A FLEXIBLE MIDDLEWARE COMPONENT FOR CONTEXT AWARE APPLICATIONS

Cristina Barbero, Paola Dal Zovo and Barbara Gobbi
Concept Reply (a Business Unit of Santer Reply SPA), via Cardinal Massaia 83, 10147 Torino, Italy

Keywords: Context Aware Computing, Pervasive Computing, Service Oriented Architecture and Middleware, Semantic Technologies, Automated Reasoning, Internet of Things.

Abstract: The ever-growing complexity of pervasive and Internet of Things enabled environments raises great challenges to context-aware pervasive application development. In particular, context representation and reasoning methods, as well as middleware and supporting infrastructures for context sensitive application engineering, must have a high level of flexibility in order to cope with the increasing dynamicity and heterogeneity of pervasive scenarios. This paper presents a solution devised to provide the foundations for the development of context-adaptive applications with diverse requirements. The Context Awareness component consists of an extensible and configurable framework that integrates a semantic reasoning module and multiple processing agents providing specialized / optimized processing capabilities. Finally, a case study shows how the adopted solution allows tackling the complexity of context-aware applications development.

1 INTRODUCTION

The increasing complexity of pervasive computing environments, and the Internet of Things (IoT) vision of a world of everything connected, open great market opportunities and raise many challenges. Context-aware pervasive applications, while potentially offering huge added values, still suffer from limitations hindering their full development and spreading.

The development of context-aware applications for the IoT is complex as it requires considering aspects related to distributed computing, discovery of services, network connectivity, limited processing resources of target devices, mobility. Moreover, the domain of the IoT is so wide and the requirements of context-aware applications may be so diverse that a single automated reasoning methodology cannot provide a satisfactory answer to all needs.

Context-adaptive application engineering must be based on an efficient infrastructure and middleware, in order to reduce engineering effort and complexity and enhance the soundness of the produced solutions. Some challenges are related to the adequacy of context representation and reasoning methods to cope with increasing complex, dynamic and heterogeneous pervasive environments and settings. An extensive review of such issues is (Soylu, 2009).

This paper presents a Context-Awareness (CA) solution based on a modular and flexible platform supporting the creation and management of pervasive, IoT-enabling applications. The CA component is given by an extensible framework that integrates a semantic representation and reasoning module and multiple processing agents providing additional, specialized and/or optimized processing capabilities (e.g. raw data filtering and aggregation, probabilistic and statistic data management, light event processing for resource-constrained devices). Case studies have shown the effectiveness of our CA solution, which derives from both the strong platform base and the flexibility of the reasoning framework.

The paper is organized as follows. Section 2 presents the motivations of our approach and related works. Section 3 gives an overview of the platform, while section 4 describes in details the CA framework. Section 5 presents a case study. Section 6 presents the conclusions and future work.

Barbero C., Dal Zovo P. and Gobbi B. (2011).
A FLEXIBLE MIDDLEWARE COMPONENT FOR CONTEXT AWARE APPLICATIONS.
In *Proceedings of the 1st International Conference on Pervasive and Embedded Computing and Communication Systems*, pages 32-41
DOI: 10.5220/0003363300320041
Copyright © SciTePress

2 MOTIVATIONS AND RELATED WORKS

In the past few years a lot of work has been done in areas that are key enablers for context-aware computing, such as pervasive computing, service-oriented architectures, automated reasoning, and semantic modeling languages.

In the current state-of-the-art of context-aware middleware different approaches have been proposed to sense context and to adapt to changes, often with a different focus. An overview is provided by the survey (Kjær, 2007), which classifies a set of existing context-aware middleware according to a taxonomy including relevant aspects such as adaption mechanisms, and concludes that no single middleware system is appropriate for all settings.

With respect to the semantic representation of data we believe that an interesting initiative is the W3C Semantic Web (W3C Semantic Web, 2001). The semantic web is intended to allow semantic data exchange and interoperability at the semantic level, by relying on a stack of standard languages such as XML, RDF, and OWL. In particular, OWL adds more vocabulary for describing properties, classes, and relations between classes. In addition, SPARQL is a protocol and query language for semantic web data sources, while SWRL is the proposed rules language.

Semantic Web standards are being applied in an increasingly large spectrum of applications in which domain knowledge is conceptualized and formalized usually by means of ontology in order to support diversified and automated reasoning (Leger, 2008). Some works choose to adopt the Semantic Web concept in the Internet of Things field, for context-aware applications over wireless sensor networks (Huang, 2008), (Roman, 2002), (Kostelník, 2008), and the survey in (Strang, 2004) indicates that ontologies are a valid solution for context modeling.

So far, no standardized sensor ontology still exists, although there is an initiative to define a shared vocabulary for sensor networks based on W3C semantic web languages. Adopting semantic web languages, considering common concepts in proposed ontology, means to be in the right direction towards semantic interoperability. This choice presents nonetheless some challenges. In fact, some works highlight some issues in the adoption of semantic technologies (Wolf, 2009), especially attempting to exploit them in dynamic environments and when dealing with data from hardware devices, often mobiles.

