Reasoning with Modular Ontologies for Context-Aware Applications

Alberto Garcia-Sola, Teresa Garcia-Valverde and Juan A. Botia
Facultad de Informática, Universidad de Murcia
Espinardo, 30100, Murcia, Spain.
Emails: {agarciasola, mtgarcia, juanbot}@um.es

Andrés Muñoz
Computer Science Department, UCAM University
Email: amunoz@ucam.edu

Context awareness is a key feature of adaptive applications and services developed on a ubiquitous computing infrastructure. Typical middlewares for managing context information rely on a centralized server to receive, process and deliver context information. However, such an approach has many drawbacks. Among them, we can mention that rule based reasoning is not flexible at all. In this paper, a framework for the management of context information is presented. It is based on Semantic Web technologies for the representation of context, reasoning and inferencing over such information. The particularity of the presented framework is that it is distributed. The Semantic Web based ontology is partially distributed, being the rule sets used for personalized context reasoning and inferencing spread over the context information consumers. With such an approach, adaptive services and applications in ubiquitous computing scenarios can be developed easily. As a consequence, this framework paves the way for more powerful, flexible and easy to develop and deploy adaptive ubiquitous computing applications.

ACM Classification: *1.2.11*

Keywords: *Semantic Web, Distributed Inference, Context Information Streams Management, Argumentation, SOA*

1. Introduction

Improvements and progressive miniaturizations of electronic devices have enabled smart environments and smart homes to grow in popularity and implementation. Costs of these devices, especially sensors and actuators controlled by small processing units have been drastically reduced, and these systems are now affordable for average families to install in their homes. However, although the hardware is already cheap and discrete enough to be installed in homes, there is still a big gap to be filled which is that of the software that controls this hardware.

Nowadays, a very important trend is that of context-aware systems (Schilit *et al*, 1994). These systems are able to interact with the user in a non-intrusive way (usually without the need of any direct interaction from the user). The key to this interaction is an appropriate context information

Manuscript received: 13 November 2012
Communicating Editor: Rafael Valencia-Garcia

management. By context, we refer to any information that characterizes a situation related to the interaction between humans, applications, and the surrounding environment, as stated by Dey and Abowd (2001).

We can highlight several interesting points from context-aware systems. On the one hand we have the context itself. On the other hand there is the collection and distribution of this context, the use of this context for certain tasks, and finally we have the interaction of sensors and actuators with the real world. By sensors we mean devices that provide us real-world information, such as temperature (temperature sensor), image (camera surveillance) or a motion sensor. Sensors are usually the most primitive context producers. Actuators interact with the real world, and may be something like a cell phone (they can communicate with us by text, images and sound and light elements), a light bulb, an air conditioning system, a system for closing and opening doors or a small robot.

Actuators might be indirect context consumers, although they are usually connected to the system through adaptation services which are the gateway between the system and actuators. Those adaptation services are context consumers which, depending on the context information, will interact with the actuators. Those services can run on powerful desktop computers or in small devices with few processing capabilities. Therefore, the simpler and the higher order the context information is, the better for those devices; their logic and understanding will be simpler, and their processing needs smaller.

In ubiquitous computing context, management is a key task. An easy access, management and modification of this context information enables us to develop more powerful Context-Aware tools in a faster and easier way. In an environment with producers/consumers of context, if we also want to modify the logic (i.e. behaviour according to a received context) of consumers in Context-Aware systems, we have several alternatives. First, we can modify all context consumers in order to interpret the context differently. But this is usually not desirable since it makes us dependent on the presentation of data, and we must modify all consumers in that context and adapt them. Moreover, in many cases these consumers are small adaptation services that only spread small actions when they get very specific indications, for example, a service that sends a signal to turn on the light to an actuator when receiving context 'Light On' and turns it off when it gets 'Light Off'. That is all the processing performed and all the context information of interest. Another option is to modify all context producers to produce information adapted to the needs of consumers. Both options need to modify the code of the devices to adapt them to the changes in the system logic. Moreover, the logic of the system does not lie in the system, but is interpreted by each customer of the system. To avoid this we can move part of this adaptation logic to another separate layer that can be manipulated independently of the code of the devices. This layer will generate higher level information from the produced context by the system sensors using Context-Reasoning.

The Context Reasoning consists of obtaining new information from existing information by inference. This new information may be implicit, or defined for a particular domain. If the information was implicit, it already existed in the system. Therefore, the system may make it explicit by means of Context Reasoning. For example, if the system has the information that *George is taller than Paul, Paul is taller than Andy* and that *is taller than* property is transitive, then we can infer that *George is taller than Andy*. In contrast, this new information can be generated based on certain rules or patterns established explicitly for a particular domain. For instance, this allows us to define for a specific domain within an intelligent building that *if the computer's room temperature is higher than 20°* then *temperature is high*. We have a service that sends an alarm to a technician when it receives

high temperature. However, this information will depend on the domain, and may even vary over time. Thus, if we want to adapt the system for a room where food is kept frozen, we would only have to change this rule domain to suit the ideal temperature for this food.

Context Reasoning usually produces information of the same or higher order than the original. In this way we can chain several simple events that generate more complex information in order to get a more complex and higher level combined information. This idea is used in CEP (Complex Event Processing) systems, where the CEA pattern (Condition-Event-Action) is used. *Condition* models the situation of interest to be recognized, *Event* is the event that causes the new situation and *Action* is the action in response to that situation.

This Context Reasoning process is performed using some concrete techniques with a context represented in a particular way. Depending on the semantic richness that we want to achieve in the context representation, we can opt for different solutions: in the form of Java objects (e.g. ContextJ (Appeltauer *et al*, 2011) or JCAF (Bardram, 2005)); in the form of strings with a proprietary language; in structured languages such as XML (Goldfarb and Prescod, 2000) or using Semantic Web technologies like RDF (Brickley *et al*, 2004) or OWL (McGuinness *et al*, 2004). The more semantic richness the context has, the more information will be available to perform context reasoning and infer new information. The reasoning process depends heavily on the context representation.

Reasoning in computer science has been understood traditionally using rule-based systems. These consist of a **set of rules**, an **inference process** and a **working memory**, as we can see in Figure 1. Knowledge is stored as rules of the form:

IF *some condition(s)* ***THEN*** *some action(s).*

The Semantic Web reasoning offers some advantages over the traditional reasoning. It provides a very powerful expressivity. Besides, it allows us to make implicit reasoning (Ontology Reasoning or axiomatic), so that doing a good definition of the ontology we get a lot of information without having to make it manually explicit.

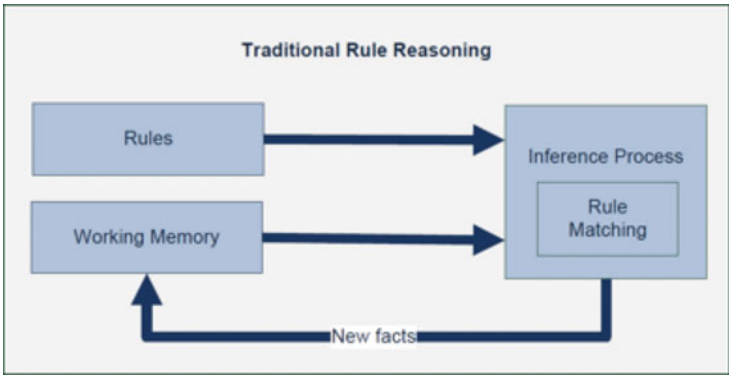


Figure 1: Traditional Rule Reasoning

Taking advantage of this technology we created OCP (Open Context Platform). OCP is a platform for the management of context that allows us to reason (García-Sola *et al*, 2010b) with this context using Semantic Web technologies and rules introduced at the beginning of its execution, so generating new context information. However, this monolithic approach had some disadvantages.

These rules cannot change during the execution of the system, whereas in actual systems it is common for the logic of the application to change at any time. In addition, everything is centred on the server, with the correspondent overload when these rules grow in number and complexity.

To solve these problems we propose a distributed and modular reasoning architecture based on ontologies. This proposal offers several advantages. We will be able to modify the adaptation logic on-the-fly seamlessly. This will be done by sets of patterns or rules to be evaluated dynamically and concurrently taking into account interdependencies. These sets of patterns and rules can be interrelated and in turn concurrent, thus achieving higher scalability and modularity. Moreover, this logic is decentralized, so that, for instance, the person responsible for maintenance of a building may be ordered to control all lighting in the common areas of a building, whereas a researcher can control lighting from his office, both within the same system.

To achieve this we have made changes in the internal structure of the OCP reasoning module. We have moved the reasoning system towards a mixed SOA (Service Oriented Architecture). Thus, part of the reasoning is done efficiently within the system itself to perform the dynamic part (e.g. context processing such as rules or pattern recognition) outside of it. The biggest challenge is getting a very small overhead for both the main system and the services, both in transfer of information and execution time, because we need the information to be reasoned very quickly, as it is of vital importance in Context-Aware systems. This architecture allows us to design compatible platforms that can provide multiple services. The system architecture is explained in depth in Section 2.

In addition to structural changes, since we are working with a service-oriented architecture, synchronization, temporalization, conflicting contexts and cycle problems with very specific particularities arise. The system may get information that has been generated before other information, later than the information that was generated afterwards. However, the system should be as reactive and fast as possible, so when new information arrives in the system it must send it without waiting until it gets the information that was generated prior to this. A deeper insight into the problem and the solution we have adopted can be found in Section 3.

To show the applicability of the system and how it works in a real working environment we have described an example in Section 4. In Section 5 we have performed a survey of different approaches that have been made to address this problem by comparing with other solutions, showing the advantages and disadvantages. Finally, in Section 6 some conclusions have been drawn as well as our planned future work, which is outlined.

2. Distributed reasoning

In context-aware environments the system behaviour depends on the context. And in many of these environments the context may change very quickly, and hence the system behaviour too. As discussed in the Introduction, the context moves the direction of the system behaviour, but this direction is defined by the domain rules that adapt the behaviour of a system by interpreting a specific context of a particular domain. This is why not only will the context vary rapidly, but also the sets of rules of the domain will define the business logic of the system. The rule set is dynamic and subject to addition of new rules or modification and deletion of existing ones. For example, preferences of users change or a company extends its installations. Additionally, they may depend on new devices coming into the system, since some rules may only make sense when they are within the environment. For instance, imagine a set of rules of a user to set the temperature in his office. These rules make no sense if he is not working.

