Journal of Web Engineering, Vol. 0, No. 0 (2010) 000–000

A Non-monotonic Expressiveness Extension on the Semantic Web Rule Language

Jose M. Alcaraz Calero *+, Andres Muñoz Ortega +, Gregorio Martinez Perez +, Juan A. Botia Blaya +, Antonio F. Gomez Skarmeta +

+ *Department of Information and Communication Engineering , University of Murcia, Campus de Espinardo s/n Murcia, 30100, Spain*
jmalcaraz, amunoz, gregorio, juanbot, skarmeta@um.es

* *Cloud and Security Lab, Hewlett-Packard Laboratories, Stroke Gifford, Filton Bristol, BS34 8QZ, United Kingdom*
jose.maria-alcaraz.calero@hp.com

SWRL (Semantic Web Rule Language) extends OWL syntax and semantics by enabling the description of Horn-like rules. However, the current SWRL specification lacks support for, among others, negative expressions, missing values and priority relationships between rules, which are frequently needed when modeling realistic scenarios. This paper motivates the necessity of surpassing some of these problems and provides an extension over the original SWRL aimed to define more expressive rules. Hence, the following four operators have been added to SWRL: *Not* operator (i.e., classical negation) to express negative facts; *NotExists* quantifier to ask for missing facts in the knowledge base (when used in the antecedent of the rule) and remove facts (when used in the consequent); *Dominance* operator to establish priorities among rules; and *Mutex* operator to establish exclusions during rule executions. The syntax and semantics of these four operators are described in this proposal. Moreover, the non-monotonicity added to the rule-based inference process by means of such elements is also explained. An implementation of the four operators has been developed as a plug-in for the Jena generic rule engine, which enables the execution of Horn-like rules, together with a parser to translate SWRL rules to the Jena specific rule language. Finally, the proposed SWRL extension and its implementation have been validated in a real scenario centered on call forwarding management in an intelligent building.

Keywords: Semantic Web Rule Language (SWRL), Ontology Web Language (OWL), Non-Monotonicity, Rule-based Inference Process
Communicated by: to be filled by the Editorial

1 Introduction

Semantic Web [4] languages such as OWL [3] and SWRL [11] have shown to be useful in managing knowledge and providing the semantics for reasoning about it. Such semantics enable computers to automatically obtain new knowledge over an application domain by means of powerful inference processes. To this end, OWL describes any application domain by means of ontologies. An ontology is the description of a given application domain by means of statements. The ontology is usually divided in two different set of statements called boxes. The terminological box (TBox) is the set of statement within the ontology used for describing the concepts, datatypes, relationship between concepts and semantics over the domain described.

The relationship between concepts can be a simple description of a feature using strings, integers or any other simple type, referred as "'literal"', or can be a complex desription using relationships between two concepts available in the domain. The assertional box (ABox) is the set of statement within the ontology used for describing particular scenarios of the domain, generally by means of instances (individuals in OWL jargon). Formally, an OWL ontology [25] is defined as the tuple $O =< R, LV, EC, ER, L, I >$ where R is the set of statements related to the application domain, $LV \in R$ is a set of literal values, EC is a mapping from classes and datatypes to subsets of R and LV respectively, ER is a mapping from properties to binary relations on R, L is a mapping from typed literals to elements of LV, and I is a mapping from individual names to elements of EC.

OWL-Lite, OWL-DL, OWL-Full, and late OWL 2 [20] (previously called OWL 1.1) are the main family of OWL languages. They provide different expressiveness and computational complexities. More specifically, OWL-Lite is a subset of OWL-DL, and in turn, the latter is a subset of OWL 2, which eventually is a subset of OWL-Full. From the point of view of the trade-off between expressiveness and computational complexity, OWL 2 presents more compelling characteristics than the other OWL languages. The main reason is that OWL-Full is not decidable whereas the remainder OWL languages are based on different decidable fragments of Description Logic (DL) [1]. Besides, OWL 2 is more expressive than OWL-Lite and OWL-DL. In this context, decidability is referred as the capability to perform inference processes in a finite time. OWL 2 offers a wide range of constructors such as inheritance among concepts and properties, transitive and inverse properties, cardinality restrictions, etc., while keeping within the decidability bounds.

On the other hand, SWRL has been designed as a syntactic and semantic extension of OWL languages in order to enable the definition of Horn-like rules. SWRL rules are in the form of an implication between a conjunction of antecedents (*body*) and consequents (*head*), meaning that whenever the conditions specified in the antecedents hold, the conditions specified in the consequent must also hold, i.e. the logical implication. Both conjunctions consist of *atoms* of the form $C(x)$, $P(x, y)$, $sameAs(x, y)$, $differentFrom(x, y)$ and $builtIn(r, x, ...)$, where C is an OWL class, P is an OWL property, r is a built-in function, and x, y are either variables, OWL individuals or OWL data values. Unfortunately, the combination of OWL and SWRL is undecidable, even when the OWL language adopted is decidable. However, it is possible to restore the decidability of this combination by restricting the variables in SWRL rules to only take values in named individuals of the component I of an OWL ontology (in OWL it is possible to define anonymous individuals). These restricted rules are known as *DL-safe*. To apply this restriction, it has to be ensured that only the variables previously declared in the antecedent part of the SWRL rule can be used lately in the consequent part. A complete explanation about this problem and its solution can be found elsewhere [21].