The survey (Perttunen, 2009) shows that OWL is conveniently used in many recent works to represent context, while reasoning is seldom based only on pure OWL inference. Usually, rules are used to incorporate human knowledge to guide the system. The survey indicates that some works make use of other reasoning methods, such as cased based reasoning, probabilistic reasoning, or hybrid approaches.

The motivations behind the need of a probabilistic approach are also well illustrated in (Garofalakis, 2006). Sensor data is typically a noisy and uncertain observation of the phenomenon it is intended to capture, hard to “trust”, and often conveyed in form of an unreliable stream. Spatial and temporal correlation between data should be exploited for better understanding the underlying physical phenomena, other than for efficiency reasons.

Another perspective worth notice is the one brought by the survey (Kapitsaki, 2009), which focuses on evaluation of the techniques that enable the exploitation of contextual information (“context handling” techniques), assessed from the viewpoint of service developer. The authors don't focus much on middleware, ontology-based or rules-based solutions, as they prefer concentrating more on approaches based on programming, message interception, and model-driven approaches. We refer to this analysis of state-of-the-art as in designing our CA solution we considered criteria such as those that they mentioned. Those criteria include:

- The flexibility that the context adaptation mechanism provides to the developer. A related criterion is about the possibility of applying the specific context adaptation mechanism to an existing system.
- The easiness of use for a developer, not very familiar with the specific context adaptation mechanism.
- The easiness of refactoring a service (if there are modifications in either the business logic or the context handling logic)
- Decoupling from business logic: the degree of independence between context information and service logic.

Another recent article addressing context-aware web service is (Truong, 2009), which analyzes CA techniques and related works in web service systems. In the context reasoning techniques section, it highlights that some systems rely only on XML-based context information and have scarce reasoning capabilities, and only few systems are based on semantic reasoning due to difficulties to integrate ontology into context-aware systems.

We choose to adopt semantic technologies for the benefits in term of expressiveness of the representation and for being in line with current standardization directions. We also recognize the need of being open to a probabilistic approach in some scenarios, and to other automated knowledge processing techniques that might be needed in some scenarios.

To summarize, the main goal we pursue in our work is to provide our middleware with instruments to facilitate the development of context-aware services, in particular building an infrastructure to:

- Allow rapid development of a certain class of rules-based context-aware services, based on semantic models of devices and services.
- Reduce the development effort needed to realize context-aware services able to answer a variety of functional and non-functional requirements, including the need to deal with uncertain data and the need of processing on resource-constrained devices.

3 ARCHITECTURAL OVERVIEW

Our CA solution is part of a platform supporting the creation and deployment of pervasive, IoT-enabling applications based on networks of connected devices. The platform, designed to be modular, flexible and scalable, is characterized by:

- A Service Oriented Architecture where each network element can expose its functionalities by means of a loosely-coupling Web Services interface. This enhances the modularity of the system and its interoperability (through the adopted standards).
- Support for a diverse and changing set of devices, communication protocols and technologies (both wired and wireless, long and short range), service exposure and discovery (self-configuration) modalities.
- A flexible context reasoning framework that can be tailored on the basis of the kind of hosting nodes and of needed applications (e.g. a resource-constrained device could only host a lightweight processing agent, a complex application could require a semantic reasoner and an event-processing specialized agent).

Figure 1 shows different kinds of nodes (specified below) managed by the platform. Even the simplest devices can be part of the infrastructure, through the support of virtualization nodes.

- *Simple Nodes* have short range connectivity (e.g. Wifi, ZigBee, Bluetooth) and reduced computational capabilities. They don't support TCP/IP and Web Services, so they are connected to the network and expose their services by means of the Virtualization Nodes.
- *Intelligent Nodes* have short range and optionally long range (e.g. GSM, GPRS, UMTS, Ethernet) connectivity, TCP/IP and Web Services support, self-configuration capabilities.
- *Virtualization Nodes* are similar to Intelligent Nodes for capabilities and provide virtualization services to sub-networks of Simple Nodes (in particular TCP/IP and Web Services support and self-configuration capabilities).
- *Aggregation Nodes* have high processing power and expose in a homogeneous way the Web Services offered by the different nodes and sub-networks to the back-end systems.

Figure 2 shows the layers and the main components provided by the platform, in particular:

- *Network Management* component, dealing with network configuration, performance, fault and diagnostics.
- *Web Services* component, managing the exposure of the services (described with homogeneous XML logical models) through different modalities, e.g. dual bindings in HttpDual protocol, single bindings in Rest protocol, DPWS

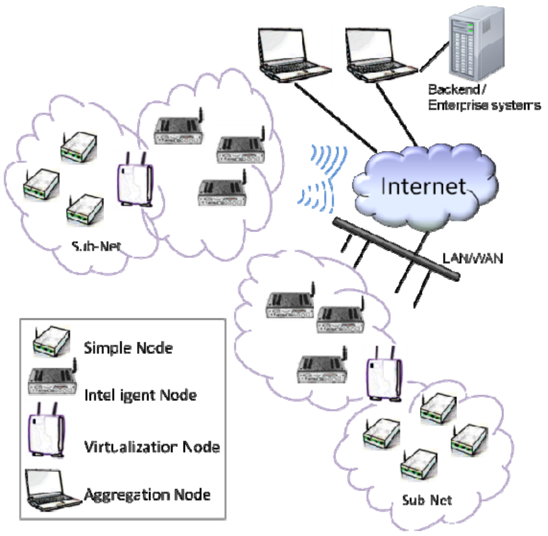


Figure 1: Network nodes managed by the platform.