This shows us that the system is not an isolated agent, but an entity that interacts with other agents which determine the system behaviour. That is, the system behaviour is distributed among the agents of the system, as it is in a naturally distributed system.

The system distribution can be performed at different levels. By distributed reasoning we understand that the OCP reasoning is not performed within the system, but rather in a collaborative way between the different agents using the system. Thus, by distributing the reasoning process, the context processing will be carried out outside the system. As part of this processing we find the rule reasoning, reasoning on top of that as the pattern recognition and others that can generate new information from existing information, i.e., processing information to find new information. There are several reasons for performing the reasoning of the system distributedly:

Dynamicity and decentralization: Since the reasoning process is usually performed dynamically and in a decentralized manner by those who need it, when not available and therefore that reasoning is not necessary, it would not take place.

Efficiency and scalability: Being the reasoning process performed on the user's machine.

High availability and security: By keeping the OCP Core more compact and simple, there are fewer chances of bugs and crashes. And letting the user execute their rules, no security issues concerning rules will appear.

Rules grouped by producers: Being rules (which may only apply to themselves) defined, executed and consumed by the user, performance increases, avoiding server latency and interaction.

Response time will increase due to the fact that load is decreased.

Versatility and **modularity** are obtained by distributing the system. The system can be extended through different types of reasoning not limited to rules. i.e. pattern recognition, which does not have to do with rules; or even the use of any condition we choose arbitrarily, such as external tools or queries on the internet. But more interesting is that all services can take advantage of the expressivity and reasoning from the Semantic Web technologies, needing only to define the information they are interested in. Finally, **privacy and mobility**. The user himself keeps his preferences in the form of rules, visible only by himself (not even the server). Furthermore, if they travel to another location, their information is kept with them. Therefore, there is no need to interchange information from different systems at different locations. And, thus, it is assured that the information will be available at any time at different locations.

To explain how we have made the reasoning distribution, it is necessary to describe how OCP works. OCP is a framework for context management which offers a server and two interfaces for producers and consumers of context. Context producers must only notify the OCP server of the new context, and it will be responsible for performing reasoning on this context, and if a consumer is interested in this context or in part of the new context generated, the OCP will notify it. The way in which a context consumer tells the server about its interests is through a subscription to the context of OCP. Consumers can choose from subscribing to only one part of the context (notify about the modification of a particular entity) to subscribing to the whole context generated in the system. However, this notification is not based on a direct matching, but uses the Semantic Web technologies. Therefore, if we subscribe to notifications about an entity called *Person*, the server will not only notify us of changes to the entity *Person*, but also of all instances that are of type or supertype *Person*, as could be an instance of type *Worker* that would be subclass of *Person*. This, as will be explained, is a big advantage when it comes to the distributed reasoning. But not only ontologies are useful for subscription. Reasoning with ontologies offers a handful of benefits with

respect to traditional reasoning. To name a few, reuse, web compliance, standardization, high expressivity, interoperatibility, formality (e.g. logic programming), evaluation of its quality (Duque-Ramos *et al*, 2011). A more detailed survey can be found at Oberle (2009).

The Context Reasoning allows OCP to infer new context information from existing information in systems using ontology reasoning. Context rules (defined in the SWRL[1] standard) will allow defining rules to generate knowledge based on existing information.

Ontologies provide a lot of information, but much of this information is implicitly defined. This information is defined using the mechanisms of the Semantic Web technologies such as two entities class-subclass relation or transitivity. Reasoners built on top of the ontology use all of that information, so they need to have access to that implicit information somehow. This implies that a rule reasoning using ontologies usually requires axiomatic reasoning which gives the rule reasoner implicit information back when the reasoner queries the ontology. This presents us with a dependency of all data in the ontology, since some information not appearing in the rule may affect the activation of this rule. Therefore, it seems that even if a reasoner is interested only in a small item of context information it will need the whole ontology.

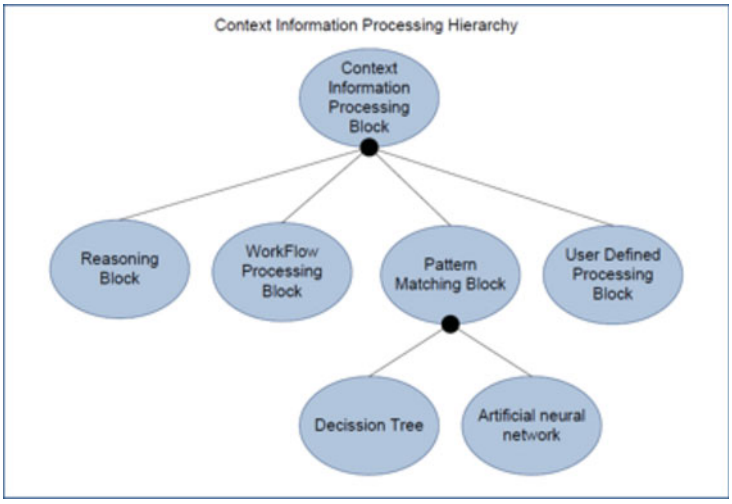


Figure 2: Context Information Processing Hierarchy

This dependence on all the data from the ontology can be avoided. To do so, we need some way of knowing what is the exact information needed to be retrieved, when we need it and when we do not handle all the ontology.

The system is divided into the **OCP Core**, which is responsible for maintaining working memory, and the **Context Information Processing Blocks (CIPBs)**, responsible for carrying out the context information processing. These blocks can be organized hierarchically depending on the type of processing they do. In Figure 2 our proposed hierarchy can be seen, although it could be organized in any other way. It includes pattern recognition, rule reasoning and other techniques such as decision trees. These Context Information Processing Blocks (CIPBs) will initially receive some information through a bootstrapping process (which will be explained later). Once they have done

1. http://www.w3.org/Submission/SWRL/

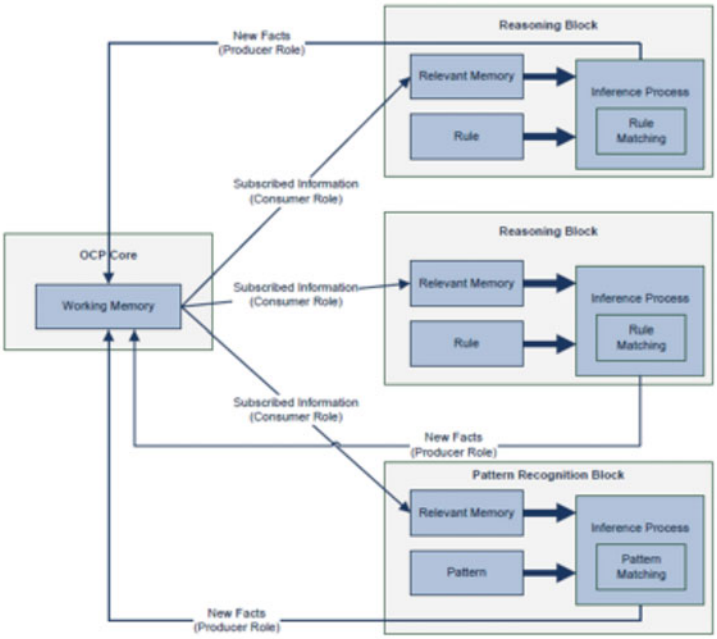


Figure 3: Distributed Reasoning Architecture

the bootstrapping process they will start receiving a continuous stream of data from the OCP Core with the information they are interested in and they will return the new facts inferred from the fired rules. In Figure 3 we can observe how the distributed system is in contrast to the traditional system that OCP was working with (Figure 1).

When new context is introduced in OCP, as already commented, this notification is done taking into account the Semantic Web. Using this property of OCP, for a CIPB (with consumer role) it would be enough to subscribe to the entities that are involved. For example, in the case of Reasoning Block, it would be enough for it to subscribe to the antecedents of the rules set. OCP will be responsible for notifying when any of these entities changes (whether directly or indirectly) so that the reasoner knows if the antecedent of the rule is satisfied and the rule should be fired.

Another advantage of using this interface is performance. As can be seen in García-Sola *et al* (2010a) the overhead introduced by adding new producers or consumers is virtually zero. There we can also find more information about the overall performance of the system with the chosen expressivity (OWL DL), as described in the following. This makes it an ideal choice for extending the system efficiently, freeing the server from this burden and any faults that could cause these extensions. Thus, the server remains compact, efficient and safe. We could even test new CIPBs without jeopardizing the central server, and not even having to stop to perform these tests. If there were any failure it would occur in the part of the CIPBs, but the server did not have to change anything.

2.1 Service Oriented Architecture

In order to achieve modularity and connectivity between different parts of the system we assume an SOA architecture (Service Oriented Architecture) in which the services are dynamic. Services

may come and go at any time, connecting with the system dynamically and flexibly. This architecture allows us to connect and update different parts or modules of the system (i.e. services) dynamically, and can be used locally or remotely.

To use an SOA architecture, the way the reasoning process is carried out needs to be restructured, since so far all the information from the ontology was needed to make this reasoning locally and monolithically (centralized). However, this approach of having the whole ontology replicated throughout is not feasible in a distributed architecture. In this work we propose an architecture to achieve distributed processing using the OCP capabilities to deliver context, as required, to remote entities via the internet. In this new architecture the rules sets can be added, modified or deleted dynamically, achieving a change in the logic of the system without having to modify application code or interrupt the application. These context processing processes, whether rules, patterns or other user-defined processing are services that can group a set of these, and in turn, be interconnected with other services, so that together they make up the global context information processing.

In the system, each of these processing services is a type of **Context Information Processing Block (CIPB)**, which will define and perform on-line processing tasks, either by pattern recognition, one or more rules or any process defined by the CIPB. For example, when using rule reasoning they are **Reasoning Blocks**.

