchmarks? In: Second International Workshop on Scalable Semantic Web Knowledge Base Systems (SSWS 2006)