Journal of Universal Computer Science, vol. 16, no. 9 (2010), 1234-1266
submitted: 11/11/09, accepted: 4/4/10, appeared: 1/5/10 © J.UCS

Knowledge Authoring with ORE: Testing, Debugging and Validating Knowledge Rules in a Semantic Web Framework

Andrés Muñoz Ortega
(Dpto. de Ingeniería de la Información y las Comunicaciones
Computer Science Faculty, University of Murcia
30100 Murcia, Spain
amunoz@um.es)

Jose M. Alcaraz Calero
(Automated Infrastructure Lab
Hewlett Packard Laboratories
BS34 8QZ Bristol, UK
jmalcaraz@hp.com)

Juan A. Botía Blaya
(Dpto. de Ingeniería de la Información y las Comunicaciones
Computer Science Faculty, University of Murcia
30100 Murcia, Spain
juanbot@um.es)

Gregorio Martínez Pérez
(Dpto. de Ingeniería de la Información y las Comunicaciones
Computer Science Faculty, University of Murcia
30100 Murcia, Spain
gregorio@um.es)

Félix J. García Clemente
(Dpto. de Ingeniería y Tecnología de Computadores
Computer Science Faculty, University of Murcia
30100 Murcia, Spain
fgarcia@um.es)

Abstract: Ontology rule editing, testing, debugging and validation are still handcrafted and painful tasks. Nowadays, there is a lack of tools that take these tasks into consideration in order to ease the work of the developer. This paper is devoted to explain how we have come to a new tool, ORE (Ontology Rule Editor), which significantly eases these tasks. It rests on a Semantic Web framework together with reasoning engines, which operate with semantic representations. Its design maintains a loosely coupling from the framework and from rule engines. Collaborative functionalities have been tackled in order to enable a real integration of the rule authoring across different tools and/or users. A practical validation of the approach by instantiating our tool with

Jena and Pellet reasoning engines is presented here. In order to demonstrate its use, the tool is applied to the task of rule-based management in a ubiquitous computing scenario.

Key Words: Knowledge authoring, Semantic Web, Ontology rule editor, Reasoning engines, Conflict management.

Category: D.2.2, I.2.1, I.2.4, M.1

1 Introduction

Knowledge authoring has become a fundamental process in the current knowledge society, since it allows organizations and entities to obtain and manage valuable information when taking decisions. This process usually consists of three stages [Suraweera et al. 2004], involving domain experts and knowledge system administrators. In the first stage we find elicitation and acquisition of knowledge over a particular domain. The result of this stage is the creation of a model representing such a domain, known as *domain model.* In the second stage different reasoning tasks are applied to this model in order to infer new knowledge from the initial acquired data. Finally, in the third stage the expert validates the domain model and the inferred knowledge. This paper is particularly focused on the management of *knowledge rules* employed in this authoring cycle, i.e. how to edit, test, debug and validate rules which are directed to entail new knowledge from an initial domain model.

The knowledge authoring processes studied in this paper are based on the use of Semantic Web technologies [Berners-Lee et al. 2001]. The adoption of the Semantic Web overcomes the search and integration limitations of knowledge management systems [Joo and Lee 2009, Hefke 2004]. More specifically, we refer to the adoption of ontologies based on Description Logic (DL) [Baader et al. 2003] as the representation of the domain model in such authoring processes. One consequence of this adoption is the possibility of defining knowledge rules by using elements from these ontologies. These rules are then employed to infer new knowledge when their conditions are fulfilled, as explained later. Normally, the management of knowledge rules amounts to a considerable work load for domain experts and knowledge system administrators. A solution to this problem resides in computerizing and automatizing the tasks involved in this process (i.e., the edition, execution and validation of such rules). Thus, a software application could guide the administrator or expert in the performance of knowledge rules authoring tasks with the aim of saving time and reducing complexity. The next paragraphs give a brief introduction about how knowledge rules can be defined by using DL ontologies and how Semantic Web technologies can support the implementation of the proposed solution.

Knowledge models based on DL ontologies are usually divided into TBox (*terminological*) and ABox (*assertional*) components (see figure 1)[1]. The TBox contains the vocabulary and schema that define domain concepts, their properties, and the relationships (called *roles* in DL) among them. Hence, a concept represents a set of elements with similar characteristics (e.g., people, devices, parts of a building such as room, floors, etc.), whereas roles symbolize binary relationships among elements (e.g., a device *is located* in a specific room; a room *is subsumed* in a particular floor). Apart from these two atomic components, the DL language offers some axioms and restrictions over them to represent more complex domains (e.g., a *computer* is a *specialization* of the *device* concept; a room must have *at least* one door).

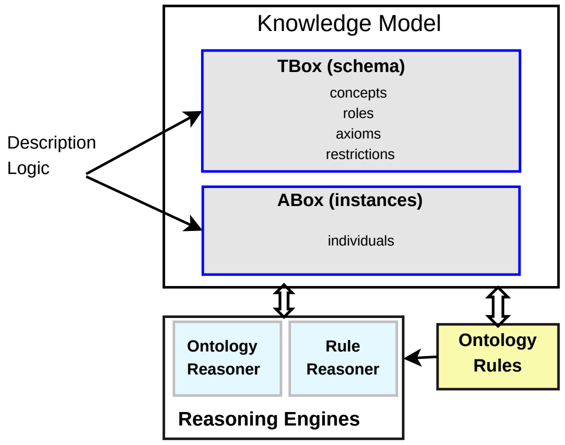


Figure 1: Knowledge models can be represented by means of DL ontologies with ontology rules. Reasoning engines perform several types of inferences on the knowledge models.

On the other hand, the ABox is populated with instances of these concepts and roles, representing a specific situation in the domain according to that schema. These instances are called *individuals* and they are defined by means of two types of assertional statements. Hence, a concept assertion $C(a)$ states that an individual a belongs to the concept C, while a role assertion $R(a, b)$ states that an individual a is related to an individual b by the role R.

As mentioned above, the knowledge represented by means of Semantic Web ontologies can be used to produce knowledge rules, which in turn derive new facts in the domain through a deductive inference process. Knowledge rules (also called here *ontology rules*, as they are formed by elements from ontologies) are of the

[1] Figure 1 has been extended from the one found in[Baader et al. 2003], page 46

form of an implication between a conjunction of antecedents and a conjunction of consequents. Both conjunctions consist of the aforementioned assertions $C(x)$ and $R(x, y)$, where x, y are either individuals in the ABox, variables or data values (e.g., integer, boolean, etc.). Predefined or user-defined functions can also be part of these rules (e.g., mathematical operations, user code, etc.). Knowledge rules can be thought of as "*if-then*" statements: All elements in the antecedent must be true in order to the elements in the consequent become true. Therefore, these rules are a natural and straightforward alternative to entail new knowledge based on the conditional existence of information. Let us see a simple example of an ontology rule.

Suppose a system which controls automatically a set of devices within a building. The domain model in this system is represented by an ontology where *Device* and *Room* are concepts, and *is_located* is a role that relates devices to their placement in a specific room. Moreover, *Device* has the property *power* which indicates whether the device must automatically be switched on or off. Suppose also that the ABox of the ontology contains the assertions $\mathit{Device}(ACME\_Com)$, $\mathit{Room}(\mathit{Lab\_6})$ and $\mathit{is\_located}(ACME\_Com, \mathit{Lab\_6})$. Now, suppose that a system administrator wants to express that all devices located in *Lab_6* must be switched off. This conditional statement can be defined through the knowledge rule *R_Lab* by using the previous ontology elements as follows:

$$R_{Lab}: \ \mathit{Device}(?x) \wedge \mathit{Room}(\mathit{Lab\_6}) \wedge \mathit{is\_located}(?x, \mathit{Lab\_6}) \Rightarrow \mathit{power}(?x, \text{“}\mathit{off}\text{”})$$

The variable $?x$ will take values from all existing individuals in the ABox that are known to be devices and to be located in *Lab_6* ($ACME\_Com$ in this case).

Apart from being employed as a language for representing knowledge models, DL ontologies empower computers to automatically perform different reasoning processes on such models. *Ontology reasoning* is one of the available processes for making inference (see figure 1, inside the reasoning engines box). Some operations of this type of reasoning allows extracting new knowledge that was implicit in the ontology, as for example *classification* (i.e., to compute the subsumption relationships between concepts so as to obtain a complete domain hierarchy) and *realization* (i.e., to find all concepts from the hierarchy to which an individual belongs). Another interesting operation of ontology reasoning is the *consistency checking* of an ontology, which validates the domain model and the specific situation represented in it. In other words, this operation checks that the schema defined by concepts and roles is consistent and the assertions stated in the ontology do not violate any axiom or restriction in such a schema. On the other hand, *rule-based reasoning* introduces an inference process which generates new knowledge by employing ontology rules and a rule reasoner (see bottom right part of figure 1). Note that all these reasoning processes are particularly useful when dealing with the tasks of executing and validating knowledge rules during the knowledge authoring cycle.

While there exist several tools for editing ontologies and an increasing number of reasoners that allow for the reasoning processes listed above, we have detected a lack of tools to manage the knowledge rules authoring cycle in a supervised manner. The main aim of this kind of tools should be directed to edit, test, debug and validate knowledge rules in an easy and intuitive manner. Specially interesting tools would be those that abstract the rule edition task from the underlying logic, i.e. the only ability requested from the user to define rules is an acceptable level of familiarity with the domain and how it is represented, but no specific DL expertise is needed. A clear explanation of the results produced when testing rules should be given, so as to detect and correct undesired effects derived from the execution of such rules. Moreover, a practical characteristic of this type of tools would be the simple integration of whichever existing reasoning engine into it for testing rules, together with the possibility of adding new reasoners. Eventually, inconsistency management between rules should be another desirable feature. This problem often arises as a result of contradictions among different well-founded rules. For example, a rule could derive the fact of switching off a particular device because of maintaining operations, while another rule may claim to the contrary, i.e. to switch the same device on due to it having to perform a critical task. In this case, detecting the conflict and offering the opportunity of disabling one of these rules could be deemed as an immediate solution.

The main contribution of this paper is the development of a generic rule authoring system in a Semantic Web framework which includes the characteristics listed in the above paragraph. To this end, we have made an appreciable improvement and extension of a tool, the Ontology Rule Editor (ORE) [Muñoz et al. 2006], whose first version consisted in a GUI that guided the user through three simple steps when editing rules. Here two augmented versions of the tool equipped with the aforementioned characteristics are presented: ORE-API and ORE-GUI. Both proposals enable a comfortable rule editing, testing, debugging and validation. ORE-API is a set of methods to be called from any application code to edit ontology rules, select different combinations of reasoning engines and infer knowledge as a result of applying ontology and rule reasoning. Optionally, the application may ask ORE-API for the derivation trace of the inferred knowledge. On the other hand, ORE-GUI consists in a graphical front-end which guides the user in the management of the functionalities offered in the ORE-API. Therefore, the ORE-API is intended to be used from any application that needs to work with ontology rules and reasoning processes, whereas ORE-GUI is a stand-alone application aimed to graphically edit, test, debug and validate ontology rules in any domain.