2.2 Plug-and-Play Reasoning Blocks

So far, rules definition was performed using ORE-GUI[2], which added these rules to the ontology. ORE-GUI is a graphical tool for defining and modifying rules developed at the University of Murcia. This tool has been modified in such a way that is compatible with OCP (García-Sola *et al*, 2010b), to directly add rules to the system. When OCP was run, these rules were loaded and checked when new information was introduced in the system. This meant that there was no need to modify or create new code to add new rules, but it was necessary to stop the system and insert a higher load to it (new rules to check when new context arrives in the system). With the new architecture there is an initial service (OCP Core) that connects to different context information processing services (i.e. CIPB) as they are created. To make the whole process once the system is installed without writing a single line of code, we opted for the automatic creation of Reasoning Blocks. These Reasoning Blocks are created from the rules defined in ORE-GUI, and are directly pluggable with OCP. Thus once the system is established, the user does not need any advance information on the structure of it to modify its business logic, a task that can be performed by an average user of the system without advanced computer skills.

2.3 Efficient Local Reasoning

Ontologies are a common way of representation of information in Semantic Web technologies. They are defined as a 'formal, explicit specification of a shared conceptualization' (Fensel *et al*, 2001). OCP uses the Web Ontology Language (OWL) (McGuinness *et al*, 2004) as the ontology language. Depending on the demanding expressiveness, OWL provides three increasingly expressive sublanguages. We have chosen OWL DL, which supports the maximum expressiveness without losing computational completeness (all entailments are guaranteed to be computed) and decidability (all computations will finish in finite time). OWL DL is so named due to its correspondence with description logics, a field of research that has studied a particular decidable

2. http://ore.sourceforge.net/

fragment of first order logic. OWL DL was designed to support the existing Description Logic business segment and has desirable computational properties for reasoning systems. In Description Logics, a distinction is drawn between the so-called TBox (terminological box) and the ABox (assertional box). In general, the TBox contains sentences describing concept hierarchies (i.e. relations between concepts) while the ABox contains ground sentences stating where in the hierarchy individuals belong (i.e. relations between individuals and concepts).

To further improve system performance, Reasoning Blocks do not include the full ontology, but rather a small part of the TBox (i.e. the ontology schema), corresponding to both the antecedent and the consequent of the rules and the ABox (i.e. instances of this schema are changed in OCP), corresponding to the rules antecedent. This implies that the partial ontology in the Reasoning Blocks can be in an inconsistent state because it is a partial copy of the global ontology, maintained by OCP. But for the reasoning we are performing that is not relevant, because these inconsistencies and potential derived problems are handled by the OCP Core (see Figure 2). Therefore, we get the same results as if we had all the information, because we have all the relevant information for that particular reasoning.

Most of the reasoners allow us to make a matching of the antecedent without the complete ontology scheme, further enhancing the performance and the need to exchange information. Some of them do not even need to use an ontology for this reasoning. However, the reasoning would still be a Semantic Reasoning. This is because the OCP Core offers the service of axiomatic reasoning. This system allows us to use tools not originally intended for the Semantic Web technologies, but by exploiting its potential. For example we could use the EulerGUI engine to transform SWRL to a different rule engine, such as Euler[3], cwm[4], Drools[5] (Van Hille *et al*, 2012) or Fuxi[6]. All this means that when a new Reasoner Block wants to start working it should only retrieve from the OCP Core existing information relative to its rules antecedents, with some information being unnecessary such as ontology schema or the rest of instances. Otherwise there would be a lot of overhead when the ontology would acquire a considerable size.

2.4 OCP Architecture

So far we have explained how to perform the reasoning and processing of context from the point of view of the CIPBs, which interact with OCP Core. In this subsection we describe the overall system architecture. This includes how the interactions between CIPBs and OCP Core are performed. OCP currently offers the possibility to subscribe to an unlimited number of consumers to receive context, and producers to send context. When a new CIPB wants to act as a context processing unit it needs to subscribe as a consumer to be aware of the ontology changes. But in turn it must also be a producer in order to notify the OCP Core of the changes that occur when rules are triggered or new information is generated. But this is not enough. As we have said, in the bootstrapping process of the CIPBs, the CIPBs need to get part of the existing information from the OCP Core. This information includes the relevant part of the ontology to make a proper assessment of all relevant rules or patterns. Therefore, initially a CIPB import some initial information from the OCP Core corresponding to its subscription of context. This bootstrapping uses the OCP Context Discovery service, which allows clients to retrieve existing context in the system.

3. http://www.w3.org/2001/sw/wiki/Euler
4. http://www.w3.org/2000/10/swap/doc/cwm.html
5. http://www.jboss.org/drools
6. http://code.google.com/p/fuxi/

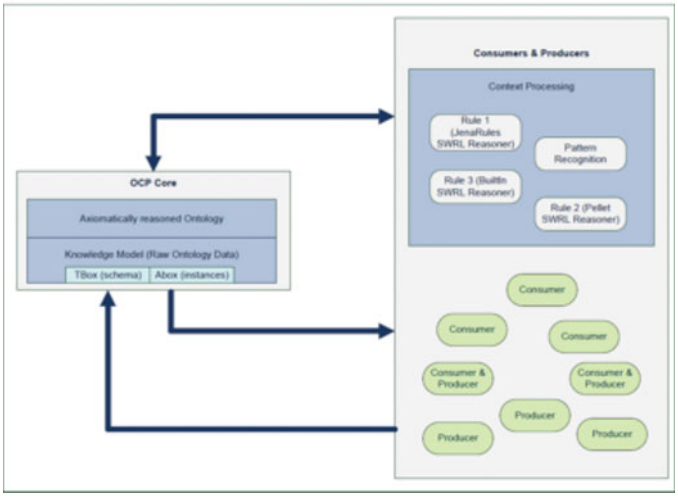


Figure 4: OCP Architecture

Thus, the architecture is as seen in Figure 4. As we can see, the OCP Core stays small, and the different reasoning processes are kept out by using the same interface as consumers and producers.

This fully decoupled architecture is thought to be extended with new types of higher-level reasoners. As it is proposed, a logical layered reasoning architecture can be established, in which reasoning can be used to produce higher-level reasoning. The current situation is a special case in which the axiomatic reasoning is available for everyone as an internal OCP service and the rule reasoning benefits from this. This logical architecture can be seen in Figure 5. The upper layers are

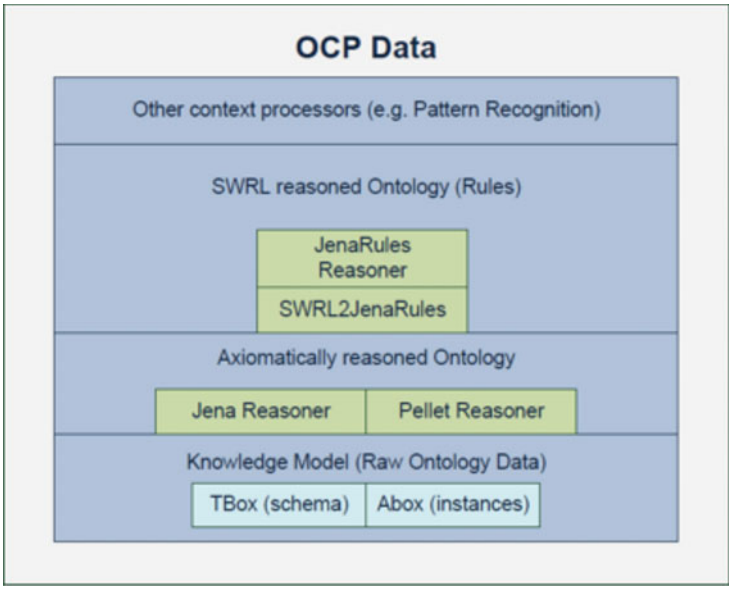


Figure 5: Layered reasoning

actually completely decoupled and can be separated without changing the OCP logic, as it is possible to add new reasoning layers that are based (or not) in these ones. Even if we would like to provide the whole reasoning process on the same machine this architecture offers some advantages, since the OCP Core is more robust, because of its being simpler, allowing us to exploit more efficiently the parallelism when reasoning, and extend it with no need to restart the system.

We have to keep in mind that in this new architecture the reasoning process is decoupled and distributed, even between different machines. This can cause some problems, because the reasoning is now no longer generated when new context information has just been introduced. This will be seen in the next Section.

3. Issues in Distributed OCP

Having distributed reasoning offers many advantages, but there are also issues to be addressed to use the system smoothly.

Basically we find and tackle three problems. On the one hand we have the problem of cycles, whereby a Reasoning Block produces, for instance, some information that makes another Reasoning Block to produce other information. That new information from the second Reasoning Block is as that which made the first Reasoning Block to produce its initial information. This would lead to an infinite loop between these two Reasoning Blocks. On the other hand there is the issue of temporalization. Since we are in a distributed system, it is possible that network delays or a slow reasoning process make a reasoned fact which was generated before another fact, reach the system after the other, so overwriting its value, when it should be the other way round.

Finally, we can find situations where the information is uncertain, incomplete or inconsistent. Therefore, in order to disambiguate such confusing situations, we propose an argumentation process (Carbogim *et al*, 2000). Thus, an inference process is performed taking into account the possible conflicts that may arise. These conflicts are then solved by argumentation techniques (Muñoz and Botía, 2008).

3.1 Dealing with Cycles

Herein we will explain the solution for the **problem of cycles**. To do this we need to explain what information is stored and exchanged between OCP and the producers and consumers. When a producer sends new context to the OCP Core, the OCP Core always stores a **timestamp (TS)** and a **unique identifier (OID)**. Additionally, it stores in an archive the entire context that has been inserted in the system. Thus, although the value of certain entities is modified, we can know the previous values or the whole state of the system at a particular time.

To mitigate the problem of cycles we have used a new metainformation element (apart from the TS and the OID). When a producer sends some context information to the OCP Core it should attach the so-called **derived identifiers (DID)**. The DID will only be used when the information is created out of some other information (i.e. reasoned information). The DID value is the set of OIDs of the context items which caused the reasoning to be fired. If the context is not reasoned or derived from another previous context (i.e. created by the producer itself, as done by a sensor), the DID value will be zero or null. If the DID is not null, the timestamp will be inherited from the parents. If there is more than one parent, the timestamp value will correspond with the oldest one, which is the time when the reasoning was made possible. However, the OID may be different, to uniquely distinguish it within the system, and it will be assigned by the OCP Core.