(Devices Profile for Web Services, 2009) recommended bindings. It also manages different service discovery modalities, e.g. based on UDDI or WS-Discovery protocol. Moreover, it manages the interaction with the other platform components, and notifies the registered clients about the updates on data related to the exposed services.

- *Context Awareness* component, in charge of processing contextual information and using the recognized context to adapt application behaviour. It will be described in details in the next section.
- *Security / Localization / UI Management* components, providing support for, respectively, security and trust, location-based services, UI of applications.

4 CONTEXT AWARENESS FRAMEWORK

In this section we describe the CA module, in charge of processing contextual information, recognizing current context and making decisions about which actions are needed. This module is designed to be modular and flexible, and to allow easy extensions by addition of needed processing agents in the form of plug-ins. Figure 3 shows the CA framework.

A reasoner host is in charge of retrieving and loading reasoning components for the system, and, once loaded, to allow them to exchange information with other reasoning services and with other nodes in the system. Communication is based on a publish/subscribe model.

The CA module is notified, from other components in the system, of relevant changes it may need for reasoning about, namely of:

- New entities, as they connect to the network. Entities may be e.g. devices, software components in our platform or external web services, system nodes, people who get known to the system.
- Disappeared entities, e.g. disconnected devices.
- Occurred changes in entities, e.g. new sensor reading, changed location, changed actuators state, etc.

Our middleware relies on XML logical models to represent system entities and to exchange and share information about them between the platform modules. Thus, the CA module receives information about entities and handles context adaptation in a uniform and abstract way, leaving to other

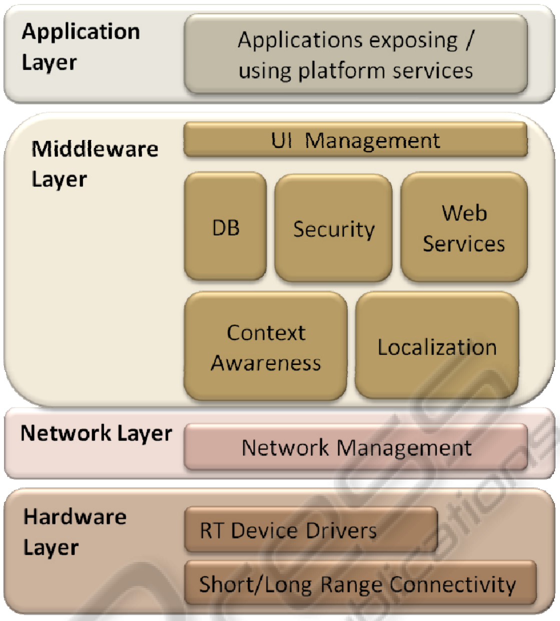


Figure 2: Platform layered view, main components.

middleware modules tasks related to low-level communication and handling of devices.

For describing the structure and constraining the contents of XML logical models we used the XML Schema Definition (XSD) (XSD, 2004). The XML Schema allows validating XML models as they are being updated, ensuring that data remain conform to the defined schema. The XML Schema describes models and their elements, attributes, data types, specializing type definitions by extending higher level types in an entity hierarchy. Among the

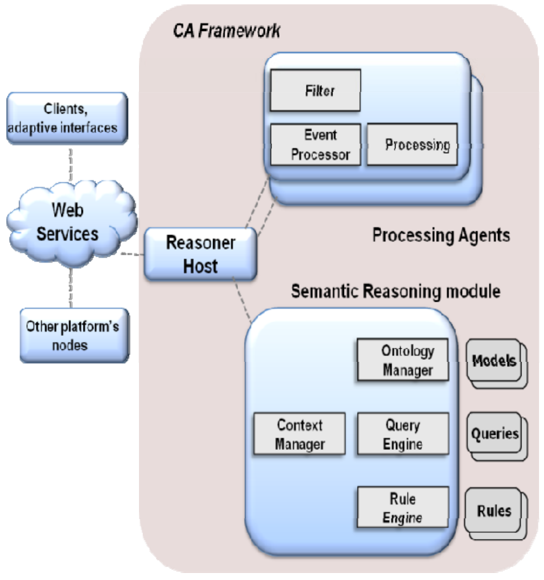


Figure 3: Context awareness framework.