The ORE project is currently being utilized in European projects such as

Ecospace[2] and in national ones such as I3Media[3], among others. In this work we use a ubiquitous computing scenario where there is a high necessity of modeling and monitoring knowledge rules to illustrate the ORE-API and ORE-GUI functionalities. Particularly, this scenario is based on the management of information directed to control devices in an intelligent building.

The rest of the paper is structured as follows. The next section introduces the underlying elements on which ORE is based, and then gives an abstract architecture of the rule authoring system. Section 3 provides a description of the ORE architecture implementation, focusing on technical aspects of the authoring process. A scenario in which ORE is used to generate knowledge rules to control an intelligent building is exposed in section 4. Section 5 discusses several tools for managing ontologies and rules, comparing them with ORE. Finally, section 6 summarizes our contribution and points out the future work.

2 Software Architecture for a Generic Rule Authoring System in a Semantic Web Framework

2.1 Motivation. An intelligent building scenario.

As previously introduced, this work is aimed to build a generic rule authoring system such that it may be applied to a number of different domains integrated into a Semantic Web framework. This system, called ORE, is presented as an API and GUI version. In order to illustrate how ORE manages the authoring cycle of knowledge rules, it has been integrated into a ubiquitous computing scenario developed in our research group. The scenario corresponds to the management of an intelligent building. This building is equipped with a pervasive system [Weiser 1991, Hansmann et al. 2003] that offers several intelligent services for workers and visitors, as for example to adapt the operation mode of some devices in the building to the worker's or visitor's context and preferences. Normally, pervasive systems consist of software agents that share a common description model about any domain. However, no individual agent has a complete vision on the entire domain. Instead, they have a partial –and possibly overlapped– point of view about the current state of affairs, because such agents hold different sets of facts and rules about the same domain. As a result, these systems need to manage the edition and monitoring of rules, presenting an excellent field to test the authoring tool developed here.

In particular, the scenario focuses on the management of air-conditioner (AC) devices allocated throughout the building. The functionality of these devices is offered as services by different agents. There are two types of agents: *personal*

[2] http://www.ami-communities.eu/wiki/ECOSPACE
[3] https://i3media.barcelonamedia.org/

agents, which manage the available services to a specific user, and *floor controllers*, which are responsible for the general operations of all the devices in each floor of the building. Suppose that Bob is a worker who has assigned an office with an AC device. He usually selects a particular set of configurations for the services in his office, including the air-conditioning one, through the personal agent installed in his laptop or PDA. This agent has previously been programmed to represent the possible configurations of such services by means of knowledge rules, which are defined with ORE by a system administrator (such rules are hidden to Bob, the personal agent in his PDA only shows him a graphic description of each configuration). For example, one of the configurations for the AC device in Bob's office states that *when Bob is in whichever part of the building, the AC device in his room must automatically be switched on.* This configuration will be represented as a knowledge rule, denoted by **BobRule** henceforth (see section 4.1). According to this, the use of rules to infer new property values over elements in the domain is a desirable feature of these systems. Thus, the generation of new knowledge through rules should be permitted in a simple manner. It is one of the characteristics that can be found in ORE, as shown in section 4.

Being more ambitious, we propose the use of these rules for enabling ORE users to define complex behaviors, which specify how the pervasive system will act when some situations of interest happen. For instance, when a person is detected in different parts of the building, some privileges (to open doors, to make use of some devices, etc.) may be granted or denied to her depending on her profile and/or the current context. Back to the particular scenario, the floor controllers are programmed to control AC devices with the aim of saving energy. This configuration is specified by the following rule: *for every person whose office has installed an AC device, if he/she is located in a floor A, the office is in floor B, and* $A \neq B$*, then the AC device must be switched off.* It will be referred as **PowerSaveRule** henceforth.

Now suppose that there are three different types of RFID presence sensors in the building, which offer location information depending on the accuracy level needed for the agents. Thus, the sensors can detect the presence of a person in the building, in a particular floor or in a specific room, respectively, due to an RFID tag attached to the hand-held device. The information provided by the sensors is accessible for all agents in the pervasive system. Suppose that at a specific moment Bob is in the building, but he is located in a different floor from his office. Besides, he has selected the configuration which corresponds to *BobRule* for his AC device. Therefore, it is possible to reach a situation in which Bob's agent finds reasons to turn the AC device on (Bob is currently in the building), while the floor controller is requested to turn it off (Bob is in a different floor from his office). Now, the pervasive system should be able to detect and manage the conflict generated due to these two rules. As shown in section 4.2, ORE could be used to assist the system and administrators in these tasks. The complete scenario will be developed in detail in section 4.

To operate with rules such as the previous ones in a Semantic Web framework is not a trivial task. If the programmer or administrator is not familiar with rule programming and its syntax, it becomes hardly feasible. Moreover, it is essential to supervise the cycle of testing, debugging and validating rules, giving a clear interpretation of the facts inferred and conflicts detected during the reasoning process. The system described in this paper tries to manage these rules in an efficient and intuitive manner, either from any application code using ORE-API or graphically by means of ORE-GUI, supposing that the users of this tool are familiarized with the domain and its representation by means of ontologies.

The underlying elements on which ORE is based are described in the rest of the section using the intelligent building scenario as example. Firstly, the languages used here to represent knowledge and rules. Secondly, the abstract architecture of the generic rule authoring system is explained in detail, together with the reasoning engines which execute the ontology and rule-based reasoning.

2.2 Ontology and rule languages

Two types of languages are necessary to define rules in a domain modeled within the Semantic Web framework: (1) *Ontology languages*, which model the knowledge contained in such rules; and (2) *rule languages*, which model their syntax and semantics. Let us start with ontology languages.

RDF (Resource Description Framework) [Klyne and Carroll 2004] is a basic ontology language to model data and state assertions (i.e., facts that are true in the domain) in the form of triples. These are represented as (s, p, o), meaning that *the subject s is related to object o by the predicate p*. Usually, the subject denotes the individual on which the assertion is made, the predicate denotes a property or relationship of that subject, and the object is the value of that property.

However, RDF does not allow for a high level of expressiveness (e.g, disjointness of concepts, cardinality restrictions of properties, special characteristics of properties such as transitivity, etc.). Consequently, the kind of domains that can be modeled with this language is limited [Horrocks et al. 2003]. As a result of the efforts aimed to solve this problem, OWL (Web Ontology Language) [Dean et al. 2004] has been designed as a new standard ontology language based on RDF, but offering more expressiveness than this latter language provides. OWL has three increasingly-expressive languages: OWL Lite, OWL-DL, and OWL Full. The particular OWL version we have centered on in this paper is OWL-DL[4], due to the features offered by the restricted subset of Description

[4] http://www.w3.org/TR/owl-guide/

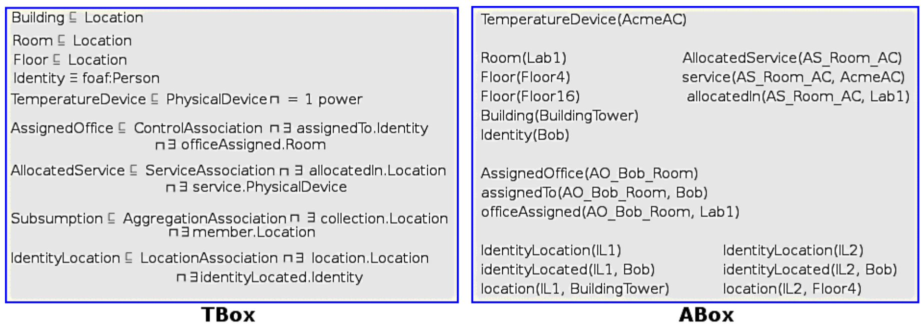
Logic used in this language (DL in OWL-DL stands for Description Logic). Let us now see some of these OWL-DL features.

The restrictions imposed to OWL-DL place it between Propositional and First-Order Logics. These restrictions are necessary to ensure that the operations included in the ontology reasoning process (classification, realization, consistency checking, etc.) are decidable, i.e. it is guaranteed that all inference operations included in this type of reasoning are computable in a finite time. In spite of such restrictions, OWL-DL still fulfills the modeling of domains which require a high level of expressiveness. Hence, the benefits of this ontology language are twofold: it gives a formal and expressive representation of a domain as a set of concepts and relationships among them, and moreover it enables an efficient and computable reasoning support.

As described in the introduction, OWL-DL ontologies are usually divided into the sets of statements known as TBox and ABox (see figure 1) to represent the schema and specific situations of a domain, respectively. To illustrate this, the intelligent building scenario has been represented with an ontology of this type (see figure 2). Elements such as building, floor, room, device, etc. are represented as concepts in the TBox. Hierarchical structures are constituted here through the subclass axiom $\sqsubseteq$ (e.g., the concept $TemperatureDevice$ is a subclass of $PhysicalDevice$). Moreover, the cardinality restriction on the property *power* indicates that it can only have one value for the same temperature device, i.e. "on" or "off". Therefore, a semantic inconsistency occurs when this property takes more than one different value at the same time. As shown in section 4, ORE provides mechanisms to discover and handle this kind of inconsistencies appropriately. Observe that OWL ontologies can reuse concepts from other OWL ontologies. Hence, $Identity$ is equivalent to the concept $Person$ defined in the FOAF ontology[5].