With this system we can use this information to see where the derived information comes from, and merely by looking at the DID field, be able to know whether it is derived or not. Although the value of the context information of an element with a particular OID is changed to another value, the value of that element for a particular TS is stored in the OCP archive. So using the TS we can know the value it had when this information was generated. When new information arrives at the OCP Core it is sufficient to look at the derivation trace (using the DIDs) to know if a cycle has been reached. At this time the OCP Core will stop sending new information to consumers interested in it. We can see this in Figure 6. As we can see, initially the OCP Core sends context information to *Reasoning Block A*, which generates new information with the DID of the context that produced it. We must remember that the Reasoning Blocks do not assign the OID, but rather OCP Core does, otherwise we could not ensure they are unique. As *Reasoning Block B* is interested in this new context, OCP sends it the information. The same applies to *Reasoning Block B*, which generates new information, and sends it to the OCP Core, which in turn resends it to *Reasoning Block A*. Eventually *Reasoning Block A* generates some new inferred information again and sends it to the OCP Core. As the OCP Core receives $t_2 = v_2$, it looks at the derivation trace and finds that it already had $t_2 = v_2$ previously with the same timestamp and, besides, both with the same DID. The OCP Core detects the cycle and stops spreading context.

When a cycle is detected, a confusing situation is generated: different Reasoning Blocks are producing information which causes a situation in which it is difficult to know the valid final information. Therefore, when a cycle is detected and OCP stops sending such information, an argumentation process is raised in order to solve the conflict and keep the ontology in a valid state. Argumentation process will be explained in Section 3.3 in depth.

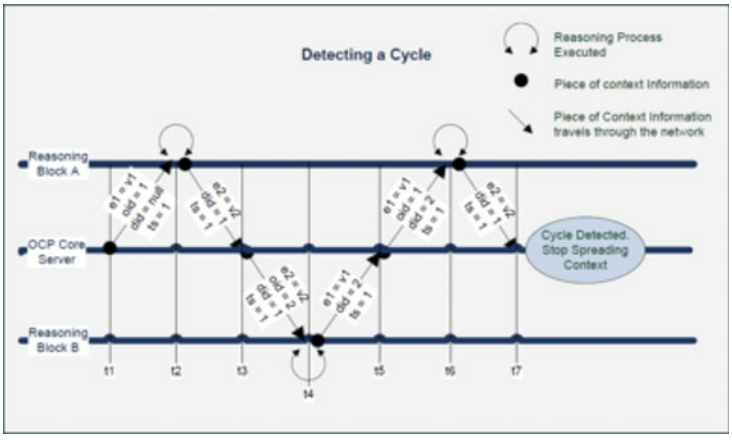


Figure 6: Cycle Detection

Moreover, this solution allows us to add a desirable and useful utility. We can see the derivation trace of the reasoning processes, and explain what has caused the generation of an item to be able to explain certain facts.

3.2 Temporalization Issues

The temporalization issue is more difficult to solve. In fact there is no perfect solution. Consider this example (Figure 7):

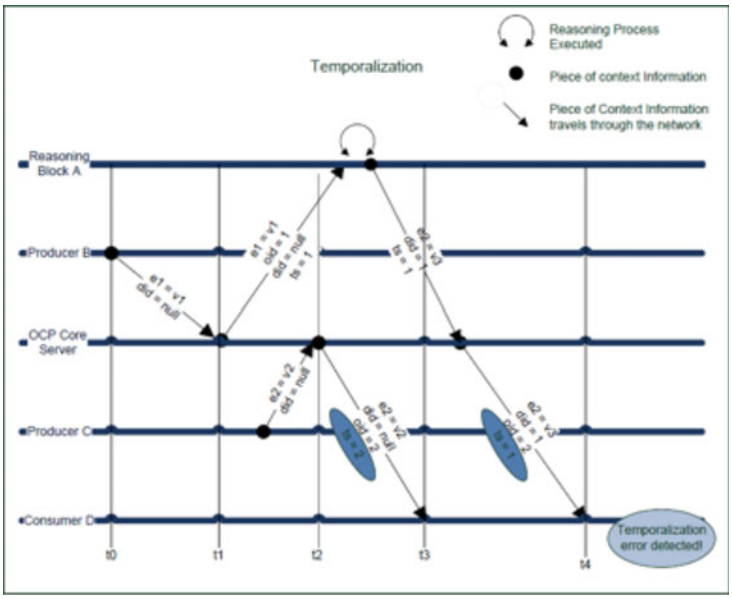


Figure 7: Temporalization Errors

There are two producers in the system (*Producer B* and *Producer C*), a Reasoning Block (*Reasoning Block A*) and a consumer (*Consumer D*). *Producer B* produces a new value v_1 in t_0 for the e_1 entity. This value comes to the OCP Core at t_1 time, which is the timestamp to be recorded. This value is sent to listeners interested in this entity (as *Reasoning Block A*). Meanwhile, at t_2 the OCP Core gets a value of v_2 for the entity e_2 from the *Producer C*, so the OCP Core assigns the value of v_2 to e_2 and sends it to consumers interested in e_2 (*Consumer D*). Finally, at t_4 *Reasoning Block A* has inferred further information about e_2, v_3. However, this information comes derived from the value of e_1 at t_1 (i.e. v_1, with OID 1), therefore, the timestamp of v_3 is t_1, i.e. prior to the current value of e_2 (which is t_2). Since *Consumer D* is interested in e_2, the OCP Core sends it that information as well. We have reached a situation in which the OCP Core has received inferred context information which is older than the information that it currently has in the system. Since information just received in the OCP Core is older than that stored, the OCP Core cannot overwrite it and change its current value, because the information already stored is more recent. However, this information could be interesting for other Reasoning Blocks which use that to infer other information, or it may be an important and specific value which some consumer was waiting for to begin executing a routine. That is, we cannot use this information to change the current value because it is older, but we cannot discard it either because it may be important for some consumers.

If we decide to wait for the OCP Core to receive all the inferred information from the different Reasoning Blocks before sending anything to the consumers some problems may arise. We do not know whether they are going to reason something or not, so the OCP Core would need some mechanism to be aware of when they have finished reasoning. This would mean an increase in latency, since the OCP Core should delay the delivery of information, although it is possible that there is no such inference. It will break the efficiency and parallelism achieved by distributing it. Conversely, the OCP Core does not really know whether they are really done with reasoning, and even if they communicate that to the OCP Core, it should not wait, as the system reaction time

with the users would be lost. The other option is to immediately send all the information the OCP Core receives, so that consumers have the information as quickly as possible. But as we saw in the aforementioned example, it may involve sending information with an earlier timestamp of already sent information. This breaks the temporalization line; no longer does everything come to us in order. However, this problem only occurs when sent information is inferred, and therefore, derived from other information that already exists in the system. This happens because the OCP Core is the one which assigns the timestamps. Therefore, it could never occur when the context is generated directly by the producer.

The proposed solution has been to opt for an intelligent function that determines whether a reasoned context is interesting to the consumer. The non reasoned context, which does not create this problem, will always be sent if the consumer is subscribed to the context. If the consumer needs to know all the information, the consumer will be aware of the entire context. Thus, we created three operating modes for consumers who are actually affected. Depending on the working mode they choose, the OCP Core will send them one piece of information or another. In the case that they do not choose any, for backward compatibility the standard mode is assigned. Three working modes are now shown:

- **Safe:** In this mode a reasoned item of information is only sent if no context is received in between the timestamp of the reasoned item and the current timestamp. We may lose some information, but everything we get will be in order.
- **Standard:** This will be the default option, and in this mode everything will be sent to the consumers unless something related with the inferred information arrives in between. That is, if the OCP Core gets new context about (for instance) *Car* and then some context inferred about *University* with an older timestamp, it will be sent to consumers because they are unrelated. Note that, still, all of the derived information is to be generated, even if some item is not sent to the listeners, because Reasoning Blocks operate in expert mode and, thus, receive all the information. Therefore, no information is lost.
- **Expert:** In this mode all the context information to which the consumer is subscribed is sent, even if the context does not arrive in order. The client is responsible for handling it, because with the TS and the DID they will be able to know when the information was received and whether it was direct or inferred. This is the default mode for the Reasoning Blocks, because they need all the information to generate new information.

In the example in Figure 7 the value v_3 for e_2 will only be sent if a listener were in expert mode, because the system has received v_2 for the entity e_2 at t_2 which is illegal in safe and standard modes. In most cases customers will no longer be interested in such information since it is something that has happened already and they have the latest information.

3.3 Argumentation-based Reasoning System

Using contextual information from different sources and making reasoning and inferencing over such information implies a lack of information to draw up some complex contexts. For example, imagine an office building in which there is a rule which establishes that in any empty room the air conditioning should be switched off. However, a manager wants to set the temperature of his office to 25° and he is out of his room for whatever reason. Conflict arises in that situation.

One way of solving such confusing situations is based on argumentation (Rahwan and Simari, 2009; Bench-Capon and Dunne, 2007). Argumentation systems differ from other knowledge

systems in the capability of generating justifications – arguments – to support inferences which can, potentially, be inconsistent. The use of criteria and heuristics allows decisions to be taken on any inconsistency detected by establishing which argument is more believable.

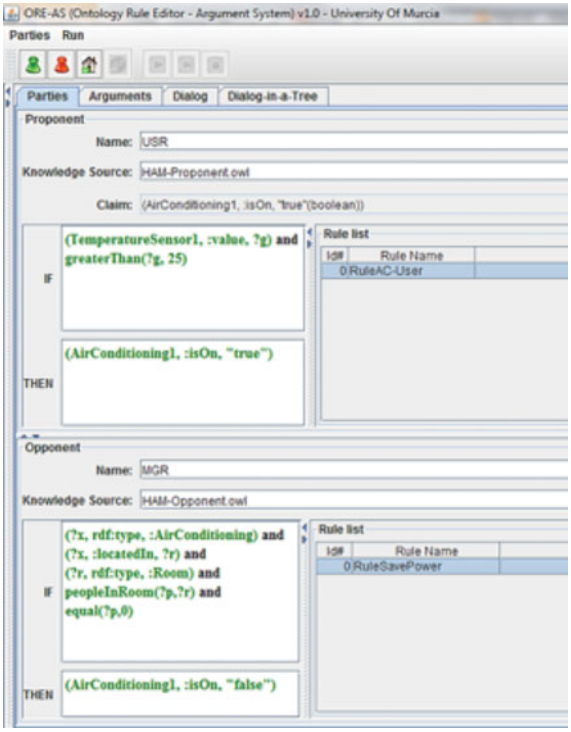


Figure 8: ORE-AS

Thus, an Argumentation-based Reasoning System (AbRS) is proposed in this work to disambiguate such confusing situations. In order to deal with the argumentation system described above we use ORE-AS[7], a tool which has been developed at the University of Murcia (see Figure 8). ORE-AS allows simulating argumentation process between a proponent agent (an agent proposing the acceptance of a fact or the execution of an action in the state of affairs) and an opponent agent (an agent opposing that fact or action). The simulation shows the arguments created by both agents and the persuasion dialog between them, whose result (proponent wins and its proposal is accepted by the opponent, or opponent wins and the proposal is discarded, or no decision can be reached) depends on the criteria adopted. Here we describe ORE-AS as the AbRS implementation, while the theoretical models that underlie the conflict detection and argumentation process are explained in Muñoz and Botía (2008, 2009).