Despite of the high expressiveness provided by the combination OWL+SWRL, there are several constructors for describing knowledge and enabling a particular type of inference processes out of this combination. The main reason is the computational complexity and the undecidability associated to these constructors. Nonetheless, this lack results in an inaccurate knowledge management in many application domains. Let us see three informal examples of this handicap. Firstly, neither OWL nor SWRL allow for a deductive processes where a fact

is inferred from the absence of some other facts in a knowledge base. For example, the rule "*if there is no sign of Bob being localized in his office, then a call forward to his mobile phone must be done*" cannot be expressed in any of the studied languages. Secondly, the removal of knowledge is not considered in OWL or SWRL. For instance, the rule "*if Bob is localized in his office then remove all his call forwards*" cannot be expressed following this restriction. Finally, SWRL does not permit to define a priority order or an exclusion relationship among rules. The possibility of stating an order is useful when two or more rules can be fired at the same time and it is desired to establish a total (o partial) execution order, whereas the exclusion relationships prevent a rule from being executed if the rule excluding it has been previously fired.

All the examples given in the paragraph above are oriented to stress the necessity to obtain a more expressive rule language for the Semantic Web. Thus, the main goal of this paper is to provide SWRL with an expressive extension by means of new operators. These operators allows for capturing a broader range of knowledge and applying more types of inference processes in the application domain. As a result, the following four operators have been added to SWRL: Not operator (i.e., classical negation) to express negative facts; $NotExists$ quantifier to ask for missing facts in the knowledge base (when used in the antecedent of the rule) and remove facts (when used in the consequent); $Dominance$ operator to establish priorities among rules; and $Mutex$ operator to establish exclusions during rule executions. These operators have been implemented taking into account their semantics and they are ready to be used in rule-based engines which are able to evaluate SWRL-like format rules.

The rest of the paper is structured as follows: Section 2 describes some concepts on which this proposal is based. Section 3 provides a related work in expressive extensions for SWRL with special emphasis on non-monotonic extensions. A running example is given in Section 4 to facilitate the understanding of the extensions proposed. The syntax and semantics of the new operators are fully explained in Section 5. Section 6 provides a justification of the expressiveness proposed and its practical usage. The inference process to apply the semantics provided by the new operators is covered in Section 7. A complete scenario is depicted in Section 8 where all the new operators and their associated inference processes are applied with the aim of showing their added-value. Next, technical implementation aspects are explained in Section 9. Finally, Section 10 summarizes the contribution of this paper and outlines the future work.

2 Non-monotonicity and OWA

This section presents some concepts that will help to better understand the proposal given in the paper. The first one is *non-monotonicity*. In order to explain this concept, let us introduce some notation first. Let L be a logic language and let KB be a set of well-formed formulas in L (also referred as *facts* or *statements*) which represent the knowledge base of certain application domain. Let S be a function for evaluating the truth value of all the formulas contained in KB at time t. This function returns $True$ (T) or $False$ (F) according to the truth value of a formula x in KB. Let define V_t as the truth value at time t, and let define S_t as the formula S applied in the time t. Formally,

$$(S : S_t(x) = V_t \ / \forall x \in \mathrm{KB}_t \ , \ V_t \in \{T|F\}) \tag{1}$$

A logic L is monotonic if and only if the addition of a set of formulas in KB does not entail any change on the truth value of its previous existing formulas. Let us represent the addition of a set of formulas Φ in KB at time t as $KB_{t+1} = KB_t \cup \Phi$. Then, monotonicity implies that all the formulas in KB_t maintain their truth value in KB_{t+1}. Formally,

$$\forall x \in \mathrm{KB}_t, \forall y \in \mathrm{KB}_{t+1}, \text{if } x = y \text{ then } S_t(x) = S_{t+1}(y) \tag{2}$$

When a logic language does not fulfill the condition given in (2), it is considered as non-monotonic. Formally, L is a non-monotonic logic if the following condition can be applied in any case:

$$\exists x \in \mathrm{KB}_t, \exists y \in \mathrm{KB}_{t+1}, \text{such as } x = y \not\rightarrow S_t(x) = S_{t+1}(y) \tag{3}$$

The second concept presented in this section is the Open World Assumption (OWA). OWA assumes that the knowledge contained in a KB is not necessarily complete. The contrary approach is the Closed World Assumption (CWA), which assumes that a KB contains a complete knowledge about a domain. The OWA approach is normally adopted in open environments such as the Semantic Web, where new knowledge is constantly being discovered, reused and exchanged, and the knowledge already stated in a KB is considered as a partial vision of the application domain. As a matter of example to emphasize the differences between both approaches, suppose a KB which does not contain a formulas f. By using function S in (1), let $S(f) = k$ be the truth value of f. Then, the value of k is neither $True$ nor $False$ in OWA because the formulas $\{f\}$ or $\{\neg f\}$ could be later added to the KB. Contrarily, $k = False$ in CWA because it is not expected further additions to the KB and the default convention $Negation\text{-}as\text{-}Failure$ (NaF) is assumed here. This convention assigns the $False$ truth value to those formulas that are not explicitly stated in the KB.