The following relationships between the domain concepts are captured: an identity has assigned one or more rooms through the $AssignedOffice$ relation; a room has allocated a device into it by means of $AllocatedService$; a part of the building is subsumed by another according to the $Subsumption$ relation; and any person in the building is located in a specific zone by $IdentityLocation$. Notice that these relationships are not modeled here as simple properties (e.g., *power* for temperature devices), but as concepts. The fundamental for this decision resides in that the types of such relationships could be used to classify them in different categories. Thus, $AssignedOffice$ is a special type of *control association* that an identity has over a room, whereas $Subsumption$ is an *aggregation association* between locations, or $AllocatedService$ is a *service association* defined in a location. Eventually, observe that the domain and the range of these associations are described through roles. For example, the role $\exists assignedTo.Identity$

[5] http://xmlns.com/foaf/spec/



```
Building ⊑ Location
Room ⊑ Location
Floor ⊑ Location
Identity ≡ foaf:Person
TemperatureDevice ⊑ PhysicalDevice ⊓ = 1 power
AssignedOffice ⊑ ControlAssociation ⊓ ∃ assignedTo.Identity
                    ⊓ ∃ officeAssigned.Room
AllocatedService ⊑ ServiceAssociation ⊓ ∃ allocatedIn.Location
                    ⊓ ∃ service.PhysicalDevice
Subsumption ⊑ AggregationAssociation ⊓ ∃ collection.Location
                    ⊓ ∃ member.Location
IdentityLocation ⊑ LocationAssociation ⊓ ∃ location.Location
                    ⊓ ∃ identityLocated.Identity
```

TBox

```
TemperatureDevice(AcmeAC)

Room(Lab1)                    AllocatedService(AS_Room_AC)
Floor(Floor4)                 service(AS_Room_AC, AcmeAC)
Floor(Floor16)                allocatedIn(AS_Room_AC, Lab1)
Building(BuildingTower)
Identity(Bob)

AssignedOffice(AO_Bob_Room)
assignedTo(AO_Bob_Room, Bob)
officeAssigned(AO_Bob_Room, Lab1)

IdentityLocation(IL1)             IdentityLocation(IL2)
identityLocated(IL1, Bob)         identityLocated(IL2, Bob)
location(IL1, BuildingTower)      location(IL2, Floor4)
```

ABox

Figure 2: The intelligent building scenario expressed in an OWL-DL ontology. The TBox contains the vocabulary and schema of the domain. A possible current state of affairs of such a domain is partially given in the ABox.

in $AssignedOffice$ means that the subject of this relationship is any instance of the $Identity$ concept, while $\exists officeAssigned.Room$ indicates that the object of the relationship takes its value as instances of $Room$.

On the other hand, the ABox in figure 2 reflects the current state of affairs in the scenario by means of concept ($C(x)$) and role ($R(x, y)$) assertions. In this case, Bob is an identity that has assigned the room $Lab1$ through the association AO_Bob_Room. Likewise, $Lab1$ has allocated the temperature device $AcmeAC$ according to AS_Room_AC. Note that Bob was detected in two different levels of location by RFID sensors, giving as a result the individuals $IL1$ and $IL2$, which place him inside the building and in the floor 4, respectively. Finally, the assertions that specify the subsumption relations among the parts of the building (i.e., $BuildingTower$ containing $Floor4$ and $Floor16$, and $Floor4$ containing $Lab1$) have been omitted for simplicity. A complete description of the scenario will be given in section 4.

Now let us see how the knowledge modeled by OWL-DL ontologies can be included in conditional rules. These rules are of the form of:

$$\textbf{IF } A_1 \textbf{ and } A_2 \textbf{ and } \ldots \textbf{ and } A_n \textbf{ THEN } B_1 \textbf{ and } B_2 \textbf{ and } \ldots \textbf{ and } B_m,$$

i.e. an implication between a set of antecedents (*body*) and a set of consequents (*head*). Each atom A_i/B_j in the body/head is an assertion of a concept $C(x)$ or role $R(x, y)$ from the DL language, where x, y are variables, individuals from the ABox or data values. Furthermore, a special type of atoms, called *built-ins*, can be included in the body and head of the rules. These built-ins are functions that offer different operations on variables, individuals or data values (e.g., comparison operations among these elements: equality, difference, etc.).

The semantic of these rules states that whenever all the atoms specified in the body are true in a domain, the atoms in the head must also be true in such a domain.

As stated at the beginning of this section, it is needed a rule language to model the syntax and semantic of this kind of rules. One proposal that maintains maximum compatibility with OWL is the Semantic Web Rule Language (SWRL)[Horrocks et al. 2004]. This rule language adds a new kind of axiom to OWL, namely Horn clause rules, which extends OWL's syntax and semantics to model rules as the one given above. Analogously to the restrictions imposed to OWL-DL in order to keep the bounds of decidability of such a language, SWRL is also restricted so as to maintain the decidability of the combination OWL-DL + SWRL. The expressiveness of the resulting language is still acceptable to define rules such as the ones employed in this scenario. Details about the restrictions on SWRL rules, so-called SWRL-DL safe rules, are shown elsewhere [Motik et al. 2005]. In particular, the rules used in this paper are SWRL-DL safe.

The floor controller's **PowerSaveRule** (see section 2.1) serves here as an example of SWRL rule. It is written below in a SWRL abstract form, where variables are preceded by the '?' mark. Lines 1 and 2 determine the floor $?fx$ where an identity $?i$ is currently located through the location association $?il$. Line 3 finds the room $?r$ assigned to that identity $?i$. Line 4 retrieves the floor $?fy$ where the room $?r$ is subsumed. Note that the floor $?fy$ could be the same floor $?fx$ where $?i$ is located or not. This condition is checked in line 5 by means of the built-in *notEqual*. This comparison function returns true if the two floors are not the same. Line 6 finds the temperature device $?ac$ allocated in room $?r$, if any exists. Finally, the rule consequent in line 8 states that $?ac$ must be powered off.

$$\begin{aligned}
\textbf{PowerSaveRule} : Identity(?i) \wedge Floor(?fx) \wedge IdentityLocation(?il) \wedge && (1)\\
location(?il, ?fx) \wedge identityLocated(?il, ?i) \wedge && (2)\\
\mathit{AssignedOffice}(?ao) \wedge \mathit{assignedTo}(?ao, ?i) \wedge \mathit{officeAssigned}(?ao, ?r) \wedge \mathit{Room}(?r) \wedge && (3)\\
Floor(?fy) \wedge \mathit{Subsumption}(?ss) \wedge collection(?ss, ?fy) \wedge member(?ss, ?r) \wedge && (4)\\
notEqual(?fx, ?fy) \wedge && (5)\\
\mathit{AllocatedService}(?as) \wedge \mathit{allocatedIn}(?as, ?r) \wedge \mathit{service}(?as, ?ac) \wedge \mathit{TempDevice}(?ac) && (6)\\
\Rightarrow && (7)\\
power(?ac, \text{“}\mathit{off}\text{”}) && (8)
\end{aligned}$$

To define SWRL rules in a friendly manner is not a trivial task. It is necessary to hide its XML-based syntax, since writing the previous rule by hand may take around 220 lines of SWRL XML code, for example. With the aim of managing rules in a simple and uniform manner, the rule structure in ORE follows the

SWRL proposal, where each concept assertion $C(x)$ and role assertion $R(x, y)$ in SWRL rules is represented in ORE by means of its equivalent triple $(x, rdf : type, C)$ and (x, R, y), respectively. The special predicate $rdf : type$ indicates that the subject of that triple belongs to the concept in the object, as for example $(Bob, rdf : type, Identity) \equiv Identity(Bob)$. Observe that this transformation between assertions and triples only consists in a change of syntax, while the level of expressiveness and semantics of both types of statements are the same. Thus, ORE offers a guided process to define each rule antecedent and consequent in three steps according to the triple structure, as it will be explained in section 3.3. Moreover, ORE rules also support built-ins, and the tool offers a set of guided steps to use them.

2.3 Generic rule authoring system architecture

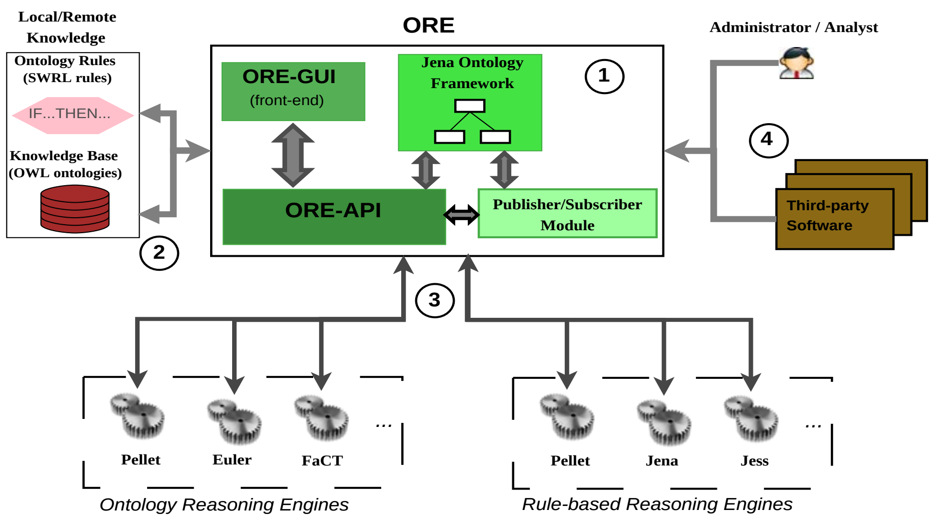


Figure 3: An abstract architecture view of ORE connected with its external resources.

Once the knowledge and rule models have been introduced, the next step in developing the generic rule authoring system resides in establishing the architecture to deal with both of them. Figure 3 depicts the abstract architecture of the rule authoring system ORE. The core system (1) is formed by the ORE-API, which encloses the methods for managing ontology rules (i.e., edition, inference, validation, etc.); ORE-GUI, a graphical interface of these tasks; the Jena framework [Carrol et al. 2004], which manages the ontologies representing the knowledge model; and a publisher/subscriber module that enables ORE to obtain and distribute knowledge from/to a remote server. Details about the specific design of ORE-API and ORE-GUI are given in section 3.2 and 3.3, respectively. The publisher/subscriber module is also explained in section 3.2. As for the Jena framework, it provides a set of methods for loading and managing OWL ontologies, together with a group of reasoning engines with different capabilities. Notice that Jena is employed in the core of the system to obtain a working model from the domain ontology, not as a reasoning engine. Hence, the working model can be accessed by ORE according to the rule authoring tasks to be performed. For example, ORE builds a domain concept hierarchy in a tree form, to be displayed when editing rules in the GUI, through the ontology working model given by Jena.

ORE receives OWL ontologies and SWRL rules as input (2). The tool obtains this knowledge set either locally through files or remotely by means of the publisher/subscriber module. As stated in the above paragraph, OWL ontologies are managed thanks to the Jena framework integrated in the core system, whereas the ORE-API is in charge of SWRL rules, which can also be graphically edited, tested, debugged and validated in the ORE-GUI.

Inference processes in ORE are accomplished by the combination of different reasoning engines (3). These engines can be distinguished according to the two types of inference that have been explained in the introduction (see figure 1, the reasoning engines box), namely ontology and rule-based reasoning. Regarding ontology reasoning engines, there exist several proposals such as Pellet [Sirin et al. 2007], Jena , Euler [Roo 2007], Fact++ [Tsarkov and Horrocks 2006] and Hoolet. Likewise, there are several implementations available in the field of rule-based engines, such as SweetRules, JEOPS, JLisa, Prova, OpenRules, Jess, RDFExpert, Pellet and Jena. Notice that some reasoners could be applied to perform both ontology and rule-based reasoning. Furthermore, different combinations of ontology and rule-based reasoners could be convenient in order to complement the advantages and drawbacks of each one, obtaining a trade-off between reasoning capabilities and computing resources. This feature is intended for an advanced usage of ORE, enabling several reasoner combinations according to the users' necessities and/or capabilities. In this context, one of the main advantages of ORE is its ease of integrating any reasoning engine in the API. Details on such combinations are given in section 3.1.