Two kinds of conflicts are considered in the system: semantic conflicts and conflicts caused by the cycles. Semantic conflicts can be *semantic-independent conflict*, so-called contradictions, and *semantic-dependent ones*, or differences. Contradictions may appear regardless of how the domain is modeled, for example, a fact and its negation.

Differences are dependent on the domain considered, for example, different values for the same entity which are received at the same time. The conflicts caused by the cycles are discussed in Section 3.1. When a cycle is detected, the OCP Core stops sending new information to consumers. Then, an argumentation process is raised in order to solve the conflict.

Imagine the example of the office building described above. Suppose that there are two agents: USR, representing the manager's agent, and MGR, representing an agent responsible for controlling the power consumption in the building. We have the following scenario: the manager is out of his room for whatever reason, and his agent USR wants to switch on the AC device in the manager's office since the temperature therein is over 25°. However, MGR opposes this action since its policy is to switch off AC devices in empty rooms and this agent detects that nobody is in the manager's room. As a result, a conflict arises between these two agents.

7. http://sourceforge.net/projects/ore-as/

The conflict is detected by the OCP Core when different values for the AC are received at the same time (semantic conflicts). In that moment, OCP notifies the Reasoning Blocks involved in the conflict of the situation. OCP provides each Reasoning Block with the information it needs about the other so that they can communicate between themselves and start the argumentation process.

In order to obtain a solution, this conflict is simulated in ORE-AS. USR takes the proponent role with its rule stating that if the temperature sensor in the manager's room (T*emperatureSensor1*) has a value greater than 25°, then the AC device in that room (*AirConditioning1*) must be switched on. This rule, named RuleAC-User, can be seen in the Proponent box shown in the upper part of Figure 8. On the other hand, MGR takes the opponent role with its rule stating that any AC conditioning (represented with variable *?x*) located in an empty room (represented with *?r*) must be switched off. This rule, named RuleSavePower, can be seen in the Opponent box shown in the bottom part of Figure 8. Moreover, in this scenario ORE-AS is configured with the criterion *RuleAC-User* < *RuleSavePower*, indicating that RuleSavePower is preferred to RuleAC-User in cases of conflict.

The persuasion dialog resulting from the argumentation process is shown in Figure 9. First, URS claims that *AirConditioning1* must be on. Then, MGR asks for reasons for this assertion. In that moment, USR answers with argument Arg_USR_1 (not shown in the figure). This argument for switching on the AC device is supported by the fact that the temperature is over 25° and by the rule RuleAC-User. Next, MGR rebuts – attacks – such an argument with its own argument Arg_MGR_1 (not shown in the figure) supporting the AC device to be switched off. This argument is justified by the fact that the manager's room is empty and by the rule RuleSavePower. Since the adopted criterion establishes this rule as preferred to RuleAC-User, and USR has no more arguments to support its claim, this agent concedes MGR counterclaim and retracts its own claim. As a result, USR loses the dialog and its proposal of switching *AirConditioning1* on is discarded. Therefore, the argumentation-based process offers a valid alternative to provide sensible qualitative reasoning in the presence of inconsistent and incomplete situations.

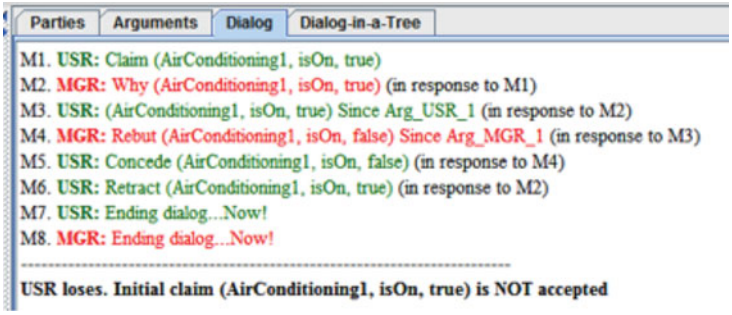


Figure 9: Argumentation process

4. Example

In this section we describe an example of practical application of our system running in a real environment. This scenario has been deployed in an intelligent building located close to the University of Murcia, and although it is still in the testing phase the results are very positive. Next, in Subsection 4.1 we will describe the details of the software implementation of our system. Then, we will discuss how it is deployed in a real environment in Subsection 4.2.

4.1 Implementation Details

The definition of the system described in Section 2 tells us how different elements interact with each other in the system. But it does not describe specific technologies, leaving them to our election.

Everything we have done is connected to OCP. All parts of the OCP system are coded in Java, as well as the different used technologies. However, when using SOA architecture we are free to write the different services using other programming languages.

Initially we will start by describing the implementation that we used for the SOA architecture described in Section 2.1, which is closely related to 2.4. In order to achieve the modularity and connectivity between different parts of the system we use OSGi. OSGi (Open Services Gateway Initiative) is a technology that allows us to design compatible platforms that can provide multiple services.

This technology allows us to connect and update different parts or modules of the system (called bundles) dynamically, and in its latest version (OSGi R4.2 (Alliance, 2008)) where the Remote Services can be used remotely and, therefore, we can use different virtual machines. The system will use bundles to perform reasoning tasks. These bundles are connected with the OCP Core to exchange information. Thus, each of these reasoning bundles corresponds to the Reasoning Blocks described in the architecture. In fact, OCP is also integrated in the system as a bundle.

Another important point of the implementation are the reasoners included in the Reasoning Blocks as well as the creation of the Reasoning Blocks. For the implementation of the Reasoning Blocks we have chosen to develop a tool that automatically generates the OSGi bundles. This tool is an extension of the ORE-GUI tool. Figure 10 shows this tool.

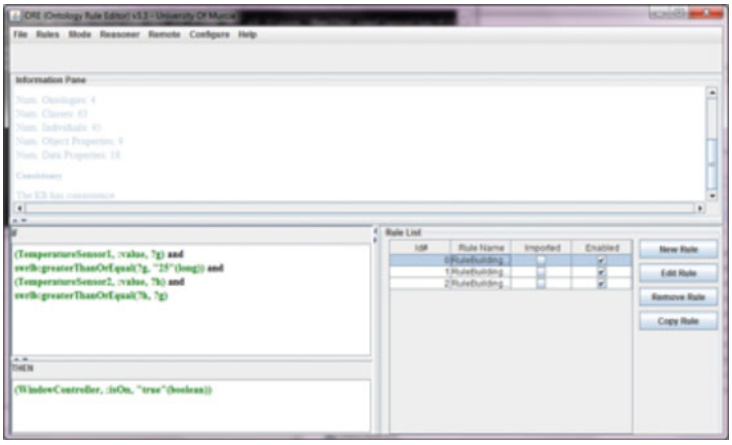


Figure 10: ORE-GUI tool

ORE-GUI connects to the OCP Core to load the ontology and let the user define the rules. The rules are defined in a graphical, friendly and easy way. Figure 11 shows how a rule is defined using the ORE-GUI. As the figure describes, using the ORE-GUI a rule is defined in three simple steps and it is possible to navigate through the ontology to obtain all the needed information. Thus, once the rules have been defined with ORE-GUI the user can create a bundle easily.

In addition, the user can choose from several rule reasoning engines. The default engine used is JenaRules, which provides very efficient reasoning without requiring the whole ontology to be

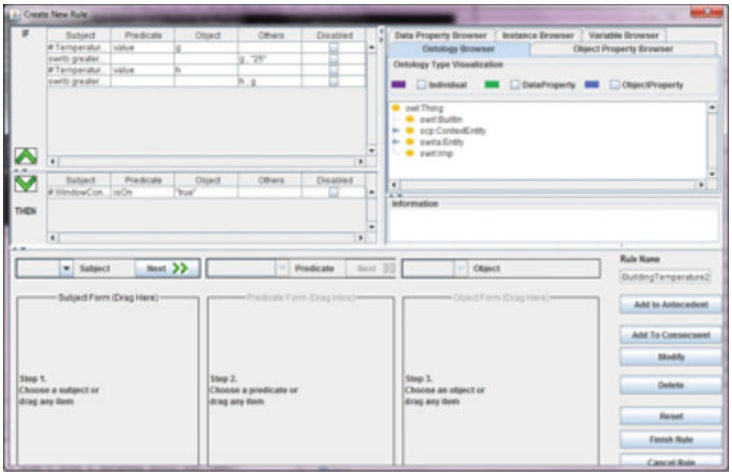


Figure 11: Defining rules with ORE-GUI

present. Each of these bundles includes the code generated to connect to the OCP Core, the chosen rule reasoning engine and the rules themselves. Thus we can choose to directly connect the bundle to the system as a Reasoning Block or modifying the code to suit it to our needs.