Note that the adoption of OWA implies the definition of a third truth value in the range associated to the function S given in (1). This value is labeled as $Undefined$ (U). As a result, the function S could be redefined when adopting the OWA approach as described next:

$$S : S_t(x) = V_t \ / x \in \{KB\} \ , \ V_t \in \{T|F|U\} \begin{cases} V = T, & \text{if } x \in \mathrm{KB} \\ V = F, & \text{if } \neg x \in \mathrm{KB} \\ V = U, & \text{if } x \notin \mathrm{KB} \end{cases} \tag{4}$$

The definition of the function S in (4) assigns the truth value T (true) to positive facts stated in the KB, F (false) to negated facts stated in the KB, and U (undefined) to facts which are not stated in the KB. To conclude this section, let us now see the relationships among monotonicity and OWA/CWA. Normally, non-monotonic logics adopt the CWA approach. Nevertheless, there exist some works that combine this kind of logics with OWA [14, 5, 9]. On the other hand, monotonic logics usually adopt OWA. This is the case of the family of Description Logics. Since OWL is based on Description Logics, this language is also monotonic and adopts the OWA approach. Likewise, SWRL follows monotonicity and OWA as it is based on an extension of OWL.

Our intention is to relax the inherited monotonicity of SWRL in order to include some non-monotonic operators, thus incorporating new expressiveness in the rule language. These

operators are able to produce consequences in the knowledge base which imply a revision of the truth value of the facts contained therein, e.g. removal of facts. While operators of non-monotonic nature are well-known in CWA-based logics, their inclusion on OWA-based logics as DL remains an open issue. At the same time, there are many scenarios where these kinds of operators are needed in order to capture the knowledge of their application domain. For example, the scenario exposed in Section 4 is a clear example in which these operators are essential in the modeling of the application domain.

3 Related Work

There exist some theoretical researches related to the inclusion of non-monotonic reasoning in OWL such as the extensions of OWL expressiveness with abductive reasoning [5], reasoning by default [16], circumscription [29] and autoepistemic reasoning [9], [14]. All of them try to incorporate new expressiveness in OWL in order to enable additional reasoning processes which are not covered in the original OWL proposal. Thus, [5] provides a detailed description of the inclusion of abductive reasoning in OWL, but the authors do not offer the algorithm of the inference process which performs this reasoning in DL due to its complexity. On the other hand, the reasoning by default extension [16] provides a theoretical study and practical implementation to achieve this kind of reasoning in OWL DL. Moreover, autoepistemic reasoning extensions [9, 14] promote the inclusion of the k epistemic operator used to explicitly ensure the existence of facts and to deal with assumptions in the knowledge base. Finally, another extension to OWL [17] offers a conflict management for inconsistent knowledge based on a 4-valued logic.

It is worth mentioning the substantial gap between the practical and theoretical approaches, where the practical perspective presents a reduced group of implementations to demonstrate some of the previous works. There are some implementations of OWL reasoners such as Jena [18] or Hoolet [2] working on OWL-Lite expressiveness. The number of implementations is reduced for OWL reasoners working on OWL-DL expressiveness, being Bossam [13] or KAON2 [12] some examples. The incorporation of some of the theoretic features such as OWL 2 leads to a drastic reduction of the number of available reasoners, being Pellet [30] and FaCT++ [32] the only ones with a fully coverage for these features. Pronto [15] is an extension built on top of Pellet to support uncertainty and probabilistic capabilities. Regarding non-monotonic extensions, only Pellet in a commercial version provides a certain treatment of k operator as stated in [14] and certain treatment of reasoning by default as stated in [16].

On the other hand, there are some research works pointing to the addition of new expressiveness to SWRL. The most relevant proposals are fuzzy extensions [23], [31], mathematical extensions [26], constraint extensions [19], temporal extensions [27] and first-order logic extensions [24]. However, to the best of our knowledge, there are not efforts to provide SWRL with non-monotonic extensions. Thus, the main contribution of this proposal is to relaxing monotonicity in SWRL in order to enable the description of new operators of non-monotonic nature which augment the expressiveness of this language.