Finally, third-party software and administrators or analysts represent the ORE clients (4). Software applications can make use of the ORE-API to work with ontologies, rules and inference processes directly. Some examples of this type of applications are the software agents involved in the intelligent building scenario. Besides, ORE-GUI is a helpful tool for users that look for a visual presentation when editing, testing, debugging and validating ontology rules. By following the intuitive steps of the *wizard* integrated in the front-end, the tedious task of generating rules is significantly reduced.

3 Practical Implementation of the Architecture

The architecture previously presented in figure 3 has been implemented as framework for authoring of ontology rules. Thus, ORE [6] framework has been published to the community as a free and open source project. The following subsections present a description of all ORE framework components. Subsection 3.1 provides details in the integration of the reasoning engines employed in the tool. Likewise, details of ORE-API and ORE-GUI are exposed in subsections 3.2 and 3.3, respectively.

3.1 Details on implementation of the generic rule authoring system

Testing and debugging tasks in the rule authoring process demand that the developers get the full control in the execution of the rules, step by step, during the whole authoring session. Moreover, developers need to track the reasons for which rules have been fired in order to discover errors in the design of the rules. At the same time, both ontology and rule-based reasoning processes have to be taken into account in ORE to test and validate the edited rules. Notice that OWL and SWRL requires different reasoning processes and both should be covered in order to get a full reasoning platform.

ORE architecture has been designed in order to enable the combination of whichever combinations of ontology and rule reasoners being able to manage OWL and SWRL language, respectively. In fact, all the different engines cited in section 2.3 could be added to ORE. There are several criteria for selecting those engines to be included in ORE. Regarding ontology reasoners, for example, Pellet provides a high level of expressiveness, namely SHOIQ [Horrocks and Sattler 2007], which in turn is a key feature for the ORE framework in order to offer a complete ontology reasoning process. For this reason, Pellet is included by default in ORE. Moreover, due to the easy integration between the underlying Jena ontology model and the reasoners provided in the Jena framework, the Jena reasoning engines have also been included in ORE framework. Almost all the other engines exposed in section 2.3 do not have a direct support for OWL and they have not been included in ORE yet.

Analogously, the ORE framework enables the incorporation of rule-based reasoners such as those exposed in section 2.3. However, there are some key features to decide which of those rule reasoners will be included in ORE based

[6] ORE is available at http://sourceforge.net/projects/ore, version 3.0

on their usefulness in the testing and debugging tasks. On the one hand, the selected rule reasoners should provide the knowledge inferred by the rules isolated from the rest of the knowledge inferred by other reasoning processes. This feature is required during the debugging processes in order to get the control of all the knowledge produced by the rules solely. For example, the Jena rule reasoner does this distinction whereas the Pellet rule reasoner does not, and for this reason Pellet is mixing the ontology and rule reasoning results together. On the other hand, the rule reasoner should provide derivation traces, as it is an essential feature for rule testing, debugging and validation. They constitute the explanation proofs which demonstrate why inferred facts should be hold. Such explanations are the foundations in knowledge authoring for rule behaviors analysis. The rule-based reasoners in Jena and Pellet provides derivation traces. Both of them have been added to the ORE framework presented here, whereas the rest of rule-based engines enumerated in section 2.3 could be also added to ORE in the future. Jena is a suitable rule engine for the ORE framework since it has the ability to split the inferred knowledge from the base knowledge, being this feature essential for debugging purposes.

The selected engine combination of the current ORE version for testing and debugging purposes consists of Pellet as the ontology reasoner and Jena as the rule reasoner. This combination is supposed to be a one-time decision, only to be changed when an improved engine is released or more functionality is needed (e.g. to include a fuzzy reasoner or to deal with uncertainty). These new releases are directly related to the current state of the art in reasoning engines. Hence, if the user decides to insert a new engine combination, the ORE framework offers an easy method to achieve it. Figure 4 depicts the class diagram of the ORE reasoning architecture. A *RuleReasoner* is in charge of providing a debugger component in ORE-API. A *Debugger* is a composition of one or more reasoner implementations. Each *Reasoner* implementation inherits from an abstract class *Reasoner* which provides basic features related to rule and ontology managements.

Essentially, in case a user decides to extend the rule reasoner support, he has to create a new class implementing the interface *Reasoner*. This interface only has one abstract method for carrying out the inference process. Then, the user can retrieve all the needed information from *OntologyManager* and *RuleManager* utility classes using the methods provided by the *Reasoner* classes. This information is then combined with the API of the new reasoner and the results obtained from the reasoning process are published using the analogous methods.

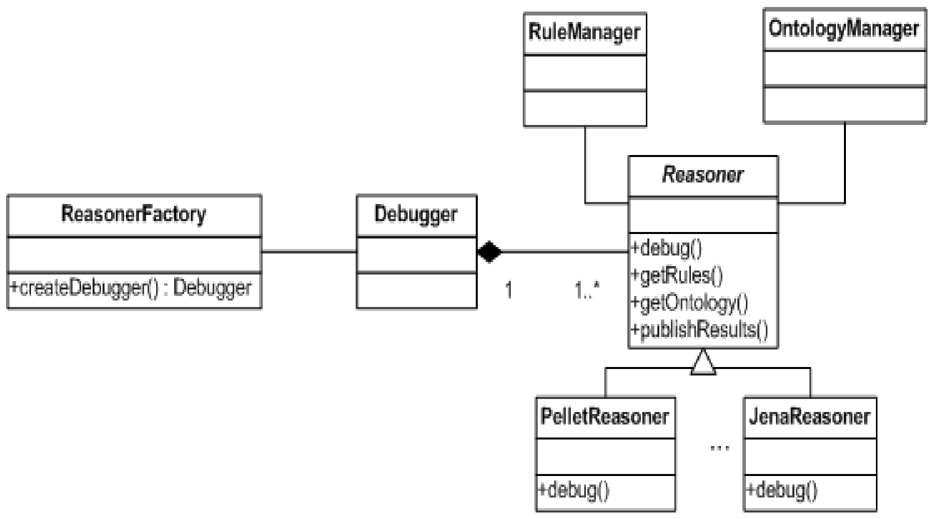


Figure 4: Class diagram of the ORE reasoning architecture.

3.2 ORE-API. Authoring tool for software applications

ORE-API is a helpful tool for third-party software applications which need knowledge rule authoring services. These services are directly implemented in the ORE-API as a set of methods to create ontology rules, perform ontology inference processes, test and debug ontology rules, retrieve the inferred knowledge generated by ontology and rule reasoning processes, obtain the explanations for such inferred facts, and activate/deactivate rules during the authoring process

ORE-API also offers methods for managing OWL ontologies on which the rules are defined and executed. The ontology administration is accomplished through Jena services, including the ontology load and import mechanisms. The knowledge base managed in ORE can be loaded either locally or remotely. The former uses OWL and SWRL local files to this end, whereas in the latter retrieves the domain model and rules by means of a publication/subscription mechanism based on Web Services. Hence, ORE is able to get a knowledge base from a server and update it with new rules or information so as to other ORE users can use this augmented knowledge base later. To this end, ORE-API communicates with the publisher/subscriber connector integrated in ORE. This connector is implemented by adopting the WS-Eventing technology [Box et al. 2006] and it has been designed to be useful in collaborative scenarios where users belonging to the same administrative domain want to share knowledge in a friendly fashion. The sequence diagram of this remote service is shown in figure 5.

Following the figure 5, *ORE User A* publishes the knowledge base (ontolo-

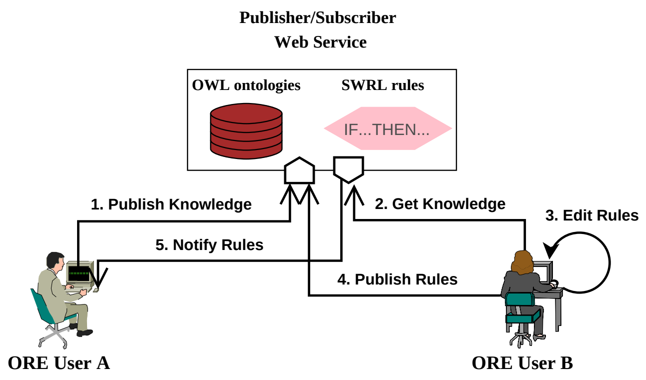


Figure 5: Two ORE users sharing knowledge and rules through a publisher/subscriber knowledge system.

gies and rules) on which he is currently working and he subscribes to the Web Service in order to be notified when some change happens. Then, *ORE User B* downloads this knowledge in order to edit some rules that she is interested in. After the rules have been edited, she updates the knowledge base by publishing them in the publication/subscription system. *ORE User A* who has been previously subscribed in the system is notified about this change according to the WS-Eventing notification mechanism. In this manner, he can supervise the new knowledge or rules that have been added to his model. This feature enables the knowledge interchange and synchronization among all the parties that are using the ORE-API in a collaborative fashion.

As previously stated, ORE-API offers reasoning processes with different reasoner combinations. The integration of the reasoning processes in ORE-API has been depicted in figure 6. It starts dividing the initial knowledge model into rules on one hand (SWRL), and ontology model (OWL) on the other hand. Next, the ontology model is loaded into the Pellet ontology reasoner producing a semantic enrichment of this ontology model as output. The reason for doing this separation between SWRL and OWL is for isolating the execution of the rules in order to get the full control over their execution for debugging and validating purposes. Then, ontology rules have to be transformed from the SWRL syntax managed in the ORE architecture to the specific rule format imposed by the rule reasoner in case it is necessary. Next, these rules and the semantically enriched ontology model are inserted in the rule reasoner. Finally, this reasoner infers new facts that could be grouped as rule-based knowledge as shown in figure 6. Regarding testing and debugging capabilities, ORE allows enabling or disabling rules in order to isolate groups of rules and control their behavior in a debugging session. This selective capability provides a rapid method for detecting rules which do not work as expected. Furthermore, after discovering an unexpected behavior in a specific ontology rule, ORE-API offers the possibility of disabling an atom (or a set of atoms) appearing in the antecedent or consequent to exactly identify the possible errors in the rules during the debugging process.