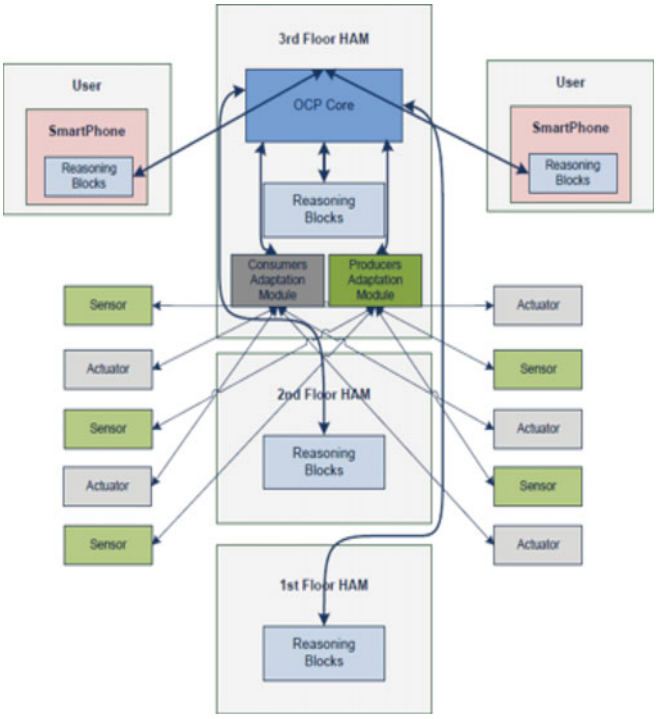
4.2 Deployment Scenario

The system deployment has been done using the architecture depicted in Zamora-Izquierdo *et al* (2009). This system in currently being used in an intelligent building located close to the University of Murcia. The main element of the architecture is the **Home Automation Module** (HAM). This comprises an embedded computer which is connected with all appliances, sensors and actuators. In addition, some users are equipped with state-of-the-art SmartPhones connected to the HAMs.

The system is deployed in a three-storey building. Each floor has a Home Automation Module (HAM). They get context information from a wide range of sensors as well as from the users. For instance, there are RFID antennas which record the users' locations (who are equipped with RFID tags). They can also control the automatic opening of windows, switch on the air conditioning to the desired temperature, or close/open the blinds according to the desired light intensity before using artificial lighting.

The current architecture of the system is as depicted in Figure 12. We have installed the OCP Core in one HAM located on the third floor. This is the main HAM, which is connected with the other two HAMs of the lower floors. This HAM, as well as the other two, has a Reasoning Block within it. Each HAM has rules exclusively regarding the floor they are installed on. Thus, a HAM controls its own floor. In the main HAM there is another Reasoning Block, which encloses rules which relate to the whole system, such as security (which has to be managed globally). Finally, each user with a SmartPhone connected to the system has some other Reasoning Blocks installed in their devices. In this way they can manage their own rules that only affect themselves.

In the main HAM we can also see two adapters for both the sensors and the actuators. They are just a small piece of software that gather the information from the sensors and send actions to the actuators, according to the context given.



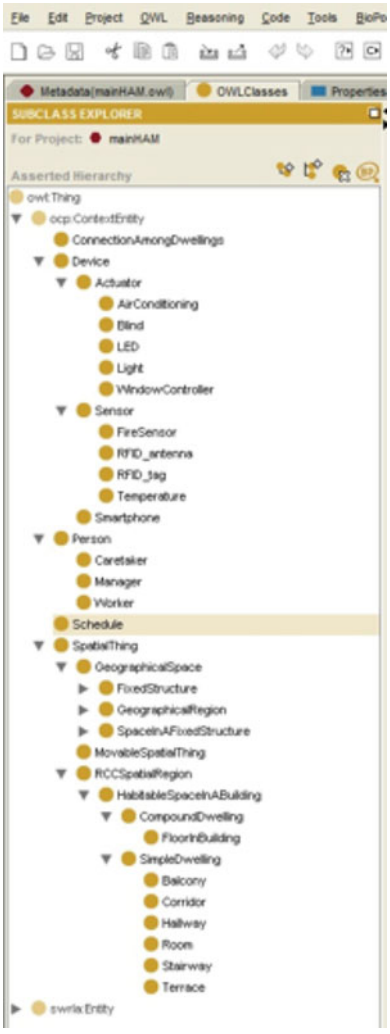


Figure 12 (above): Deployed architecture in the Intelligent Building

Figure 13 (right): Ontology

Since the OCP Core is installed in the HAM of the third floor, it contains the whole ontology. Figure 13 describes the hierarchy of classes of the global ontology, maintained by OCP.

However, as described above, the other HAMs and the users do not need so much information. They keep a partial ontology in the Reasoning Blocks. The global ontology can be found at the following links: http://webs.um.es/mtgarcia/ontology/mainHAM/, and examples of the partial ontologies, such as the partial ontology of the HAM on the second floor: http://webs.um.es/mtgarcia/ontology/floor2HAM/ and the ontology of the Reasoning Bock installed in the SmartPhone of the worker Stephen Raymond: http://webs.um.es/mtgarcia/ontology/user1/. All these ontologies have been simplified for a better understanding of the example.

As in most context-aware systems, rules in the system have been introduced trying to make users' daily tasks easier in a non-intrusive way. Some global rules have been included in the system, as may be done in other context-aware systems. For instance, if the temperature inside the building is above 25° and the temperature outside is lower, the windows are opened. Otherwise, the air conditioning is switched on.

$$R_{MainHAM_1}: \begin{array}{l} \text{value}(TemperatureSensor1, ?g) \\ \wedge\ \text{swrlb: greaterThanOrEqual}(?g, 25) \\ \wedge\ \text{value}(TemperatureSensor2, ?h) \\ \wedge\ \text{swrlb: lessThan}(?h, ?g) \\ \Rightarrow\ \text{isOn}(AirConditioning, true) \end{array}$$

$$R_{MainHAM_2}: \begin{array}{l} \text{value}(TemperatureSensor1, ?g) \\ \wedge\ \text{swrlb: greaterThanOrEqual}(?g, 25) \\ \wedge\ \text{value}(TemperatureSensor2, ?h) \\ \wedge\ \text{swrlb: greaterThanOrEqual}(?h, ?g) \\ \Rightarrow\ \text{isOn}(\text{WindowController}, true) \end{array}$$

They try to diminish the power consumption. Lights and the air conditioning are automatically turned off when there are no workers on the floor:

$$R_{MainHAM_3}: \begin{array}{l} \text{sqwrl: isEmpty}(Person) \Rightarrow \text{isOn}(WindowController, false) \\ \wedge\ \text{isOn}(AirConditioning, false) \wedge \text{isOn}(Ligth, false) \end{array}$$

Since all of these rules are common to all floors they are introduced in the Reasoning Block of the main HAM. Nevertheless, rules which affect only one floor will be included in the HAM of that floor. Thus, if something happens on one particular floor (for instance, the power supply goes off), the rules stop working. Only the rules which are needed remain active. This makes sense, since there is no need to keep them working if there is nothing to interact with. If we had all the rules in a different location, and something like this happens, all the rules regarding the blackout location would keep running, but they would be useless.

This same principle of decentralization is applied to users. The worker *Stephen Raymond* has set up in his user profile that he wants his room to be always at 21°. Therefore, the system should turn the air conditioning on at that temperature when he is in the building, so as to have the desired temperature when he comes into his office. In his SmartPhone he has set up a Reasoning Block with this rule:

$$R_{User1_1}: \begin{array}{l} \text{locatedIn}(?tempSensor, Office7) \\ \wedge\ \text{value}(?tempSensor, ?g) \\ \wedge\ \text{swrlb: notEqual}(?g, 21) \\ \wedge\ locatedIn(?airCond, Office7) \\ \Rightarrow\ \text{isOn}(\text{airCond}, true) \end{array}$$

We should note that he does not specify his location. But since the Reasoning Blocks of the SmartPhone are only running when they are connected with the system, they are only applied when the user is in the building. This makes the rules definition easier to the users.

Imagine now that in the example of 4.1 the project manager sets up a meeting regarding a concrete project and he wants all the workers to attend it. He puts it into his calendar, writing down the date and time, marking it as important, and labeling it with the project name. This information is automatically sent to the OCP Core (as described in that section), which saves that context and sends to consumers.

The regular temperature in the system is 23° for the common areas, but since the project manager has set up the temperature at 21°, 10 minutes before the meeting the temperature of the meeting room is set to 21°. The following rule describes this situation:

$$R_{Floor2HAM_1}: \begin{array}{l} \text{isPlaced}(?m, MeetingRoom2) \\ \wedge\ \text{status}(?m, "Open") \\ \wedge\ \text{swrlb: dateTime}(?presentTime) \\ \wedge\ dateTime(?m, ?dateMeeting) \\ \wedge\ \text{swrlb: subtractTimes}\begin{pmatrix} ?presentTime, \\ ?dateMeeting, 10 \end{pmatrix} \\ \wedge\ locatedIn(?tempSensor, MeetingRoom2) \\ \wedge\ \text{value}(?tempSensor, ?g) \\ \wedge\ \text{swrlb: notEqual}(?g, 21) \\ \Rightarrow \text{isOn}(\text{AirConditioning1}, true) \end{array}$$

Furthermore, just a few metres before the first worker enters the room, the lights are turned on:

$$R_{Floor2HAM_2}: \begin{array}{l} \text{isPlaced}(?m, MeetingRoom2) \\ \wedge\ \text{status}(?m, "Open") \\ \wedge\ \text{swrlb: dateTime}(?presentTime) \\ \wedge\ dateTime(?m, ?dateMeeting) \\ \wedge\ \text{swrlb: subtractTimes}(?dif, ?dateMeeting, 10) \\ \wedge\ \text{swrlb: lessThanOrEqual}(?dif, ?presentTime) \\ \wedge\ \text{participant}(?p, ?m) \\ \wedge\ \text{locatedIn}(?rfid, MeetingRoom2) \\ \wedge\ \text{detectedBy}(?rfid, ?p) \\ \Rightarrow \text{isOn}(LightMeetingRoom2, true) \end{array}$$

In this rule, the light should be turned on a few metres before the first worker enters the room. Nevertheless, it should happen only if the time of the meeting is close (less than 10 minutes).

Likewise, it would be done in a similar manner for setting alarms an hour before the meeting, muting the phone during the meeting and removing these rules once fired and the meeting is over.

Finally, consider the hypothetical situation that one of the workers is unable to attend the meeting because he is on a research visit for three months at the hypothetical University of OCP. This system is also installed at the University of OCP. Since he is not in the building of the meeting, those rules will not be fired for him. However, when this worker enters his new assigned office in that University he checks that the temperature and light intensity is the same as in his home university.

Since his personal preferences are set up in his SmartPhone, when he enters a new place, all of his preferences are loaded into the system of that location. Actually, all of his defined Reasoning Blocks are connected with the OCP Core of that building. As we can see rules and preferences of users go with them. If they travel to another location, we do not have to transfer all of those rules, which may be outdated, but rather the user itself keeps them. It is the user who uses them and knows when they need them. All the problems regarding transfer of information, privacy or similar are wiped out. The user is the one who keeps his own internal rules which nobody knows, and decides when to plug them into the system.