4 Running Example

The aim of this section is to provide a running example used in the rest of this paper. The scenario stresses the necessity of certain type of expressiveness in SWRL. It depicts the call forwards management in an intelligent building modeled by means of an OWL-DL ontology-based representation of the DMTF-CIM management standard [8]. The CIM standard is widely adopted in the industry for representing information systems. It is composed of some thousands of concepts and several thousands of properties related to information system

models. Some of the CIM properties are modelled as OWL classes rather than OWL properties into the ontology. The reason of this modellation is due to the simple fact that there are different kinds of properties in CIM and this differentiation cannot be modelled using regular properties in OWL language, thus the mapping to OWL classes is preferred by some authors.

The intelligent building is composed of floors which contain rooms. All the employees in the building have assigned a room and they may have a mobile phone. A landline telephone could also be available inside of each room. Moreover, in the building there are some software agents deployed to control different activities. Hence, there is a floor controller agent that monitors all the floor services. In addition, a user agent is also present in each room in order to manage the user's preferences in the building. The aim of this scenario is to provide an intelligent pervasive service for call forwarding. This service will be available for all the employees working in the building. The simplified version of the OWL definition of this domain is expressed using an abstract syntax in Table 1.

OWL Definition	Description
Class(Room), Class(Floor), Class(Building), Class(FloorController), Class(CallService), Class(CallDevice), Class(Identity), Class(Location)	All the resources available on the application domain. All these resources inherit from Class(ManagedElement). SubClassOf(Room, ManagementElement), SubClassOf(Floor, ManagementElement), SubClassOf(Building, ManagementElement), SubClassOf(FloorController, ManagementElement), SubClassOf(CallService, ManagementElement), SubClassOf(CallDevice, ManagementElement), SubClassOf(Identity, ManagementElement), SubClassOf(Location, ManagementElement)
Class(AssociatedLocation), Class(AllocatedResource), Class(ServiceAvailableToElement), Class(AssignedOffice), Class(Subsumption), Class(CallForward)	All the relationships among resources (modelled as classes). All these resources inherit from Class(Association) in order to represent a relationship on CIM. SubClassOf(AssociatedLocation, Association), SubClassOf(AllocatedResource, Association), SubClassOf(ServiceAvailableToElement, Association), SubClassOf(AssignedOffice, Association), SubClassOf(Subsumption, Association), SubClassOf(CallForward, Association)
ObjectProperty(elementLocated { domain(AssociatedLocation)}) ObjectProperty(location { domain(AssociatedLocation)}) ObjectProperty(resource { domain(AllocatedResource)}) ObjectProperty(allocatedIn { domain(AllocatedResource)}) ObjectProperty(serviceProvided { domain(ServiceAvailableToElement)}) ObjectProperty(userOfService { domain(ServiceAvailableToElement)}) ObjectProperty(assignedTo { domain(AssignedOffice)}) ObjectProperty(elementAssigned { domain(AssignedOffice)}) ObjectProperty(from { domain(CallForward)}) ObjectProperty(to { domain(CallForward)}) ObjectProperty(member { domain(Subsumption)}) ObjectProperty(collection { domain(Subsumption)})	These are all the properties associated to CIM associations used to establish relationship among resources. For simplicity, the range of all of them has been assumed as ManagedElement.
SubClassOf(Room,Location), SubClassOf(Floor,Location), SubClassOf(Building,Location)	These are the inherit relationships related to the localization

Table 1. OWL abstract syntax of the application domain used as running example

Figure 1 shows the scenario used as running example from the instantiation of the application domain described in Table 1. This scenario describes an ACME Tower building which is composed by twenty floors. (Floor4 and Floor16 are shown in the figure and the rest are omitted for simplicity). The composition relationship among ACME Tower and its floors has been modeled using the *Subsumption* association. Bob is an employee of this building, who has his own room (BobRoom) and his own mobile phone assigned. This room is located in Floor4. Each floor has a floor controller agent which manages the devices allocated therein. On the other hand, Bob's room has associated a personal agent (BobAgent) responsible for configuring the behavior desired by Bob over his available services. In this scenario, the available service in Bob's room is the call service for Bob's telephone. Analogously, Alice is another employee in the system, and she has her own room in Floor16. Her room has the same features as Bob's room, such as a landline telephone, the call service and a personal agent (AliceAgent). The elements related to Alice are not showed in Figure 1 for simplicity.