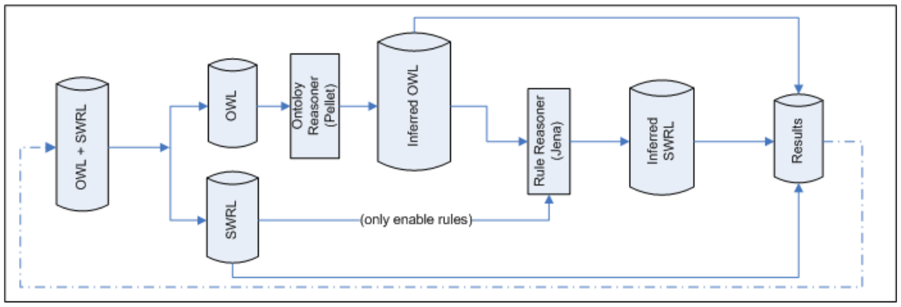


Figure 6: Ontology processing for knowledge authoring.

The rule reasoner infers knowledge aligned twofold. This alignment is defined as the ability to provide more information over an inferred fact. Firstly, each fact is associated with the rule that produces it through the derivation trace provided by the rule engine. Secondly, each fact is linked to the list of grounded antecedents that fired that rule. Thanks to both associations, the debugging process in ORE is able to draw the justifications of any inferred fact. These associations are highly valuable to detect mistakes when designing and testing rules. Consequently, since derivation traces are available for each inferred fact, the identification of unexpected rule firings can be detected in ORE.

After the debugging tasks have been completed, it is necessary to validate the new inferred facts with respect to the domain model. This operation is also supported by ORE-API in order to ensure *consistency* in the knowledge base. Consistency is referred as the non-existence of contradictory facts in a model. Usually, inconsistencies appears as violations of the axioms or restrictions defined in the domain. They are difficult to detect and usually arises because of unexpected assertions of consequents, ill-designed or conflictive rules. For example, the property *power* in temperature devices (see section 2.2) can only take one value for the same device; therefore a semantic inconsistency occurs when this property takes more than one different value, e.g. “on” and “off”. Thus, ORE-API is able to discover inconsistencies, and to notify it to the application software which is the responsible for handling it appropriately.

According to the architecture depicted in figure 6, the knowledge inferred by the rule reasoner has been isolated on each execution of the rule reasoner. Each of those executions could be considered as an step in the debugging process. As a result, after this execution, the third-party application can obtain the variable values, the derivation traces and the explanations associated to the inferred knowledge. Then, this application can decide to include the inferred knowledge again in the initial knowledge base in order to start a new step in the authoring process or just simply discard the inferred results because a rule has a mistake, for example. Notice that after discarding the inferred knowledge, the same testing step could be done again but now changing the activation/deactivation of the rules and their atoms in order to find the previous mistakes.

3.3 ORE-GUI. The rule editing, testing, debugging and validation graphical tool

ORE-GUI is a visual tool intended for being used by system administrators. All the functionalities involved in the knowledge rule authoring process that have been implemented by ORE-API are now offered here in a graphical mode. ORE-GUI provides a full control in the navigation across the knowledge models managed in ORE-API. As previously seen in section 3.2, the knowledge model is accessed via two different manners. Both local and remote knowledge models are managed in ORE-GUI indistinctly, with the latter requiring an enrollment in the publication/subscription system and enabling a collaborative environment among ORE users. ORE-GUI notifies graphically the updates received by others ORE users/applications in real-time. In this context, ORE users can cooperate in order to create the ontology rules in a friendly environment.

The rule authoring process is performed hiding the details of the underlying SWRL syntax to the user. She edits rules through a *wizard* that guides the creation or modification tasks. These tasks are executed in an easy and intuitive "drag and drop" manner, where ontology elements (concepts, roles, individuals, etc.) are dragged to the correspondent part of the rule structure displayed in the GUI. This structure is based on RDF triples, as previously exposed in section 2.2. To this end, the rule editor in ORE-GUI offers a complete vision on the domain ontology, where the user can navigate across all the information represented in it. Both antecedents and consequents of a rule are defined in the same manner: the wizard guides the user through three steps, dragging and dropping the subject, predicate and object from the domain model to the correspondent triple element of the antecedent/consequent being edited in each step. For example, according to the intelligent building scenario introduced in section 2, to state that Bob is a person, the following RDF triple is defined:

$$(Bob, rdf\_type, Identity),$$

To edit this triple in ORE-GUI, the user should first drag the individual *Bob* from the domain model to the subject part. Then, she selects the predicate *is a* (*is a* is referred here as the *rdf_type* property in a friendly manner) in the second step. Finally, she drags the *Identity* concept to the object part. The edition task is validated by the wizard, avoiding the appearance of syntactical errors during the rule definition. A full definition of a rule through the wizard instructions is shown in section 4.

Regarding debugging capabilities provided by ORE-GUI, the tool offers some options such as the activation/deactivation of rules and rule atoms in a selective fashion by means of simple clicks. Therefore, they can be rapidly debugged to establish dependence relationships among them, and to identify whether the rule has the expected reaction or not. Another important option is to determine why the rule has been fired. In this sense, ORE-GUI gives a textual explanation that links the facts causing the fulfillment of the antecedents to the fired rule. After a rule execution in a debugging session, the user decides whether she wants to insert the inferred facts into the initial knowledge base in order to perform the next debugging step or discard these facts totally or partially. As a result, ORE-GUI can be seen as an ontology rule editing, testing, debugging and validation tool.

ORE-GUI supports all the reasoning combinations integrated in the ORE-API. The inference process depicted in figure 6 and performed by the selected reasoning combination is done just by a click action. Not only will this action provide new inferred knowledge and a consistence validation, but it also offers the different knowledge bases showing all the information involved in each step of the knowledge rule authoring process.

Finally, all the conflictive facts detected after the reasoning task are grouped, allowing for a manual conflict solving mechanism. This option is offered as a graphical form in ORE-GUI, by indicating first the conflictive situation to the user, and then asking her to determine which conflictive rule or fact should be deactivated or removed. This last feature confers an extra debugging power to ORE over other tools of knowledge authoring as Prot*é*g*é* [Noy et al. 2001] or Ontotrack [Liebig et al. 2004], which do not implement it. The benefits of all ORE-API and ORE-GUI features will be illustrated in section 4 by means of the intelligent building scenario described in section 2.1.

4 Using the Rule Authoring Tool in the Intelligent Building Scenario

ORE has been successfully integrated and tested in a ubiquitous computing scenario. It consists on the management of the pervasive services offered by an intelligent building and the combination of users' preferences with those services (see section 2.1). Particularly, this scenario deals with the management of AC devices. On one hand, a worker called Bob wants that the AC device placed in his room remains switched on during his stay in the building. This configuration has been edited by means of a system administrator who transforms it into a rule by using ORE-GUI. Contrarily, the policy of the building administration on AC devices states that they must be switched off if the owner of the room allocating the AC device is located in a different floor of the building where the room is. This rule is added to the ubicomp system by a system administrator thanks to ORE-API (see the rule *PowerSaveRule* in section 2.2).

This scenario is used here to illustrate the entire knowledge authoring process described in the introduction of the paper, and it specially focused on the management of rules. Regarding the first stage, dedicated to the acquisition of knowledge, the system administrator has modeled the intelligent building domain by means of an OWL ontology based on the DMTF-CIM standard, called OWL-CIM [García et al. 2008]. This OWL ontology represents the concepts as partially shown in the TBox of figure 2.A complete vision of the scenario is depicted in figure 7. The creation of the OWL-CIM ontology that represents this specific domain has been performed thanks to Prot*égé* [Noy et al. 2001], a broadly used OWL editor. The specific scenario involving Bob and the AC device management (partially represented in the ABox of figure 2) has been modeled following the same idea. This scenario allows establishing the real environment where a set of rules is going to be defined to control the AC devices. As shown next, the authoring cycle of these rules could efficiently be managed by ORE.

The OWL files of this scenario exposed in figure 7 has been distributed within the ORE framework, as a case of study. It describes the ACME business building tower, *BuildingTower*, which is composed of more than twenty floors. For simplicity, just the two floors involved in the scenario, *Floor*4 and *Floor*16, has been depicted. The composition relationship has been modeled using the *Subsumption* association (see section 2.2). Each floor contains in turn some rooms (also through the *Subsumption* association). *Bob* is an employee of this building whose assigned room is *Lab*1 (by means of the *AssignedOffice* association). This room is located in *Floor*4. Each floor is equipped with a *FloorController* that manages the devices present therein. The controller is linked to these services by the *ServiceAvailableToElement* association. For simplicity, only the *Floor*4 controller has been depicted in figure 7. Notice that the services available to the *FloorController* are the aggregation of the own floor services together with all the services obtainable from the rooms contained in this floor. Thus, the controller can access to the *ACMEAirConditioner* service in Bob's room.

On the other hand, the Bob's personal agent, *BobAgent*, shows him the accessible services in *Lab*1 in a graphic and friendly manner. This agent is linked to that room using the *ServiceAvailableToElement* association, in the same manner that the floor controller does. The available services in the room (e.g.

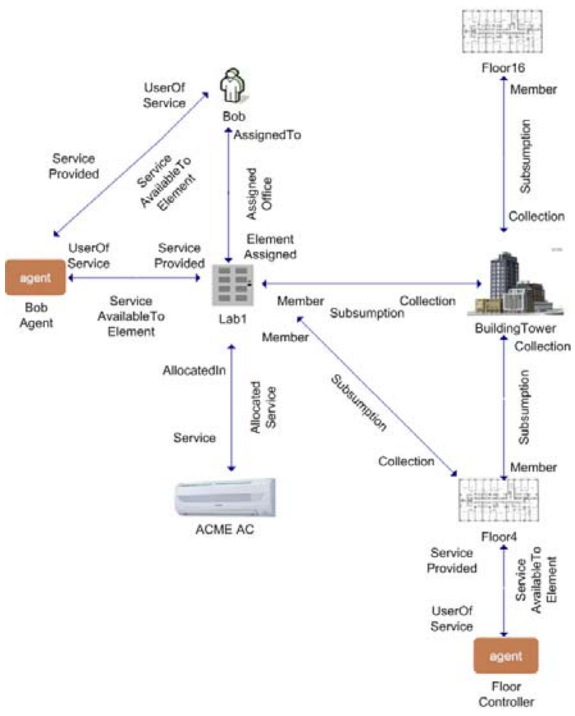


Figure 7: The running scenario of the intelligent building domain.

air conditioning) are modeled by *AllocatedService* associations. Hence, Bob's personal agent can access to the *ACMEAirConditioner* service through the composition of these two associations.