We can draw some conclusions from this example. As we have seen, using this distributed approach, rules are not only organized and located regarding their nature and range of use, but also active just when they are needed. Therefore, new reasoned context is only introduced in the system when it is needed. Only common information is managed at all times. The system does not have to manage a vast variety of information and rules from users which are not using it all the time. Typically only a small subset is used, and it will depend on the users, not on the system.

Therefore we place those rules in the users, plugging them into the system when they are needed.

OCP Core leverages the system users to avoid high loads being distributed through them. If we had a centralized system, all the rules from all possible users should be running all the time, even if they use the system just a few days a year. Using this decentralized architecture the system only runs the rules it needs.

5. Related Work and Discussion

There has been a lot of work related to Context-Aware frameworks. The authors of Hong *et al* (2009) show a view of context-aware focus growing in journals from 2000 to 2007. Context-aware articles have grown considerably since 2000, up to seven-fold more in 2007. There are several platforms and middlewares for context management with diverse approaches. Not all of them are based on the same principles. In Context-Aware systems the representation of the context is a key-factor. This representation will determine other aspects of the system such as the reasoning techniques to apply. In Hong *et al* (2009) and Strang (2004) we can find a detailed survey of some of them, including different approaches (e.g. object-oriented models, ontology based models, key-valued models). They both conclude that ontologies are the most promising alternative.

There are several techniques to reason using Semantic Web technologies. The most basic is usually a must in every Semantic Web system, and is axiomatic reasoning. This reasoning lets the user access all the information that is implicit when an ontology is defined, as explained in the previous sections.

The other reasoning techniques work on top of that. The most well known is SWRL (Horrocks *et al*, 2004), which stands for Semantic Web Rule Language. This Rule Reasoning Language offers a standardized way to define rules to infer new information from given information.

Regarding the use of reasoning in Context-Aware systems, there is also a wide range of works. A variety of context reasoning schemes have been proposed (Schmidtke, 2012) to deal with context reasoning, including Bayesian networks (Ma *et al*, 2008), Dempster-Shafer Theory (Zhang *et al*, 2009), logic-based (Hu *et al*, 2012) and ontology-based context reasoning schemes among others, but only ontology-based reasoning schemes incorporate semantics into context representation and reasoning.

Though we do not strictly modularize ontologies, related work in this field is important to fully understand the process we follow. A few books such as Stuckenschmidt *et al* (2009), Del Vescovo *et al* (2010) and Hitzler *et al* (2011) may help get a better understanding on modularization of the ontology. In this work we have focused on the research of how to get Semantic Web reasoning distributed. Although several works have been proposed to distribute classical reasoning, little information can be found about distributed Semantic Web reasoning. Most focus on different ways to distribute description logic (Batsakis *et al*, 2011; Baader *et al*, 2007), but by using an approach quite distant from ours.

In Serafini and Homola (2010) we can find a recent overview of some of the techniques used in Distributed Reasoning in the Semantic Web technologies. The past ten years have seen several proposals of logics to define a formal semantic of the integration of modular ontologies. The most influential formalisms of modular ontologies for the Semantic Web technologies have been those based on Description Logics (DL).

Drago (Serafini and Tamilin, 2005) addresses the problem of reasoning with multiple ontologies interconnected by semantic mappings. Bridge rules map concepts of a source ontology into

concepts of a target ontology. DDL has been improved (Homola and Serafini, 2010) to support subsumption propagation through chains of bridge rules.

ε-connections (Cuenca Grau *et al*, 2011) is a framework for combining non-overlapping ontologies encoded in Description Logic but possibly also other logics by special inter-ontology roles. Like DDL, *ε-connections* treat local domains as disjointed and do not support subsumption relations between modules. *ε-connections* supports Pellet OWL reasoner and SWOOP ontology editor, but reasoning is performed by a single Pellet reasoner and not distributed. A similar approach is that of Schlicht and Stuckenschmidt (2008), using imports and ε-connections. But these works need to modify the original OWL language.

KAONp2p (Haase and Wang, 2007) is an infrastructure for query answering over distributed ontologies based on the KAON2 system, rather than global reasoning. The approach is focused on ontology management and knowledge selection for reasoning about an individual (ABox). The relevant parts of the terminologies are copied to the peer that answers the query. This makes more intensive use of the network communications.

Another approach is that of Ensan and Du (2010). The authors have created a framework to develop and work with ontologies in a modular manner. Or as Hirankitti and Xuan (2011) do, create a meta-language.

Some of them try to focus on scalability (Hogan *et al*, 2011) and their feasibility (Jiménez-Ruiz *et al*, 2012). However, all of the systems described above have drawbacks. They need to change the original OWL language, adding some *import-like* operations, rather than using the original OWL import semantics using logical import from the OWL specification. They have some restrictions on the use of definitions from remote ontologies in local definitions or on the way knowledge is distributed a priori.

Schlicht and Stuckenschmidt (2010) propose a method to overcome these problems, covering the problem of big ontologies in terms of description of the ontology itself, but not the problem of distributing the size of the individuals of the ontology. They are able to distribute using TBox distribution, but they are unable to perform ABox distribution, and have a common TBox.

These pieces of research focus on distributed reasoning on different (and disjointed) ontologies and their main idea is to overcome the heterogeneity of ontologies through ontology mapping. They provide a mechanism to do reasoning on different ontologies but have not paid much attention to the performance of reasoning, e.g., time efficiency, communication overhead and system scalability.

Fang *et al* (2008) try to achieve this goal using a simple yet effective idea. They (as we do) create a first TBox, reasoning locally, to make an ABox reasoning using further ABox information. ABox Reasoning is carried out using rule engines in each node. In this way they create a subset of OWL-DL (rather than the whole set) by means of rules, using the rule set of Minerva (Zhou *et al*, 2006). Besides, the reasoning process does not efficiently address the ontology update problem.

Generally speaking there is no general solution to our problem, but there are some attempts in application specific solutions (Strobbe *et al*, 2012). They have achieved, through different techniques, the problem of distributing the Ontology Reasoning, but with some big constraints hindering their use that we cannot afford in our system, such as modifying the expressivity or the language itself.

To sum up, little work has been done in our direction. Apart from some works done in distributed Semantic Web reasoning, most of the works described above follow different approaches. And

none of them is able to fulfill our requirements. Our work covers a gap in the world of Semantic Web technologies.

6. Conclusions and Future Work

We have shown OCP, our framework to develop Context-Aware applications. This framework includes several options to process streams of context information distributedly using the advantages of Semantic Web technologies in an efficient way. This processing can be built using aggregation of different sources (e.g. sensors) by successive processing executions, made distributedly by the clients.

To do so, we have proposed a novel architecture to distribute the Semantic Web Reasoning. This architecture is based on the listener/consumer interface. The context processing is done in the so-called *Context Information Processing Blocks (CIPBs)*. CIPBs subscribe just to the part of information they need to do their processing. The OCP Core sends a continuous stream of Semantic Information related to their subscription. This subscription can include information produced by other CIPBs. Therefore, a CIPB can be built on-top-of other CIPBs, so creating a complex processing system using an aggregation of simple CIPBs.

Making the system distributed has some advantages. The system becomes much more **dynamic and decentralized**. The users will be able to change the system behaviour. OCP will be more **efficient and scalable**. The distribution has been done in such a way that processing done by the clients can be executed very efficiently. In this way, the system is released from that burden, and has a **faster response time**. Furthermore, information is usually produced in the place where it is consumed. The system is also more **secure, having higher availability**, since it does not have to deal with high demanding operations from possible malicious users. The system is now **modular and more versatile**. We can extend it with new reasoning and processing methods by dynamically using the advantages of the Semantic Web technologies. Finally, it achieves **improved privacy and mobility**. Since users make reasoning locally, keeping their preferences in their devices, they will be visible only to them and available everywhere they are, with no need to send that information to every new system they visit.

But distributing the system also has some drawbacks. On the one hand, we have the possibility of having cycles from the information sent and received by the CIPBs. We have handled this by adding some metadata which also allows us to get a derivation trace from the reasoned information. On the other hand, we have a problem with temporalization. This implies that in some cases the clients will have to be aware of this distribution. Finally, for situations where information may be uncertain, incomplete or inconsistent, we propose an argumentation process.

For future work we are trying to distribute the Ontology Reasoning as well. Although we know it will still be desirable in many situations to keep distribution within the system, we will add it as an option for the scenarios where the processing capabilities of all the devices are very limited and we need to do it distributedly. We are also working on dealing with problems of rules defined by users who are not trusted. To do so, we are thinking of reputation and argumentation using the derivation traces. Finally, we are creating tools to help the user build high-level processing blocks such as pattern-recognition in an easy and friendly way.

References

ALLIANCE, O. (2008): OSGi service platform release 4 version 4.2-early draft.

APPELTAUER, M., HIRSCHFELD, R., HAUPT, M. and MASUHARA, H. (2011): Contextj: Context-oriented programming with java, *Information and Media Technologies* 6(2): 399–419.

BAADER, F., CALVANESE, D., McGUINNESS, D., NARDI, D. and PATEL-SCHNEIDER, P. (2007): *The Description Logic Handbook*, Cambridge University Press New York, NY, USA.

BARDRAM, J. (2005): The Java Context Awareness Framework (JCAF) – A service infrastructure and programming framework for context-aware applications, *Pervasive Computing*, 98-115.

BATSAKIS, S., STRAVOSKOUFOS, K. and PETRAKIS, E. (2011): Temporal reasoning for supporting temporal queries in owl 2.0, *Knowledge-Based and Intelligent Information and Engineering Systems*, 558–567.

BENCH-CAPON, T.J.M. and DUNNE, P.E. (2007): Argumentation in Artificial Intelligence, *Artificial Intelligence*, 171(10-15): 619–641.

BRICKLEY, D., GUHA, R. and McBRIDE, B. (2004): RDF vocabulary description language 1.0: RDF schema, *W3C recommendation*, 10: 27–08.

CARBOGIM, D., ROBERTSON, D. and LEE, J. (2000): Argument-based applications to knowledge engineering, *The Knowledge Engineering Review*, 15(2): 119–149.