There are some RFID sensors spread throughout the building so as to detect the presence of users in a specific area. Building sensors are located in the external doors of the building and they detect users which are entering/leaving the building. Floor sensors are located near to the lifts and ladders detecting the presence of users in a specific floor. Eventually, room sensors are located near room doors and they provide the presence of users in such rooms. All these sensors have a controller that is in charge of updating the location of the user into the knowledge base. Actually, the pervasive system uses a system such as OCP (Open Context Platform) [22] for updating the knowledge base with the location information. OCP is a middleware system whose main goal is to manage the context information of the whole system. However, the disscusion of this issue is beyond the scope of this paper. For the purposes of this scenario, the sensor information is simulated by means of the following SWRL rules expressed in human readable syntax. We use these rules just for simplication enabling us to get an artifact to fire such events in the system when we want to fire them creating a controlled

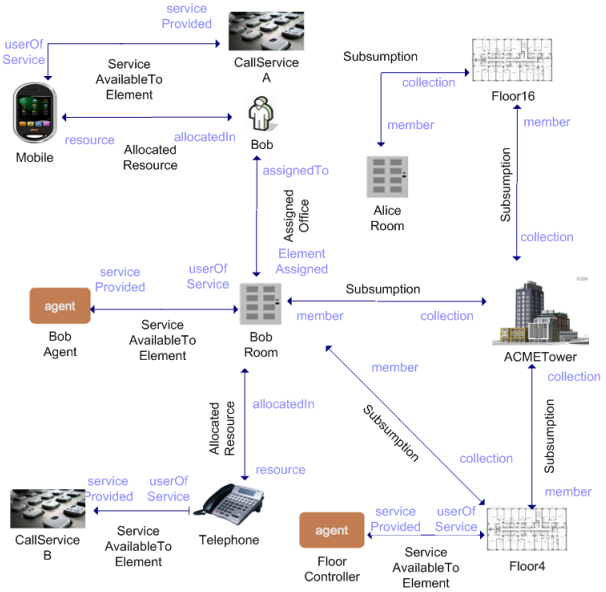

Fig. 1. The running scenario of an intelligent building

environment for testing. :

$$B1 :\rightarrow AssociatedLocation(?al) \wedge elementLocated(?al, Bob) \wedge location(?al, BuildTower)$$
$$F4 :\rightarrow AssociatedLocation(?al) \wedge elementLocated(?al, Bob) \wedge location(?al, Floor4)$$
$$F16 :\rightarrow AssociatedLocation(?al) \wedge elementLocated(?al, Bob) \wedge location(?al, Floor16)$$
$$RA :\rightarrow AssociatedLocation(?al) \wedge elementLocated(?al, Bob) \wedge location(?al, AliceRoom)$$
$$RB :\rightarrow AssociatedLocation(?al) \wedge elementLocated(?al, Bob) \wedge location(?al, BobRoom)$$

These rules will be used to activate the Bob's location according to our interest on different building sites. Observe that these rules have no antecedent. As a result, when the activation of one of these rules is decided, it will be automatically fired since the antecedent is trivially satisfied. The consequents of the rules create an *AssociatedLocation* association among Bob (referenced by the *elementLocated* property) and the place where he has been located (referenced by the *location* property). Thus, rule $B1$ simulates that the building sensor has detected Bob entering the building. Analogously, the rules $F4$, $F16$, RA and RB simulate that Bob has been detected in Floor4, Floor16, AliceRoom and BobRoom, respectively. Additionally, a *BreakService* rule is used to simulate that the call service of Bob's telephone is out of order, this rule will be described in Section 8.

Using this scenario, Bob could insert his preferences about call forwarding using his agent, and analogously, the system administrator could insert the preferences provided by the building service manager into the floor controllers. To this end, there are some rules that cannot be represented on the SWRL standard. In fact, in the following sections, some lacks about the expressiveness in SWRL language will be identified as necessary for enabling realistic scenarios such as the intelligent building one.

5 SWRL Expressiveness Extension

The current SWRL specification only allows the definition of rules whose atoms represent positive facts. By analogy with the function S given in (4), these facts are equivalent to those whose truth value is $True$ in such a function (i.e., positive facts stated in the KB). This section describes the non-monotonic expressiveness extension for SWRL proposed in this paper, with the aim of defining the operators that will enable the use of negative and missing facts in SWRL rule atoms (i.e., facts whose truth value is $False$ and $Undefined$ according to the aforementioned function S.) The explanation of this proposal has been divided into two subsections. The first one describes the extension developed over the syntax of the SWRL language. The second one describes the new semantics adopted on the new operators defined.

5.1 Syntax Extension

The abstract syntax is specified here by means of Extended BNF. Terminals are quoted; nonterminals are bold and not quoted. Alternatives are either separated by vertical bars (|) or are given in different productions. Components that can occur at most once are enclosed in square brackets ([...]); components that can occur any number of times (including zero) are enclosed in braces ({...}). Following the original SWRL specification[a], a rule term consists of an antecedent and a consequent, each of which consists of a (possibly empty) set of atoms. A rule can also be assigned to a URI reference, which could serve to identify such rule. Our syntax extends the antecedent and consequent definitions in order to enable the use of