There are three different types of RFID sensors according to the covered area (not shown in figure 7). *Building sensors* are located in the external building doors. They detect a person entering the building, and then they send this event to the pervasive system. Then, the system creates an *IdentityLocation* association between the person who enters the building and the *BuildingTower*, and inserts it in the domain model. *Floor sensors* are located near the lifts and ladders, thus detecting people in the floor, whereas *Room sensors*, placed in room doors, provides the presence of a person in these rooms. There are not floor sensors in the main hall because there are not services therein. Thus, the floor sensors are detecting users when they are getting out of the lift on a given floor. These events are sent and managed in the pervasive system in the same way as in the building sensor case, creating the corresponding location association. Actually, the pervasive system should use a system such as OCP (Open Context Platform) [Nieto et al. 2006] for updating the knowledge base with the location information from the RFID sensors. OCP is a middleware system whose main goal is to manage the system's context information. It collects information from sensors and transforms it into context knowledge, by annotating semantically the information provided by sensors. However, this issue is beyond the scope of this paper. Instead, the sensor information has been simulated in the ORE framework by means of the following ontology rules:

BT: $\Rightarrow$ $\mathit{IdentityLocation}(?il) \land \mathit{identityLocated}(?il, Bob) \land \mathit{location}(?il, BuildTower)$
F4: $\Rightarrow$ $\mathit{IdentityLocation}(?il) \land \mathit{identityLocated}(?il, Bob) \land \mathit{location}(?il, Floor4)$
F16: $\Rightarrow$ $\mathit{IdentityLocation}(?il) \land \mathit{identityLocated}(?il, Bob) \land \mathit{location}(?il, Floor16)$
Lab1: $\Rightarrow$ $\mathit{IdentityLocation}(?il) \land \mathit{identityLocated}(?il, Bob) \land \mathit{location}(?il, Lab1)$

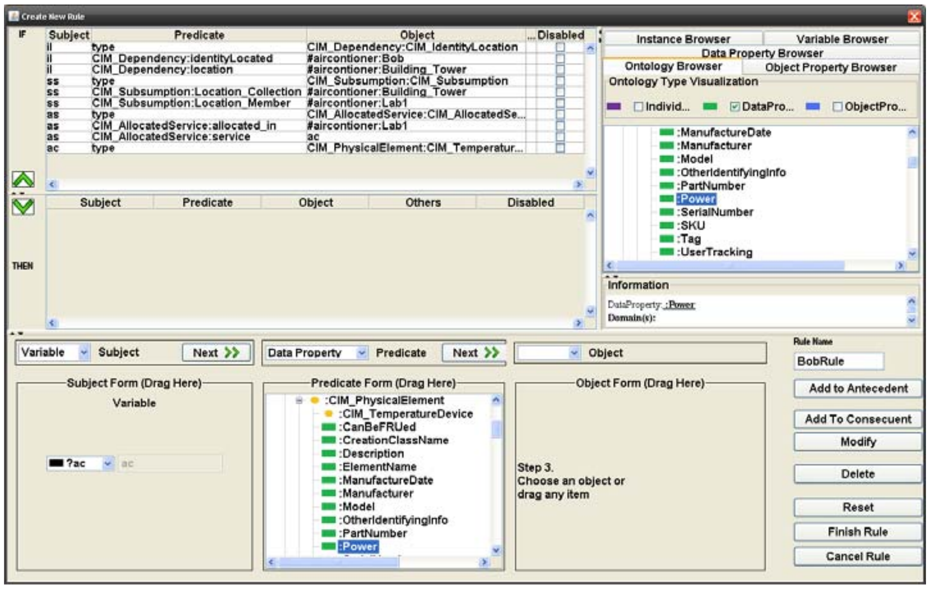
Note that these rules have an empty antecedent. It is equivalent to *true* when the rule is evaluated, therefore the consequent must always be true (assuming that the rule is activated). Indeed, these rules assert a new location association $?il$ between Bob and the corresponding part of the building depending on the sensor. Hence, if the rule BT is activated, it simulates that Bob has been detected by the building sensor and then the location association between Bob and $BuildingTower$ is asserted. Analogously, the rules $F4$, $F16$ and $Lab1$ simulate the detection of Bob in $Floor4$, $Floor16$ and $Lab1$, respectively. In this manner, the sensors functionality is emulated on the pervasive system so as to properly develop the scenario.

System administrators can use ORE-GUI to subscribe to the publication/subscription system (PSS) in order to obtain the intelligent building domain. This knowledge base is provided by the pervasive system which is subscribed to the PSS as well. Moreover, the pervasive system updates the current domain state according to a context middleware such as OCP. In our case, the context middleware has been simulated by the four rules previously described.

4.1 Rule authoring process

In order to show an example of the rule authoring process, suppose that a system administrator is responsible of editing the different users' preferences by means of rules. To this end, the system administrator uses a kind of templates which gathers such users' preferences to convert this information into rules. For this specific scenario, the system administrator models Bob's preferences with respect to the AC device placed in his room (see section 2.1). As a result, the system administrator defines a rule to switch on the AC placed in Lab1, which is Bob' office, when Bob is detected in the building. As seen in section 2.1, this rule is called $BobRule$ and it is given below in SWRL abstract syntax. $BobRule$ can be read as follows: if Bob is located in the $BuildingTower$, and this building contains the Bob's assigned room $Lab1$, and the room has an air conditioner, then turns this device on.

$$\textbf{BobRule} : IdentityLocation(?il) \land identityLocated(?il, Bob) \quad (1)$$
$$\land location(?il, BuildingTower)\land \quad (2)$$
$$Subsumption(?ss) \land collection(?ss, BuildingTower) \land member(?ss, Lab1)\land \quad (3)$$
$$AllocatedService(?as) \land allocatedIn(?as, Lab1) \land service(?as, ?ac) \quad (4)$$
$$\land TemperatureDevice(?ac) \quad (5)$$
$$\Rightarrow \quad (6)$$
$$power(?ac, \text{“}on\text{”}) \quad (7)$$


Create New Rule

Subject	Predicate	Object	...	Disabled
il	type	CIM_Dependency:CIM_IdentityLocation		☐
il	CIM_Dependency:identityLocated	#aircontioner:Bob		☐
il	CIM_Dependency:location	#aircontioner:Building_Tower		☐
ss	type	CIM_Subsumption:CIM_Subsumption		☐
ss	CIM_Subsumption:Location_Collection	#aircontioner:Building_Tower		☐
ss	CIM_Subsumption:Location_Member	#aircontioner:Lab1		☐
as	type	CIM_AllocatedService:CIM_AllocatedSe...		☐
as	CIM_AllocatedService:allocated_in	#aircontioner:Lab1		☐
as	CIM_AllocatedService:service	ac		☐
ac	type	CIM_PhysicalElement:CIM_Temperatur...		☐

Subject	Predicate	Object	Others	Disabled

Rule Name: BobRule

Step 3.
Choose an object or
drag any item

Figure 8: Bob's AC preference edited in ORE-GUI as a rule by a system administrator.

The edition of the Bob's preference through a rule in ORE-GUI can be seen in figure 8. First, the upper-right corner shows the concepts, properties, individuals and variables representing the current domain model. Essentially, these lists are the friendly, graphical and structured representation of the OWL ontology and they allow the user to utilize the ontology concepts without any knowledge about the OWL syntax. Then, the user can drag and drop concepts, properties and individuals from these lists to the items which compose the rule atoms, i.e. the triples of the form (subject, predicate, object). These three elements are shown in the bottom area of the figure 8. In case the user would like to create an atom belonging to the antecedent of the rule, he will be guided by a wizard to control and avoid any possible mistake in the atom definition. The wizard guides the user thought messages in the GUI in order to notify the next action in the authoring process. In an atom definition, the user will start dragging a concept from the list in the subject area and validate the action by clicking the button "next" button. This process is repeated in the same manner for the predicate and object of the atom. Then, in order to finish the process, the user has to press in "Add to Antecedent" or "Add to Consequent" according to their aim.

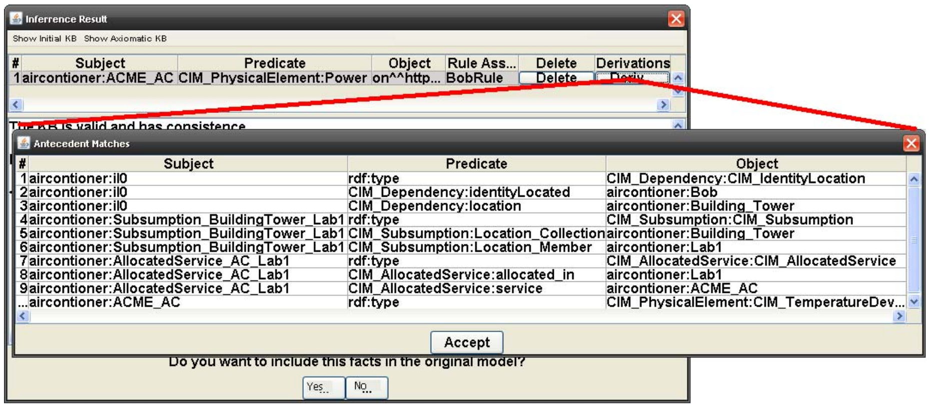
The snapshot in figure 8 represents a situation in which all the atoms of the *BobRule* antecedent have been already inserted in by the system administrator. He is now using ORE-GUI to edit the consequent atom $power(?ac, \text{"}on\text{"})$. The subject of the triple, $(?ac)$, has been achieved dropping this variable on the subject panel, while the predicate is established by dragging the property (*power*) from the ontology browser to its corresponding panel. Particularly, the figure represents the moment when the system administrator inserts the object part. As seen in the upper-left corner of the figure 8, ORE-GUI is always showing the current definition of the rule. Notice that the wizard guides the user through messages in the GUI (e.g., the appearance of the "Step 3. Choose an object or drag any item" in the bottom area of the figure 8). To finish with the rule edition, the system administrator just have to select the option "Data value" in the object area and insert *on*, and then clicking subsequently on "Add to Consequent" and "Finish Rule". It is worth mentioning the check boxes available next to each atom in the rule for enabling/disabling such atoms during a debugging session.

4.2 Rule testing, debugging and validation tasks

After defining *BobRule*, it is published in the PSS in a simple manner by just clicking on the button "Publish Rules" available in the tool. This rule is received by the pervasive system since it is registered in the publication/subscription system. Analogously, the system administration uses ORE-GUI to insert the energy policy of the building as a rule, i.e. *PowerSaveRule* (see section 2.2), and then he also publishes such a rule in the PSS. In this manner, the pervasive system could use these two rules to infer new knowledge. Furthermore, in this scenario the ORE-GUI of the system administrator is also configured to automatically execute an inference process whenever it receives information from other entities through the PSS, as for example from the pervasive system when it detects a person in the building, in a floor or in a room. Notice that this is an interesting feature to track and monitor all the changes in the system and to start the debugging processes of the rules when some unexpected situation appears.