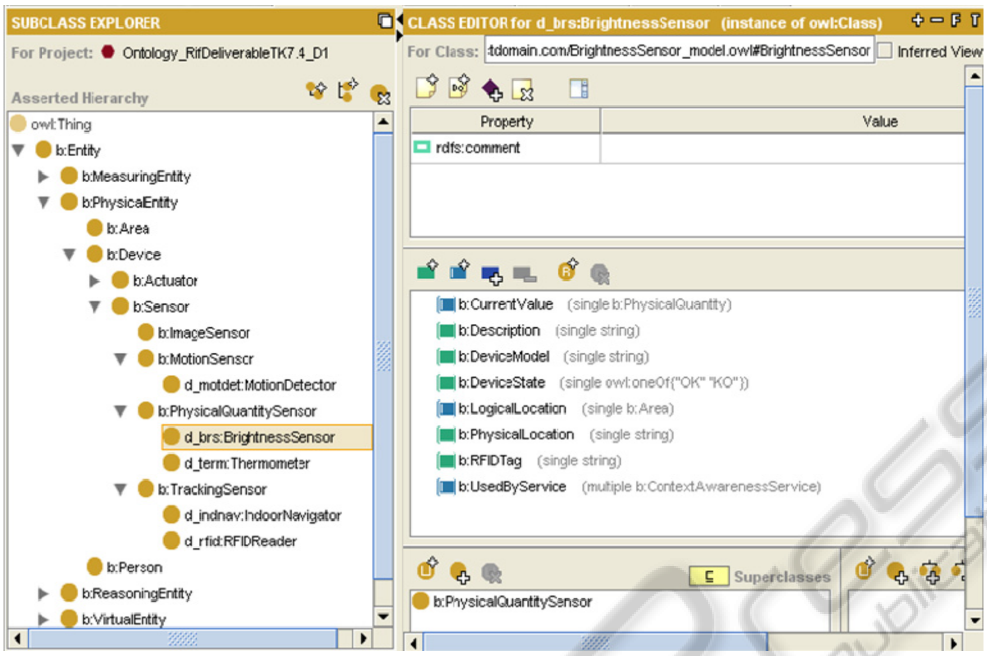

SUBCLASS EXPLORER
For Project: Ontology_RifDeliverableTK7.4_D1
Asserted Hierarchy
owl:Thing
b:Entity
b:MeasuringEntity
b:PhysicalEntity
b:Area
b:Device
b:Actuator
b:Sensor
b:ImageSensor
b:MotionSensor
d_motdet:MotionDetector
b:PhysicalQuantitySensor
d_brs:BrightnessSensor
d_term:Thermometer
b:TrackingSensor
d_indnav:IndoorNavigator
d_rfid:RFIDReader
b:Person
b:ReasoningEntity
b:VirtualEntity
CLASS EDITOR for d_brs:BrightnessSensor (instance of owl:Class)
For Class: tdomain.com/BrightnessSensor_model.owl#BrightnessSensor
Inferred View
Property
Value
rdfs:comment
b:CurrentValue (single b:PhysicalQuantity)
b:Description (single string)
b:DeviceModel (single string)
b:DeviceState (single owl:oneOf{"OK" "KO"})
b:LogicalLocation (single b:Area)
b:PhysicalLocation (single string)
b:RFIDTag (single string)
b:UsedByService (multiple b:ContextAwarenessService)
Superclasses
b:PhysicalQuantitySensor

Figure 4: A snapshot of the ontology.

defined models there are entities classified as:

- Physical entities such as various sensors, actuators, people;
- Virtual entities, such as external web services, e.g. a weather forecast web service;
- Reasoning entities, including reasoner and CA services.

The latter are used to convey information related to reasoning and context-aware services from the CA module to other modules. In fact, a reasoner module presents itself to the system through its logical models, which can be used for monitoring and configuration purposes. The context awareness module also instantiates another type of Reasoning entities, to represent the context awareness services it provides and to disseminate information about their state and about recognized context. When a context-aware service requests actuations for adapting to the context, it requests an update of a logical model, and the middleware validates the request, and dispatch it to the appropriate module (e.g. to the network layer to traduce the request in an action on a device). The entity hierarchy adopted in the XML schema is aligned to the class hierarchy in the ontology model.

4.1 Semantic Reasoning

The Semantic Reasoner module implements a semantic reasoning approach, takes care of maintaining a semantic representation of the system knowledge base (OWL-DL ontology), and perform rules-based reasoning for context recognition and adaptation. The semantic reasoner integrates Jena Semantic Web (Jena, 2009), using it in Microsoft .NET environment through IKVM.NET (IKVM.NET, 2010). Jena's APIs are exploited to create and manage the ontology model, to execute queries on it and to make inferences.

The XML schema and the ontology-based model are very similar, XML types are mapped on OWL classes, XML models are mapped on OWL classes, and the subclass hierarchy in the OWL ontology model is aligned to the complex types specialization defined in XML schema.

The two representations are maintained aligned to allow a seamless and automated update of the ontology based on the information, provided by other modules, about the current context. A snapshot of the ontology is shown in Figure 4, as displayed in Protégé (Protégé, 2010).