[a]see `http://www.w3.org/Submission/SWRL/` for details about the original SWRL abstract syntax

quantifiers. A quantifier is applied over one or more atoms in order to provide them with a particular semantic. In this proposal the quantifiers $Exists$ and $notExists$ are allowed:

constructor ::= **rule** | **dominance** | **mutex**
rule::= 'Implies(' [*URIreference*] { **annotation** } **antecedent consequent** ')'
antecedent ::= 'Antecedent(' { **quantifier** } ')'
consequent ::= 'Consequent(' { **quantifier** } ')'
quantifier ::= **atom** | 'Exist(' { **atom** } ')' | 'NotExist (' { **atom** } ')'

Regarding atom definition, it has been also extended from the original SWRL proposal, which enables the description of $dataRange$, $individualvaluedPropertyID$, $datavaluedPropertyID$, $sameAs$, $differentFrom$ and $builtin$ atoms types. In our proposal two new kind of atoms have been added in order to allow for the definition of both negative object properties and negative data properties (see *not* atoms). Note that atoms may refer to individuals, data literals, individual variables or data variables. In the context, a *d-object* is referred to either a data literals or data variable and a *i-object* is referred to either an individual or a individual variable.

atom ::= *description* '(' **i-object** ')'
| '*dataRange* (' **d-object** ')'
| '*individualvaluedPropertyID* (' **i-object i-object** ')'
| '*datavaluedPropertyID* (' **i-object d-object** ')'
| 'sameAs(' **i-object i-object** ')'
| 'differentFrom(' **i-object i-object** ')'
| 'builtIn(' *builtinID* { **d-object** } ')'
| 'not(' *individualvaluedPropertyID* '(' **i-object i-object** ')' ')'
| 'not(' *datavaluedPropertyID* '(' **i-object d-object** ')' ')'

i-object ::= **i-variable** | *individualID*
d-object ::= **d-variable** | *dataLiteral*
i-variable ::= 'I-variable(' *URIreference* ')'
d-variable ::= 'D-variable(' *URIreference* ')'

On the other hand, two new types of constructors has been provided as extensions of the SWRL syntax: *dominance* and *mutex*. These constructors are proposed to achieve the definition of relationships between rules and their semantics will be later explained. The URI reference terminals serve to identify the rules of these relationships.

dominance ::= 'dominance(' *URIreference* , *URIreference* ')'
mutex ::= 'mutex (' *URIreference* , *URIreference* ')'

5.2 Semantics

The semantics provided in SWRL are a straightforward extension of the semantics for OWL given in the OWL Semantics [25]. The main extension of SWRL is the definition of bindings, which are extensions of OWL interpretations that also map variables to elements of the domain. Then, a rule is satisfied by an interpretation if and only if every binding that

satisfies the antecedent also satisfies the consequent.

Thus, let $T =< R, LV, EC, ER, L, I >$ be an OWL interpretation as defined in the introduction, where R is a set of resources, $LV \in R$ is a set of literal values, EC is a mapping from classes and datatypes to subsets of R and LV respectively, ER is a mapping from properties to binary relations on R, L is a mapping from typed literals to elements of LV, and I is a mapping from individual names to elements of EC. Then, a binding $B(T)$ is an abstract OWL interpretation that extends the tuple T such that I maps *i-variables* to elements of EC (individuals) and L maps *d-variables* to elements of LV, respectively. Then, an atom is satisfied by an interpretation T when certain condition are fulfilled. Table 2 shows the conditions for all the atoms provided in this proposal, where C is an OWL class, D is an OWL data range, P is an OWL individual-valued property, Q is an OWL data-valued property, f is a built-in relation, x, y are variables or OWL individuals, z is a variable or an OWL data value and S is the function given in (4) (see Section 2) which defines the truth value of the interpretation in the KB. Note that the atoms in Table 2 contain conditions on interpretation related to the $True$ truth value (rows 1-4 rows and 9), the $False$ truth value (rows 5-6) and the equality among truth values (rows 7-8). For example, the condition on the interpretation of atom $C(x)$ is the existence of an individual a in the KB (mapped into the variable x by the function I) such as there is a fact in the knowledge base asserting that a is belonging to the OWL class C, and for this reason the function S returns T.

# Row	**Atom**	**Condition on Interpretation**
1	$C(x)$	$I(x) \in EC(C) \wedge S(C(x)) = T$
2	$D(z)$	$I(z) \in EC(D) \wedge S(D(z)) = T$
3	$P(x, y)$	$< I(x), I(y) > \in ER(P) \wedge S(P(x, y)) = T$
4	$Q(x, z)$	$< I(x), L(z) > \in ER(Q) \wedge S(Q(x, z)) = T$
5	$not(Q(x, z))$	$< I(x), L(z) > \in ER(Q) \wedge S(Q(x, z)) = F$
6	$not(P(x, y))$	$< I(x), I(y) > \in ER(P) \wedge S(P(x, y)) = F$
7	$sameAs(x, y)$	$I(x) = I(y)$
8	$differentFrom(x, y)$	$I(x) \neq I(y)$
9	$builtIn(f, z_1, ..., z_n)$	$< I(z_1), ..., I(z_n) > \in D(r) \wedge S(r(z_1, ..., z_n)) = T$