In the initial situation of the scenario (when the pervasive system has not detected Bob in the building yet), the reasoning process does not produce any new knowledge as result. Previously to enter the building, suppose that Bob uses his PDA to activate the AC configuration that corresponds with his preferences. In this case, such a configuration matches with *BobRule*, and therefore the personal agent in Bob's PDA sends the activation order of this rule to the PSS, which is received by the pervasive system and the ORE-GUI of the system administrator (note that Bob is not aware of the existence or activation of this rule, he only uses the personal agent in his PDA, which shows him the possible AC configurations in a graphic mode, to activate/deactivate them). Moreover, suppose that *PowerSaveRule* is always activated for this scenario.

Next, the pervasive system detects Bob entering the building. To simulate this situation, the rule *BT* is activated. This new information is received by all the registered entities in the PSS (including the ORE-GUI of the system adminstrator). Thus, the AC device placed in Bob's room must be switched on according to *BobRule*. Specifically, the ORE-GUI of the system administrator automatically infers the new AC state. Part of the outcome of this inference can be seen in figure 9. The inference process provide all the inferred facts produced by the both ontology and rule reasoning using *BT* and *BobRule*. Figure 8 shows only the rule-based knowledge in a isolated way to enable debugging processes. From the debugging and testing perspective, this first inference process could be shown as the first step in a debugging session. Notice that the figure 9 displays the foundations that support the rule firing, i.e. the derivations traces. These traces can be used to determine whether the rule has been appropriately fired or its execution is due to an erroneous design. According to the inferred information,



#	Subject	Predicate	Object	Rule Ass...	Delete	Derivations
1	aircontioner:ACME_AC	CIM_PhysicalElement:Power	on^^http...	BobRule	Delete	Deriv...

The KB is valid and has consistence

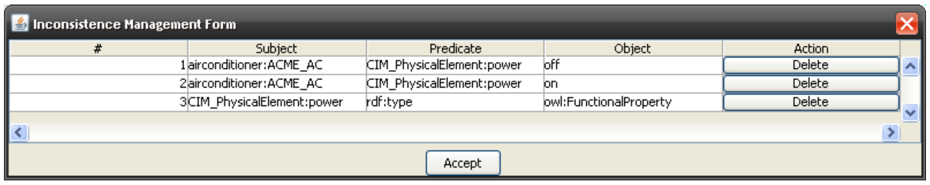
#	Subject	Predicate	Object
1	aircontioner:il0	rdf:type	CIM_Dependency:CIM_IdentityLocation
2	aircontioner:il0	CIM_Dependency:identityLocated	aircontioner:Bob
3	aircontioner:il0	CIM_Dependency:location	aircontioner:Building_Tower
4	aircontioner:Subsumption_BuildingTower_Lab1	rdf:type	CIM_Subsumption:CIM_Subsumption
5	aircontioner:Subsumption_BuildingTower_Lab1	CIM_Subsumption:Location_Collection	aircontioner:Building_Tower
6	aircontioner:Subsumption_BuildingTower_Lab1	CIM_Subsumption:Location_Member	aircontioner:Lab1
7	aircontioner:AllocatedService_AC_Lab1	rdf:type	CIM_AllocatedService:CIM_AllocatedService
8	aircontioner:AllocatedService_AC_Lab1	CIM_AllocatedService:allocated_in	aircontioner:Lab1
9	aircontioner:AllocatedService_AC_Lab1	CIM_AllocatedService:service	aircontioner:ACME_AC
...	aircontioner:ACME_AC	rdf:type	CIM_PhysicalElement:CIM_TemperatureDev...

Do you want to include this facts in the original model?

Figure 9: Inference results associated to the entrance of Bob into the building.

the administrator could perform some actions. On one hand, he can decide to insert the inferred information into the system passing to the next step in the debugging session. On the other hand, he can decide to discard totally or partially the inferred information and to repeat the inference process changing any rule definition or enabling/disabling totally or partially some rules in order to test and validate the correctness of the rules available in the system.

Thus, continuing with the example, the pervasive system now simulates that Bob is located by a floor sensor in the Floor 4 (Rule $F4$). In this case, this fact does not produce any new knowledge nor fire any other rule in the system administrator console. Then, Bob enters his room and he checks that the AC device is switched on. After some time, Bob decides to go to visit his friend Alice, whose office is in the sixteenth floor. He takes the lift, and after leaving it, the sensor detects Bob in that floor. Rule $F16$ simulates the new Bob's location in the pervasive system and in turn, rule $F4$ has been disabled to remove the previous Bob's location. In this case, after being notified by means of the publication/subscriber system, the ORE-GUI of the system administrator shows the debugging information related to $PowerSaveRule$ which has now been fired. This situation is reflected in figure 10.



#	Subject	Predicate	Object	Action
1	airconditioner:ACME_AC	CIM_PhysicalElement:power	off	Delete
2	airconditioner:ACME_AC	CIM_PhysicalElement:power	on	Delete
3	CIM_PhysicalElement:power	rdf:type	owl:FunctionalProperty	Delete

Figure 10: Inconsistency discovered after detecting Bob in Floor14.

In this case, the ORE framework has detected an inconsistency in the knowledge base due to the existence of a conflict between the inferred facts according to $BobRule$ and $PowerSaveRule$. Both rules force the AC device to be switched on and off at same time when Bob leaves the fourth floor. This situation violates the cardinality constraint over the $power$ property (functional property), which is restricted to have a unique value. ORE is able to detect this conflict and offers mechanisms to solve it manually. These mechanisms consist of the selective deletion of one of the conflicting facts. Moreover, thanks to ORE, the necessity of modifying the incompatible rules has been discovered. In this scenario, the conflict might be solved by changing the $location(?il, BuildingTower)$ antecedent in the $BobRule$ for the $location(?il, Floor4)$ condition. This change causes that the AC device will only be switched on when Bob is detected by the fourth floor sensor (and not when he is just entering the building). Such a change could be directly applied over the same debugging session by the administration of the system in an easy and rapid manner.

5 Related Work

As the popularity of the Semantic Web has rapidly increased, several ontology tools has been developed at the same time. Prot*égé* [Noy et al. 2001] is a famous ontology tool with an OWL plug-in that allows the user to define her own ontologies, and to export them into a variety of formats including RDF(S), OWL, and XML. It also supports the edition and execution of SWRL rules, but this process is based on a text editor with code completion capabilities oriented for advanced SWRL experts and do not offer any kind of wizard or visual GUI to help the authoring process. In this sense, Prot*égé* is observed as an effective and useful tool for the first stage of knowledge authoring. Ontologies obtained as a result of modeling domains with Prot*égé* are then a possible input for editing and validating rules in ORE during the second and third stages. Since both tools complement each other, they represent a compelling alternative to manage the entire cycle of the knowledge authoring process. Prot*égé* also contains visual ontology plug-ins, as Jambalaya, but they are focused on conceptual diagrams and do not offer any editing capabilities.

SWOOP [Parsia et al. 2005] is an IDE for developing ontologies, based on a Web browser interface. Hence, it allows browsing through hyper-links, which can be considered as an initial idea of showing ontologies in an intuitive way to the user. This tool has demonstrated to be useful for ontology debugging. However, it does not permit to debug or validate rules.

Some other rule editors have been found in the analysis made in this work, but they do not offer the ORE's debugging and validating characteristics. AEGONT [Murat et al. 2006] is an ontology development environment on .NET framework, whose major innovation lies in the Rule and Query Views, although they both are not fully functional yet. The Rule View is designed to ease writing of domain rules by means of a rule editor for people who are unfamiliar with OWL Rules Language, but it does not offer debugging capabilities and collaborative edition of rules. The Query View aims to present a user friendly interface for writing queries, and it offers statistics analysis as well as query results. On the other hand, The Graphical Rule Editor allows creating domain rules and translating them into F-Logic automatically. The rules are defined graphically by means of boxes, arrows and operators in such a flowchart style, however it does not provide debugging capabilities.

Finally, there are two rule editors that are nearer the ORE's philosophy and graphical design. The first is SWeDE [Pereira and Freire 2006] (Semantic

Web Development Environment), an extensible framework built on the Eclipse IDE including an OWL editor with features like syntax highlighting, auto-completion, and error-detection. It also integrates existing tools like the OWL Validator and DumpOnt (an ontology visualizer). The second one is RuleVISor [Matheus et al. 2005], an alpha-tested rule editor. Its main feature is the ability to deal with rule definition at a conceptual level that abstracts out the syntactic complexities of the SWRL representation. The user has the option of adding or deleting binary atoms, atomic atoms, instances, data value ranges and built-in functions simply by clicking on the appropriate icons. Both tools are extensible with respect to inference engines and they hide the specific SWRL syntax when editing rules, in a similar fashion with respect to ORE. Nevertheless, they lack a fine-grained debugging mechanism, an API to be used from software applications and the implementation of any kind of rule conflict detection and solution mechanisms such as ORE owns.

6 Conclusions and Future Work

This paper presents a new tool designed to manage the authoring process of rules in knowledge systems based on Semantic Web technologies. The knowledge model of these systems is normally given by means of ontologies. From these ontologies it is possible to define production rules (i.e., “if-then”) in order to describe preferences, behaviours, etc. in a natural and straightforward manner. These rules are mainly directed to model some aspects of the system according to the preferences stated in them.

However, controlling the entire authoring process of rules in Semantic Web frameworks (i.e., edition, test, debug and validation) is far from an easy task. Only the definition of these rules requires a deep specific knowledge about the rule programming language, not to mention the understanding of the inference mechanism. As a result, the aim of this work is to design and implement an advanced rule editor framework, called ORE (Ontology Rule Editor). This tool allows the users to edit, test, debug and validate such a kind of rules in an intuitive and guided manner in a collaborative environment.

In this paper, ORE is presented in the form of an API and of a GUI. The API is accessed by other software applications to manage rules and performing rule-based reasoning over the domain elements. The management of ontology elements, the retrieval of the inferred facts generated by ontology and rule-based reasoning, or the activation/deactivation of rules during the inference process are some of the ORE-API’s features. On the other hand, ORE-GUI is a stand-alone application for users such as system administrators which offers the same operations than in the API in a visual and friendly manner. In ORE-GUI the definition of rules is guided through several steps, by browsing the domain elements in the ontology through different views, and dragging and dropping them to their correspondent slots in the rule antecedents and consequents. In this manner, this tool hides the specific details about the syntax of the rule language that is being used.

Once new rules have been created, they can be tested by means of the facilities incorporated in ORE. The platform which performs the inference process over the knowledge model is based on Jena and Pellet reasoning engines, although it may effortless be extended with new reasoner capabilities (e.g. fuzzy inference). As for debugging and validation, both syntactic and semantic checking of rule definitions has also been included in the ORE framework. The former avoids ill-formed rules, by warning the user if the rule is being bad defined. The latter detects knowledge conflicts among rules, which usually are complicated to discover. These conflicts arises because of numerous causes: contradictory consequents, ontology axiom violation, etc. The conflict is then reported to the user, and besides a manual solving mechanism is provided.