CUENCA GRAU, B., PARSIA, B. and SIRIN, E. (2011): Combining OWL ontologies using e-connections, *Web Semantics: Science, Services and Agents on the World Wide Web*, 4(1).

DEL VESCOVO, C., PARSIA, B., SATTLER, U. and SCHNEIDER, T. (2010): The modular structure of an ontology: An empirical study, *Proc. of DL*, 10: 4–7.

DEY, A., ABOWD, G. and SALBER, D. (2001): A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications, *Human-Computer Interaction*, 16(2): 97–166.

DUQUE-RAMOS, A., FERNÁNDEZ-BREIS, J.T., STEVENS, R. and AUSSENAC-GILLES, N. (2011): Oquare: A square-based approach for evaluating the quality of ontologies, *Journal of Research and Practice in Information Technology*, 43(2): 159.

ENSAN, F. and DU, W. (2010): A knowledge encapsulation approach to ontology modularization, *Knowledge and Information Systems*, 1–35.

FANG, Q., ZHAO, Y., YANG, G. and ZHENG, W. (2008): Scalable distributed ontology reasoning using DHT-based partitioning, *The Semantic Web*, 91–105.

FENSEL, D., HORROCKS, I., VAN HARMELEN, F., McGUINNESS, D. and PATEL-SCHNEIDER, P. (2001): OIL: Ontology infrastructure to enable the Semantic Web, *IEEE Intelligent Systems*, 16(2): 38–45.

GARCÍA-SOLA, A., GARCÍA-VALVERDE, T. and BOTÍA, J. (2010a): On the efficiency of semantic web based context computing infrastructures, in *13th International Conference on Information Fusion*, IET, 274–283.

GARCÍA-SOLA, A., GARCÍA-VALVERDE, T. and BOTÍA, J. (2010b): Reasoning on a Semantic Web Based Context-Awareness Middleware, *Trends in Practical Applications of Agents and Multiagent Systems*, 147–155.

GOLDFARB, C. and PRESCOD, P. (2000): *XML Handbook*, Prentice Hall PTR, Upper Saddle River, NJ, USA.

HAASE, P. and WANG, Y. (2007): A decentralized infrastructure for query answering over distributed ontologies, in *Proceedings of the 2007 ACM Symposium on Applied Computing*, ACM, 1356.

HIRANKITTI, V. and XUAN, T. (2011): A meta-reasoning approach for reasoning with SWRL ontologies, in *Internation Multiconference of Engineers*.

HITZLER, P., KROTZSCH, M. and RUDOLPH, S. (2011): *Foundations of Semantic Web Technologies*, Chapman and Hall/CRC.

HOGAN, A., PAN, J., POLLERES, A. and REN, Y. (2011): Scalable OWL 2 reasoning for linked data, *Reasoning Web. Semantic Technologies for the Web of Data*, 250–325.

HOMOLA, M. and SERAFINI, L. (2010): Augmenting subsumption propagation in distributed description logics, *Applied Artificial Intelligence* 24(1–2): 39–76.

HONG, J., SUH, E. and KIM, S. (2009): Context-aware systems: A literature review and classification, *Expert Systems with Applications* 36(4): 8509–8522.

HORROCKS, I., PATEL-SCHNEIDER, P., BOLEY, H., TABET, S., GROSOF, B. and DEAN, M. (2004): SWRL: A semantic web rule language combining OWL and RuleML, *W3C Member submission* 21.

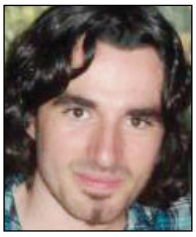
HU, B., WANG, Z. and DONG, Q. (2012): A modeling and reasoning approach using description logic for context-aware pervasive computing, *Emerging Research in Artificial Intelligence and Computational Intelligence*, 155–165.

JIMÉNEZ-RUIZ, E., GRAU, B. and HORROCKS, I. (2012): On the feasibility of using OWL 2 DL reasoners for ontology matching problems, in *OWL Reasoner Evaluation (ORE) Workshop*.

MA, Y., KALASHNIKOV, D. and MEHROTRA, S. (2008): Toward managing uncertain spatial information for situational awareness applications, *IEEE Transactions on Knowledge and Data Engineering,* 1408–1423.
McGUINNESS, D.L and VAN HARMELEN, F. (2004): OWL web ontology language overview, *W3C recommendation* 10, 2004–03.
MUÑOZ, A. and BOTÍA, J.A. (2008): ASBO: Argumentation System Based on Ontologies, in KLUSCH, M., PECHOUCEK, M. and POLLERES, A. eds, Cooperative Information Agents XII, Vol. 5180 of *Lecture Notes in Artificial Intelligence*, Springer Verlag, Prague, Chezc Republic, 191–205.
MUÑOZ, A. and BOTÍA, J.A. (2009): A Formal Model of Persuasion Dialogs for Interactions among Argumentative Software Agents, *Journal of Physical Agents,* 3(3): 3–10.
OBERLE, D. (2009): How ontologies benefit enterprise applications.
RAHWAN, I. and SIMARI, G. eds (2009): *Argumentation in Artificial Intelligence*, Springer Series, NY, USA.
SCHILIT, B., ADAMS, N. and WANT, R. (1994): Contextaware computing applications, in *First Workshop on Mobile Computing Systems and Applications*, WMCSA., IEEE, 85–90.
SCHLICHT, A. and STUCKENSCHMIDT, H. (2008): Distributed resolution for alc, in *Proceedings of the International Workshop on Description Logics*.
SCHLICHT, A. and STUCKENSCHMIDT, H. (2010): Peer-to-peer reasoning for interlinked ontologies, *International Journal of Semantic Computing*, Special Issue on Web Scale Reasoning.
SCHMIDTKE, H. (2012): Contextual reasoning in context-aware systems, in *Intelligent Environments: Workshop Proceedings*, IOS Press, Incorporated, 82.
SERAFINI, L. and HOMOLA, M. (2010): Modular Knowledge Representation and Reasoning in the Semantic Web, *Semantic Web Information Management: A Model-Based Perspective,* 147.
SERAFINI, L. and TAMILIN, A. (2005): Drago: Distributed reasoning architecture for the semantic web, *The Semantic Web: Research and Applications,* 361–376.
STRANG T. and LINNHO-POPIEN, C. (2004): A context modeling survey, in *Workshop on Advanced Context Modelling, Reasoning and Management UbiComp*, Citeseer.
STROBBE, M., VAN LAERE, O., DHOEDT, B., DE TURCK, F. and DEMEESTER, P. (2012): Hybrid reasoning technique for improving context-aware applications, *Knowledge and Information Systems* 31(3): 581-616.
STUCKENSCHMIDT, H., PARENT, C. and SPACCAPIETRA, S. (2009): *Modular Ontologies: Concepts, Theories and Techniques for Knowledge Modularization*, 5445, Springer.
VAN HILLE, P., JACQUES, J., TAILLARD, J., ROSIER, A., DELERUE, D., BURGUN, A. and DAMERON, O. (2012): Comparing drools and ontology reasoning approaches for telecardiology decision support.
ZAMORA-IZQUIERDO, M.A., SANTA, J. and GÓMEZ-SKARMETA, A.F. (2010): An integral and networked home automation solution for indoor ambient intelligence. *Pervasive Computing*, IEEE, 9(4): 66–77.
ZHANG, D., CAO, J. and ZHOU, J. (2009): Extended Dempster-Shafer Theory in Context Reasoning for Ubiquitous Computing Environments, in *Proceedings of the 7th IEEE International Conference on Embedded and Ubiquitous Computing*.
ZHOU, J., MA, L., LIU, Q., ZHANG, L., YU, Y. and PAN, Y. (2006): Minerva: A scalable OWL ontology storage and inference system, *The Semantic Web-ASWC 2006*, 429–443.

Biographical Notes

Alberto Garcia-Sola *received the BSc degree in computer science from the School of Computer Science, the University of Murcia, Murcia, Spain, in 2007, and the MSc degree in computer science from the Department of Computer Science, at the same university, in 2008. He has been working toward the PhD degree in ambient intelligent systems at the University of Murcia. His research interests include ambient intelligence, computational intelligence, fuzzy logic, indoor localisation, pervasive computing, and intelligent buildings. In 2014 he obtained a permanent position as IT Manager in the Government of Spain.*



Alberto
Garcia-Sola

Teresa Garcia-Valverde

Teresa Garcia-Valverde *is a senior research officer in the Department of Information and Communications Engineering (DIIC) at the University of Murcia, Spain, where she had previously obtained her BSc and MSc degrees (computer science) in 2005 and 2007. She obtained her PhD in 2012 at the same university. She has worked on several projects in both academia and industry and she is a member of the IEEE Computational Intelligence Society (CIS). She develops her research activity in Murcia University. Her main interests include ambient intelligence and intelligent data analysis. Teresa has co-authored several publications in workshops, conferences, journals and books.*

Andrés Muñoz

Andrés Muñoz *is a senior lecturer in the Technical School at the Catholic University of Murcia (UCAM), Spain. He had previously obtained his BSc and MSc degrees (computer science) in 2003 and 2005 respectively at the University of Murcia. He obtained his PhD in 2011 at the same university. He has worked on several research projects in artificial intelligence and education. He develops his research activity both in UCAM and the University of Murcia. His main interests include argumentation in intelligent systems, Semantic Web technologies and ambient intelligence.*

Juan A. Botia

Juan A. Botia *obtained his PhD degree in computer science and artificial intelligence in 2003. He was associate professor at the Faculty of Computer Science, the University of Murcia, Spain from 2009 to 2014. While there he taught machine learning related subjects mainly, to undergraduate and postgraduate students. He also developed a line of research centred on the design and application of machine learning techniques to different data analysis problems within the context of smart and context-aware systems. He led research projects as principal investigator with both public and private funding and wrote more than 40 publications in JCR indexed journals. In July 2014 he joined the Department of Medical and Molecular Genetics at King's College London (KCL), as a Research Fellow under the supervision of Dr Mike Weale to design new algorithms for co-expression network analysis within brain tissue. He is currently focusing on characterising the key molecular pathways within human substantia nigra and putamen in control human brain samples with the aim of identifying the mechanisms that lead to Parkinson's disease.*