Table 2. Interpretation Condition in Atoms

Regarding quantifiers, these provide an scope for a set of atoms which indicates how such atoms should be evaluated according to the semantics of each quantifier. Note that it is allowed to define atoms which are not under the scope of any quantifier (see the syntax referred to quantifiers in Section 5.1). This is allowed in order to provide backward compatibility with the original SWRL proposal. In that case, all these non-quantified atoms are implicitly defined by the scope of the quantifier $Exists$. The semantics provided by the quantifiers $Exists$ and $NotExists$ are shown in Table 3, where A is the set of the atoms scoped by the quantifier and B is the binding function which represents a binding of the atom a with the OWL interpretation T as previously described.

# Row	**Quantifier**	**Condition on Interpretation**
1	$Exists(A)$	$\forall a \in A \;\; \exists B(a)$
2	$Antecedent : NotExists(A)$	$\exists a \in A \;\; \nexists B(a) \leftrightarrow S(B(a)) = U$
3	$Consequent : NotExists(A)$	$\forall a \in A \;\; \nexists B(a) \leftrightarrow S(B(a)) = U$

Table 3. Interpretation Condition in Quantifiers

Exists quantifier produces the regular evaluation of SWRL rules since its semantics indicate that the rule is satisfied if and only if all the atoms scoped by this quantifier have a binding on the OWL interpretation I (KB). Note that a binding between an atom and an OWL interpretation implies that the fact bound to this atom is available into the KB. Formally, $\exists B(a) \leftrightarrow B(a) \in KB$. On the other hand, $NotExists$ quantifier has different semantics depending on its localization. Thus, in case $NotExists$ is located in the antecedent of the rule, it produces that all the atoms scoped by this quantifier will satisfy the rule condition if there is any atom for which there is not a binding in the OWL interpretation T. Note that the nonexistence of a binding in the OWL interpretation implies that this fact is not available in the KB and also for this reason, it is evaluated as *undefined (U)*. Formally, $NotExist(B(a)) \leftrightarrow B(a) \notin KB \leftrightarrow S(B(a)) = U$. Contrarily, when $NotExists$ is located in the consequent of a rule, it produces that all the atoms scoped by this quantifier are evaluated as $undefined$, which implies that such facts are removed from the KB. Note that the conditions on interpretation given in rows 2 and 3 of Table 3 are related to the $Undefined$ truth value. While the first case is fulfilled when any atom under the scope of this quantifier matches the condition on the interpretation, the second one fulfills the condition on the interpretation for all the atoms under the scope of this quantifier.

In order to describe the semantics associated to the new *Dominance* and $Mutex$ constructors, let us define F_t as the conflict set composed of all the rules that are fulfilled at the same time t. Thus, since *non-monotonicity* is being considered, the order in the execution of the rules should be taken into account. The reason for this resides in that the execution of a rule implies the re-evaluation of the truth value of the facts in the KB, and this re-evaluation could affect in the execution of other rules. Thus, *Dominance* operator is used to establish an execution order between all the rules of the conflict set. Formally, $Dominance(A, B) \leftrightarrow P(A) > P(B)$, where P is the function which determines the priority in the execution and A,B are rules. The rules without any dominance relationship are considered by default as the ones with the lesser priority. Then, the rule with the highest priority is the first to be executed. Formally, $\forall r \in F_t\ , r_i = max(P(r))$. Since r_i is the one with max priority, it is the one to be executed.

Regarding $Mutex$ operator, this is used to exclude a rule r_B from the conflict set F_t when the rule r_A associated to r_B through this term has been already executed. Let us define E as the set of rules that has been executed. Then, $Mutex(r_A, r_B)$ defines that if $R_A \in E$, R_B is excluded from the conflict set F_t. Formally, if $Mutex(R_A, R_B)$ and $R_A \in E$, then $R_B \notin F_t$. Note that $Mutex$ is not defined as a symmetric operator and for this reason $Mutex(R_A, R_B) \not\rightarrow Mutex(R_B, R_A)$.

6 Justification of the Proposed Expressiveness Extension

6.1 Negative Property Assertions

Recently, OWL 2 has added the capability to define negative property assertions in OWL ontologies. However, the current SWRL proposal does not cover this new expressiveness and it only enables the definitions of atoms related to positive assertions, i.e., evaluated as $True$ according to the function S given in (4). This fact has driven our proposal to include this capability in SWRL rules.

This extension has been already analyzed by offering the description of this operator in SWRL [24]. But this proposal does not describe the syntax of the extension and it leaves aside some essential descriptions such as how to evaluate semantically this operator and how to define an inference process to manage this expressiveness. Thus, we have decided to provide

this expressiveness extension despite of being an operator of monotonic nature.