The benefits of ORE have been illustrated by integrating the tool into a ubiquitous computing system. In this system we have developed a scenario where intelligent services are implemented by combining different kind of knowledge such as the current context, user's preferences and desired behaviors of the system. Such preferences and behaviours are expressed by means of rules in this scenario. The entire cycle of managing these rules, including the inference process and conflict detection, has been demonstrated in this scenario by means of the usage of ORE.

Future work includes enabling a visual and iconographical browsing through the ontology. We are working on a visual ontology which gives different capabilities to the domain elements that are shown in the ontology browser. These capabilities are related to how they can be visually presented (e.g. if a concept is an spatial container, one may double-click on it and see what is hidden inside). This will leave the door open to any user in order to create their own rules to express preferences, for example.

Another expected extension of the ORE framework is to adapt it to the new OWL release, OWL 2.0 [Motik et al. 2008], which includes more expressive constructions, such as negative assertions. OWL 2.0 management is already included in Pellet, which in turn is included in ORE. However, the ability to define these new constructions in ORE-GUI is not included yet. We are currently studying other combinations of axiomatic and rule engines in the ORE framework to improve the semantic expressiveness supported. For instance, Pronto [Klinov 2008] is a reasoning engine with probabilistic reasoning support that may be included in ORE framework.

Acknowledgments

This work has been partially supported by the Fundación Séneca within the Program "Generación del Conocimiento Científico de Excelencia" (04552/GERM/06) and by the Spanish Ministerio de Ciencia e Innovación (MICINN) within the FPU Program grants AP2006-4154 and AP2006-4150 and the research projects TIN2008-06441-C02-02 and TIN2009-14475-C04. It has been also partially supported by the European Comission FEDER funds under Grant CSD2006-00046.

References

[Baader et al. 2003] Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors. *"The Description Logic Handbook: Theory, Implementation, and Applications"*. Cambridge University Press, New York, NY, USA, 2003.

[Baader and Sattler 2000] Franz Baader and Ulrike Sattler. "Tableau Algorithms for Description Logics". In *International Conference on Automated Reasoning with Analytic Tableaux and Related Methods, TABLEAUX 2000*, pages 1–18. Springer-Verlag, July 2000.

[Berners-Lee et al. 2001] Tim Berners-Lee, James Hendler, and Ora Lassila. "The Semantic Web". *Scientific American*, 284(5):3443, 2001.

[Box et al. 2006] Don Box, Luis Felipe Cabrera, Craig Critchley, Francisco Curbera, Donald Ferguson, Steve Graham, David Hull, Gopal Kakivaya, Amelia Lewis, Brad Lovering, Peter Niblett, David Orchard, Shivajee Samdarshi, Jeffrey Schlimmer, Igor Sedukhin, John Shewchuk, Sanjiva Weerawarana, and David Wortendyke. "Web Services Eventing (WS-Eventing)". Technical report, W3C, 2006. http://www.w3.org/Submission/WS-Eventing/

[Bozsak et al. 2002] Erol Bozsak and et al. "KAON - Towards a Large Scale Semantic Web." In *EC-WEB '02: Proceedings of the Third International Conference on E-Commerce and Web Technologies*, pages 304–313, London, UK, 2002. Springer-Verlag.

[Carrol et al. 2004] Jeremy J. Carroll, Ian Dickinson, Chris Dollin, Dave Reynolds, Andy Seaborne, and Kevin Wilkinson. "Jena: implementing the Semantic Web recommendations." In *Proceedings of the 13th international World Wide Web conference*, pages 74–83. ACM Press, 2004.

[Dean et al. 2004] Mike Dean, Dan Connoll, Frank van Harmelen, James Hendler, Ian Horrocks, Deborah L. McGuinness, Peter F. Patel-Schneider, and Lynn Andrea Stein. Web Ontology Language (OWL). Technical report, W3C, 2004. http://www.w3.org/TR/owl-features/

[García et al. 2008] Felix J. García, Gregorio Martínez, Andrés Muñoz, Juan A. Botía, and Antonio F. Gómez Skarmeta. "Towards Semantic Web-based Management of Security Services". *Springer Annals of Telecommunications*, 63(3-4):183–194, 2008.

[Mime 2001] Mime Sweeper Research Group. RdfExpert: A Web-powered Expert System for Generic Inference Tasks. http://public.research.mimesweeper.com/RDF/RDFExpert/Documentation/HTML/Overview.html, August 2001.

[Hansmann et al. 2003] Uwe Hansmann, Lothar Merk, Martin S. Nicklous, and Thomas Stober. *"Pervasive Computing : The Mobile World"*. Springer, 2003.

[Hefke 2004] M. Hefke "A Framework for the Successful Introduction of KM Using CBR and Semantic Web Technologies". *Journal of Universal Computer Science*, 10(6):731–739, 2004.

[Horrocks et al. 2003] Ian Horrocks, Peter F. Patel-Schneider, and Frank van Harmelen. "From SHIQ and RDF to OWL: The making of a Web Ontology Language". *Journal of Web Semantics*, 1:7–26, 2003.

[Horrocks et al. 2004] I. Horrocks, P. F. Patel-Schneider, H. Boley, S. Tabet, B. Grosof, and M. Dean. SWRL: A Semantic Web Rule Language Combining OWL and RuleML. Technical report, W3C, 2004. http://www.w3.org/Submission/SWRL/

[Horrocks and Patel-Schneider 2004] Ian Horrocks and Peter F. Patel-Schneider. "A Proposal for an OWL Rules Language." In *WWW '04: Proceedings of the 13th international conference on World Wide Web*, pages 723–731, New York, NY, USA, 2004. ACM.

[Joo and Lee 2009] J. Joo, and S. M. Lee. "Adoption of the Semantic Web for Overcoming Technical Limitations of Knowledge Management Systems". *Expert Systems with Applications: An International Journal*, 36(3):7318–7327, 2009.

[Kifer et al. 1995] Michael Kifer, Georg Lausen, and James Wu. "Logical Foundations of Object-oriented and Frame-based Languages". *Journal ACM*, 42(4):741–843, 1995.

[Klinov 2008] Pavel Klinov. Pronto: a Non-monotonic Probabilistic Description Logic Reasoner. In *Proceeding at 5th European Semantic Web Conference (ESWC08)*, 2008.

[Klyne and Carroll 2004] Graham Klyne and Jeremy J. Carroll. Resource Description Framework (RDF): Concepts and Abstract Syntax. Technical report, W3C, 2004. *http://www.w3.org/TR/rdf-concepts/*

[Liebig et al. 2004] Thorsten Liebig, Holger Pfeifer, and Friedrich von Henke. "Reasoning Services for an OWL Authoring Tool: An Experience Report". In *Proceedings of the International Workshop on Description Logics*, 2004.

[Matheus et al. 2005] Christopher J. Matheus, and Kenneth Baclawski Mieczyslaw M. Kokar, and Jerzy A. Letkowski. An Application of Semantic Web Technologies to Situation Awareness. In *4th International Semantic Web Conference*, 2005.

[Motik et al. 2008] Boris Motik, Peter F. Patel-Schneider, and Ian Horrocks. OWL 2 Web Ontology Language: Structural Specification and Functional-style Syntax. Technical Report, W3C, April 2008. http://www.w3.org/TR/owl2-syntax/

[Motik et al. 2005] Boris Motik, Ulrike Sattler, and Rudi Studer. "Query Answering for OWL-DL with Rules". *Journal of Web Semantics: Science, Services and Agents on the World Wide Web*, 3(1):41–60, JUL 2005.

[Muñoz et al. 2006] Andrés Muñoz, Antonia Vera, Juan A. Botía, and Antonio F. Gómez Skarmeta. Defining Basic Behaviours in Ambient Intelligence Environments by means of Rule-based Programming with Visual Tools. In J. C. Augusto, editor, *1st Workshp of Artificial Intelligence Techniques for Ambient Intelligence. ECAI*, 2006.

[Murat et al. 2006] Tugba Ozacar Murat Osman Unalir, Ovunc Ozturk. Aegean Ontology Environment (AEGONT). Project Report MSR 2003176, Ege University, Bornova-Izmir, March 2006.

[Nieto et al. 2006] Ignacio Nieto, Juan A. Botía, and Antonio F. Gómez-Skarmeta. "Information and Hybrid Architecture Model of the OCP Contextual Information Management System." *Journal of Universal Computer Science*, 12(3):357–366, 2006.

[Noy et al. 2001] N. F. Noy, M. Sintek, S. Decker, M. Crubezy, R. W. Fergerson, and M. A. Musen. "Creating Semantic Web Contents with Protege-2000". *IEEE Intelligent Systems*, 16(2):60–71, 2001.

[O'Connor et al. 2005] Martin O'Connor, Holger Knublauch, Samson Tu, Benjamin N. Grosof, Mike Dean, William Grosso, , and Mark Musen. "Supporting rule system interoperability on the Semantic Web with SWRL". In *Proceeding of 4th International Semantic Web Conference ISWC*, Ireland, Nov 2005.

[Parsia et al. 2005] Bijan Parsia, Evren Sirin, and Aditya Kalyanpur. "Debugging OWL ontologies". In *Proceedings of the 14th international conference on World*

Wide Web, pages 633–640, 2005.

[Pereira and Freire 2006] R.G. Pereira and M.M. Freire. “SWEDE: A Semantic Web Editor Integrating Ontologies and Semantic Annotations with Resource Description Framework”. In *International Conference on Internet and Web Applications and Services/Advanced AICT-ICIW '06*, 2006.

[Roo 2007] Jos De Roo. Euler Proof Mechanism. http://www.agfa.com/w3c/euler/, 10 2007.

[Sirin et al. 2007] Evren Sirin, Bijan Parsia, Bernardo Cuenca Grau, Aditya Kalyanpur, and Yarden Katz. “Pellet: A Practical OWL-DL Reasoner”. *Journal of Web Semantics*, 5(2):51–53, 2007.

[Suraweera et al. 2004] Pramuditha Suraweera, Antonija Mitrovic, and Brent Martin. “The Role of Domain Ontology in Knowledge Acquisition for ITSs. In *7th International Conference on Intelligent Tutoring Systems*, pages 207–216, 2004.

[Tsarkov and Horrocks 2006] Dmitry Tsarkov and Ian Horrocks. *Automated Reasoning*, volume 4130 of *Springer Lecture Notes in Computer Science*, chapter FaCT++ Description Logic Reasoner: System Description, pages 292–297. Springer Berlin / Heidelberg, 2006.

[Horrocks and Sattler 2007] Ian Horrocks and Ulrike Sattler. A Tableaux Decision Procedure for SHOIQ In *Journal of Automated Reasoning* Springer Verlag, 39(3):245–429, 2007

[Weiser 1991] Mark Weiser. The Computer for the Twenty-First Century. *Scientific American*, 265(3):94–104, 1991.