Context change events communicated to the semantic reasoner are queued for processing. When these events are handled, the knowledge base is updated accordingly. Sensed context change may require simply changing a property (e.g. currently sensed value), or can require creating/deleting individuals in the ontology. Prior to reasoning, management of context-aware services may be needed (in case new context-aware services are to be instantiated, or existing services are no longer needed). Some basic context information aggregations are typically performed too. For instance, an average value is computed from values measured from sensors of the same type in a certain area. To these purposes, SPARQL queries are used. Queries can be extended when new services are added and when a different aggregation is needed.

When the ontology has been fully updated, the rules engine is invoked to run the set of concerned rules (Jena SWRL-like rules). Rules are organized in two sets, Context Rules and Action Rules, for each service.

First, context recognition rules are run to recognize the current context of the system, in relation to existing services. Then, Action Rules are executed for decision making based on recognized context. Decided actions are written in the ontology along with justification to simplify understanding of why they were required.

Requests of decided actions affecting actuators are then handled and propagated to the involved device. In other cases decisions may simply concern providing recommendations suitable to the context.

Rules may also trigger internal events such as delayed rules execution (e.g. to check whether a certain context situation persists for longer than a certain time span). Rules are based on an expressive ontological model, which includes some information that is not present in the XML models and that is used only by the semantic reasoner. This knowledge concerns the relations between the system entities, e.g. between the context-aware services and the devices (properties set by the module itself), information related to rules and queries files and to other aspects pertaining the semantic reasoner only.

In order to be able to meet requirements of pervasive computing environments, the reasoning framework has been designed to manage dynamic changes in the knowledge base. The semantic reasoner module allows retrieving needed models, queries and rules from an external repository. The ontology model consists of a base ontology and of ontology models for specific entity classes. The semantic reasoner supports a dynamic management of the ontology and instantiates at run time in the ontology only device/service classes really present in the system at that moment. The set of classes present in the ontology is not pre-defined, since OWL classes are added at run time, only if needed, and removed when no longer useful. The ontology can be extended dynamically in terms of both individuals and classes, without need of recompiling the software module.

A remarkable aspect is that the semantic reasoner is able to deal with types of devices not known at system deployment time or with new context-aware services, provided that their models are given. When a new device appears in the system, if its ontology class is not yet present, the proper OWL model is downloaded from the model repository and added to the knowledge base, and finally the individual representing the device is created.

4.2 Event Processing

The semantic reasoning approach, though offering many benefits (in terms of expressiveness, use of formal and standard languages, clear separation of reasoning logic from application code), also has some notable limitations, e.g.:

- Significant memory requirements and performance times, usually not suitable to resource-constrained devices. For example, no framework with Jena equivalent reasoning capabilities is available to run on resource constrained devices. We are aware of prototypes like µJena (µJena, 2010) and Enhanced Micro Jena (Enhanced Micro Jena, 2010), but they seem not mature enough for commercial solutions and they lack some needed functionalities.
- Limited support for retraction in ontology-based reasoning. OWL-DL is mainly designed for monotonic inference, but context recognition is typically based on dynamic sensor measurements: as new measurements replace previous ones, previous measurements and derived information need to be retracted. Jena rules engine was originally designed for monotonic inference, non-monotonic inference is supported, but with some limitations (Jena-dev, 2008).
- No suitable support for uncertain and fuzzy data management and for managing data stream and temporal reasoning.

These considerations led us to design a flexible, plug-in-based architecture able to integrate, besides the pure ontology-based reasoner, other processing methodologies. Through the plug-in architecture and a late binding and reflection mechanism, multiple Processing Agents can be integrated to the reasoning framework (retrieving them either locally or from a remote repository).

Processing Agents have some capabilities of the semantic reasoner, namely the ability to communicate with the other modules through XML logical models and related events (they also presents themselves to the network through a logical model), and to filter events of interest.

In case of stateful processing, the Agents are able to keep in memory the data needed for their processing, e.g. a collection of XML logical models or derived structures. Moreover, Processing Agents can also raise “local events”, providing their processing results to other reasoning modules running on the same node, without propagating this information to other nodes.

Compared to the semantic reasoning approach, the Processing Agents can provide additional processing capabilities and perform efficiently a variety of complex tasks, if needed by integrating and exploiting specialized libraries or engines. We summarize below some processing tasks that could be performed by Processing Agents:

- A Processing Agent can be used as a light reasoning module on nodes where the semantic reasoner could not be hosted due to resource constraints. In this case, it provides standalone CA services, which could potentially be used by semantic reasoners on more powerful nodes for providing higher-level services.
- Another task that could be performed by a Processing Agent is the pre-processing (filtering, aggregating, checking, enriching, etc) on data which will then be used by the semantic reasoner. The Agent can provide simple data transformations, such as measurement unit conversions (especially in non-linear transformations), or can add reconfigurable capabilities that are difficult or impossible to implement in the semantic reasoner. For instance, the semantic reasoner doesn’t address sensor data uncertainty. In fact, in many cases, data coming from sensor networks are unreliable and affected by noise and errors and need some processing as a preliminary step to use them as context information.
- Specialized tasks for a Processing Agent could be given by the application of probabilistic and statistic methods, e.g. for managing uncertain and fuzzy data, making predictive evaluations on the basis of historical values, analysing user habits and preferences.

With the data centric communication style adopted in our platform and with the simple defined interfaces it is possible to build composite services, sharing basic computations blocks and their results, and exploiting the different capabilities of the agents and of the semantic reasoner.

The more recurrent tasks that will be needed in the system will be accommodated conveniently in ready-made reasoning blocks, but it will be possible to add other reasoning blocks when required, after the deployment.

5 CONTEXT-AWARE APPLICATION SCENARIOS

We briefly illustrate in this section how the reasoning modules have been exploited for building context-aware applications.

As a case study we developed a set of services for smart buildings, for monitoring of large facilities and public areas. As shown in Figure 5, we developed some services for a conference room, in particular for temperature management, for light management, for managing projection screen and lights in case of meeting, for fire prevention. We also developed some services for an exposition area, in particular to count the people present in the area itself, and to estimate waiting time for people in queue for accessing to the area.

Most of the services have been developed using the semantic module. As long as there are no strict response time constraints and computation needs are limited, in fact, the semantic reasoner can allow engineering context-aware applications easily, without coding. Therefore these rules-based applications can be developed by technical personnel such as technical matter experts of the domain without programming skills.

The queue waiting time service, on the contrary, was developed by using a more convenient, specialized processing agent.

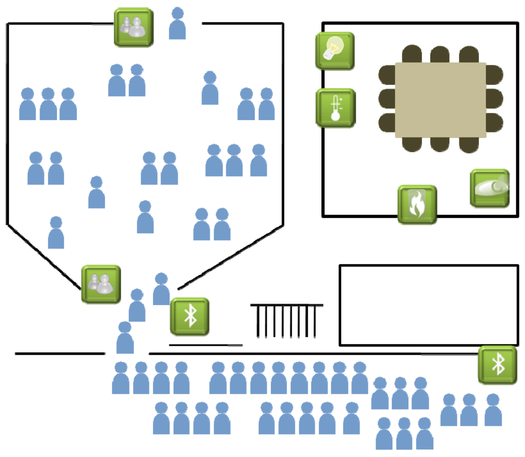


Figure 5: Services developed on a smart building.

The following paragraphs describe the development of the above applications and present related evaluation remarks.

5.1 Context-aware Application Development

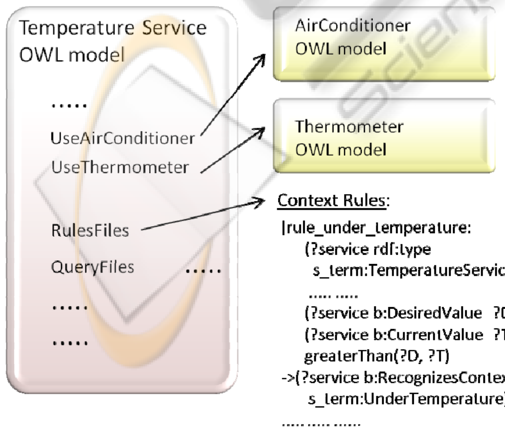
The services realized for the conference room (as well as the people counter service for the exposition area) have been developed through the semantic reasoner. Devices involved in these services are lamps, brightness sensors, thermometers, air conditioners, projectors, curtains, white screens, and people counters devices. People presence and regulations directly requested by user were considered for regulating room conditions.

The middleware provided a convenient support, and allowed the context-aware application developer to focus on the application logic, without need of caring of communications, localization, networking, and so forth. Involved entities are handled easily thanks to the abstracted logical models.

To add a CA service based on already connected devices it is needed to (see Figure 6):

- Add an OWL model of the service.
- Provide the pair of rules sets for context recognition and action selection.
- Extend queries related to aggregation of sensor values, and to conditions for considering the service 'available'.

If the location area includes more than one sensor of the same model, e.g. more than one brightness sensor, the system aggregates through SPARQL queries the measures coming from the sensors, providing an aggregate value, e.g. an average.



```
|rule_under_temperature:
    (?service rdf:type
      s_term:TemperatureService)
     ..... .....
    (?service b:DesiredValue  ?D)
    (?service b:CurrentValue  ?T)
    greaterThan(?D, ?T)
->(?service b:RecognizesContext
      s_term:UnderTemperature)|
..... ..... ......
```

Figure 6: Adding a new CA service.

In case devices to be used by the CA service being developed are not yet 'known' to the system it is needed to add also their models (XML and OWL) into the model repository, so that the middleware will be able to handle them. At run time, entities are dynamically instantiated when present, and also CA services are instantiated when instantiation conditions for the type of service are met. The relationships between a service and devices and entities it reasons about are described in the ontology models. When a CA service of a new type is instantiated, related rules and queries are loaded and made available for processing.

The applications above described benefit from the ontology-based models supporting the context recognition, which required considering the state of different devices and entities. Rules expressed context conditions and adaptation and not many computations were needed.

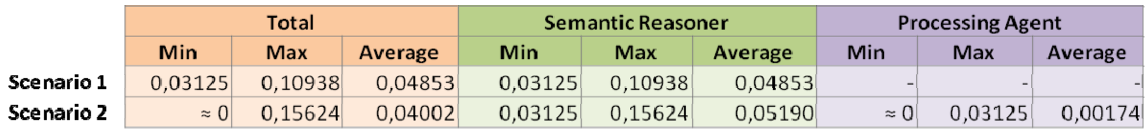
The CA service for deriving the average time that people spend in a certain waiting area, and for estimating waiting time for incoming people, was on the contrary developed through the processing agent's framework. The sensed data is given uniquely by a single entity, which provides tracks of portable Bluetooth enabled devices held by visitors, but some correlations, computations and statistics are needed, especially for waiting time estimation. Thus, we deemed more appropriate to develop it in form of a processing agent. It required some programming, but more complex aspects are already provided by the middleware and by the reasoner host. It meant specifying filtering condition and coding the ad-hoc processing, and was quite straightforward thanks to the provided processing agent's framework.

5.2 Evaluation and Remarks

As we could observe by the case study indicated in section 0, the semantic reasoner demonstrated to be able to support development of context-aware application based on models and rules. It allows dealing with complex domain models.

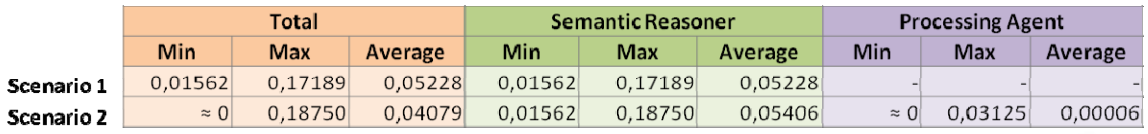
In particular the following points are worth noticing:

- The extended syntax provided by ARQ, Jena's SPARQL processor, has been used in some cases for querying the knowledge base instead of the standard W3C SPARQL syntax, to exploit some operators not included in SPARQL 1.0 but supported in ARQ. For instance, aggregation through 'GROUP BY' and negation were used, while they are still in



	Total			Semantic Reasoner			Processing Agent		
	Min	Max	Average	Min	Max	Average	Min	Max	Average
Scenario 1	0,03125	0,10938	0,04853	0,03125	0,10938	0,04853	-	-	-
Scenario 2	≈ 0	0,15624	0,04002	0,03125	0,15624	0,05190	≈ 0	0,03125	0,00174

Figure 7: Comparative test results.



	Total			Semantic Reasoner			Processing Agent		
	Min	Max	Average	Min	Max	Average	Min	Max	Average
Scenario 1	0,01562	0,17189	0,05228	0,01562	0,17189	0,05228	-	-	-
Scenario 2	≈ 0	0,18750	0,04079	0,01562	0,18750	0,05406	≈ 0	0,03125	0,00006

Figure 8: Comparative 5 hrs test results.

working phase in SPARQL W3C working group for the 1.1 version.

- The extendibility of Jena has been used to add a few procedural primitives which can be used in the rules. These built-ins were needed as the set of Jena primitives is limited and some required operators, e.g. for certain operations with dates and time, were not provided in the framework.
- We had no issue concerning the usage of Jena in .NET environment through the IKVM virtual machine, even in introducing extensions. We used a library we assembled from the Java version of Jena, not the recently published Jena .NET library version (Jena .NET Framework, 2010).

The applications developed on the semantic reasoning module were used to show information on a dynamic graphic user interface, tested both in a simulated environment and in a smart ambient equipped with real devices, and function properly.

Even if the semantic reasoner worked well with the required CA services we noticed that other kinds of data processing would be very complex to develop in form of rules-based applications. For example, the implementation of the queue waiting time estimation service in a rule-based application is not straightforward. Even with the support of Jena framework, we would need to develop specialized built-ins for statistical computation and to use a persistence system for keeping track of visitors' movements.

The processing agent infrastructure resulted beneficial in dealing with complex data processing. The pluggable processing modules can integrate reasoning framework capabilities, if needed by also including specialized libraries. Moreover, the processing agents and the semantic reasoner can co-exists and their processing can be staged in multiple ways, and allow CA service composition based on hybrid methods.

In the case study described in section 0, we performed some comparative tests over the following scenarios:

- Scenario 1: conference room management services and people counting service in the queue for the exposition area. A rule-based approach is adopted (semantic reasoner only).
- Scenario2: all services of scenario 1, plus the queue waiting time service. A hybrid approach is adopted (semantic reasoner plus the processing agent for waiting time estimation).

Figure 7 shows, for both scenarios, the minimum, maximum and average reasoning times, related to the semantic reasoner only, the processing agent only, both the semantic reasoner and the processing agent (total reasoning times). The tests were conducted in a simulated environment, on a PC with Windows XP, 1.99 GB of RAM, 2 GHz CPU.

We can observe that the processing agent has very good reasoning time performances, and does not increase significantly the total reasoning times. We also considered the semantic reasoner performances as a good result, for the need of the application developed.

Figure 8 shows the results of the same kind of tests, performed over a time period of 5 hours. We can observe that the longer test period did not significantly affect the reasoning time performances (the average reasoning time of the processing agent even decreased).

Another benefit given by the processing agents approach is that a processing agent can also be ported to other platforms, including embedded platforms, thus overcoming a limitation given by using a resource demanding semantic framework.

Finally, we claim that the criteria related to flexibility, simplicity and maintainability, defined in (Kapitsaki, 2009) and referenced in Section 2, are well satisfied by our solution.

6 CONCLUSIONS AND FUTURE WORK

The paper presented a flexible solution for the creation and management of context aware pervasive and IoT-enabling applications. The solution is based on a modular and extensible platform, providing an effective support for context-aware application deployment, from hardware to network and middleware layers. The CA component is given by a flexible framework that allows integrating different reasoning methods (from the semantic approach to event-processing techniques), according to application needs.

A case study analysis highlighted the advantages offered by the solution, which allows reducing the context-aware application engineering effort and complexity through the various provided methods and instruments, which can be flexibly combined according to application needs.

As a future work, we plan to further investigate methods and tools for complex event processing (e.g. for data streams, temporal reasoning, and uncertain data management) and statistical analysis, evaluating possible tools integration, e.g. complex event processing engines (Esper, 2010) (StreamInsight, 2009), and/or new techniques development in our system.

ACKNOWLEDGEMENTS

The authors gratefully acknowledge partial funding of this work by Regione Piemonte within the frame of the research project “Piattaforma Tecnologica Innovativa per l’Internet of Things”.

REFERENCES

Devices Profile for Web Services (DPWS), 2009. *http://docs.oasis-open.org/ws-dd/ns/dpws/2009/01*.

Enhanced Micro Jena, 2010. *http://www.nembes.org/*.

Esper, 2010. *http://esper.codehaus.org/*.

Garofalakis, M., Brown K. P., Franklin M. J., Hellerstein J. M., Zhe Wang D., 2006. Probabilistic Data Management for Pervasive Computing: The Data Furnace Project. *Bulletin of IEEE Computer Society Tech Committee on Data Engineering,* 29(1), 2006.

Huang V., Javed M. K., 2008. Semantic Sensor Information Description and Processing. In *SENSORCOMM'08, 2nd International Conference on Sensor Technologies and Applications*.

IKVM.NET, 2010. *http://www.ikvm.net/*.

Jena, 2009. *http://jena.sourceforge.net*.

Jena-dev, 2008. *http://tech.groups.yahoo.com/group/jena-dev/message/32818;http://tech.groups.yahoo.com/group/jena-dev/message/43618*.

Jena .NET Framework, 2010. *http://www.linkeddatatools.com/downloads/jena-net*.

Kapitsaki G., Prezerakos G., Tselikas N., and Venieris I., 2009. Context-aware service engineering: A survey. In *Journal of Systems and Software*, Vol. 82(8), 2009.

Kjær, K. E., 2007. A survey of context-aware middleware. In *SE'07, Proceedings of the 25th IASTED Conference on Software Engineering*. ACTA Press.

Kostelník P., Sabol T., Mach M., 2008. Applications of Semantic Technologies in AmI. In *Ambient Intelligence Forum 2008 Conference Proceedings*.

Leger A., Heinecke J., Nixon L., Shvaiko P., Charlet J., Hobson P., Goasdoue F., 2008. Semantic web take-off in a european industry perspective. In *Semantic Web for Business: Cases and Applications*. IGI Global.

µJena, Context-ADDICT, 2010. *http://poseidon.ws.dei.polimi.it/ca/?page_id=59*.

Perttunen M., Riekki J., Lassila O., 2009. Context representation and reasoning in pervasive computing: a review. In *International Journal of Multimedia and Ubiquitous Engineering*, Vol. 4(4), 2009.

Protégé, 2010. *http://protege.stanford.edu/*.

Roman M., Hess C., Cerqueira R., Ranganathan A., Campbell R., Nahrstedt K., 2002. Gaia: A middleware infrastructure to enable active spaces. In *IEEE Pervasive Computing*, 2002: 74-83.

Soylu A., De Causmaecker P., Desmet P., 2009. Context and Adaptivity in Pervasive Computing Environments: Links with Software Engineering and Ontological Engineering. In *Journal of Software*, Vol. 4(9), 2009.

Strang T., Linnhoff-Popien C., 2004. A Context Modeling Survey. In *1st International Workshop on Advanced Context Modelling, Reasoning and Management*.

StreamInsight, 2009. *http://www.microsoft.com/sqlserver/2008/en/us/r2-complex-event.aspx*.

Truong H., Dustdar S., 2009. A Survey on Context-aware Web Service Systems. In *International Journal of Web Information Systems*, Vol. 5(1), 2009.

W3C Semantic Web, 2001. *http://www.w3.org/2001/sw/*.

Wolf P., Schmidt A., Klein M., 2009. Applying semantic technologies for context-aware AAL services: What we can learn from SOPRANO. In *INFORMATIK 2009, Lectures Notes in Informatics*, Vol. P-153.

XSD, 2004. *http://www.w3.org/TR/2004/REC-xmlschema-0-20041028/*.