In the running example of Section 4, Bob can insert a preference rule to control his call forwards. Thus, if the call service of the Bob's room telephone is not available (e.g., the telephone is switched off), Bob wants to receive all the calls of this room phone in his mobile phone. This rule is depicted as follows in the SWRL abstract syntax previously exposed. Note the presence of the operator *Not* in the rule:

$$BobPhoneFailure : Identity(Bob) \wedge AssignedOffice(?ao) \wedge assignedTo(?ao, Bob) \wedge elementAssigned(?al, ?r) \wedge Room(?r) \quad (1)$$
$$AllocatedResource(?ar) \wedge allocatedIn(?ar, ?r) \wedge resource(?ar, ?tp) \wedge CallDevice(?tp) \quad (2)$$
$$ServiceAvailableToElement(?sae) \wedge UserOfService(?sae, ?tp) \wedge \mathbf{Not}(ServiceProvided(?sae, ?ts)) \wedge CallService(?ts) \quad (3)$$
$$AllocatedResource(?ar2) \wedge allocatedIn(?ar2, Bob) \wedge resource(?ar2, ?mp) \wedge CallDevice(?mp) \quad (4)$$
$$ServiceAvailableToElement(?sae2) \wedge UserOfService(?sae2, ?mp) \wedge ServiceProvided(?sae2, ?ms) \wedge CallService(?ms) \quad (5)$$
$$\rightarrow \quad (6)$$
$$CallForward(?cf) \wedge from(?cf, ?ts) \wedge to(?cf, ?ms) \quad (7)$$

First line in *BobPhoneFailure* rule obtains the Bob identity and his room associated. Line 2 specifies the available resources in Bob's room. These resources are limited to a call device. In line 3 all call services out of order for this call device are obtained. This line introduces the usage of the new negative atom to represent the services which are not provided by the device. Analogously, line 4 obtains Bob's mobile phone (call device) and line 5 gets the call services available in this mobile phone. Lines 1-5 are used to describe the antecedent of the rule. When such an antecedent is fulfilled, the line 7 is instantiated with the facts that indicate a call forward among the call services in Bob's room and Bob's mobile phone. Note that this rule cannot be expressed in the original SWRL rules since it does not allow for the definition of the *Not* operator.

6.2 Not Exist Quantifier in the Antecedent

NotExist quantifier used on the antecedent of a rule has several implications on the inference process. Firstly, this quantifier produces that atoms are fulfilled when its condition of interpretation is evaluated as *undefined*. This truth value has sense in an OWA approach because of it could change when new knowledge is added to the KB. This possible change on the truth value makes *NotExists* a clear non-monotonic operator since its usage implies that the truth value of all the facts in the KB should be evaluated each time. Consequently, since rules of this nature are executed based on assumptions, when these assumptions are invalidated, the antecedent of the rules are not fulfilled anymore and therefore, they could be retracted as an approach for restoring the consistence in the knowledge base. For this reason, rule retraction has to be managed in the inference process as it is later described in section 7.

The following rule *UserMissing* can be defined as part of our running example containing the operator described here. The rule establishes a call forward on mobile phones for those employees who are not located in the building:

$$UserMissing : Identity(?i) \wedge AlocatedResource(?ar) \wedge allocatedIn(?ar, ?i) \wedge resource(?ar, ?md) \wedge CallDevice(?md) \quad (1)$$
$$\wedge ServiceAvailableToElement(?sae) \wedge UserOfService(?sae, ?md) \wedge ServiceProvided(?sae, ?ms) \wedge CallService(?ms) \quad (2)$$
$$\wedge AssignedOffice(?ao) \wedge assignedTo(?ao, ?i) \wedge elementAssigned(?aol, ?r) \wedge Room(?r) \quad (3)$$
$$\wedge AllocatedResource(?ar2) \wedge allocatedIn(?ar2, ?r) \wedge resource(?ar2, ?td) \wedge CallDevice(?td) \quad (4)$$
$$\wedge ServiceAvailableToElement(?sae2) \wedge UserOfService(?sae2, ?td) \wedge ServiceProvided(?sae2, ?ts) \wedge CallService(?ts) \quad (5)$$
$$\wedge \mathbf{NotExists}(AssociatedLocation(?al) \wedge locatedElement(?al, ?i) \wedge location(?al, ?x) \wedge Location(?x)) \quad (6)$$
$$\rightarrow \quad (7)$$
$$\wedge CallForward(?cf) \wedge from(?cf, ?ts) \wedge to(?cf, ?ms) \quad (8)$$

Line 1 gets the call devices associated to each identity available in the system. Note that the mobile phone are associated to identities whereas the room telephones are associated to rooms. For this reason, line 1 gets only the mobile phones. These devices have certain services, and one of them is the call service obtained in line 2. On the other hand, the rooms of each identity are provided in line 3. Each room has a call device inside (telephone), and it is identified in line 4. In turn, these call devices have associated services, and these services are obtained in line 5. Finally, line 6 asks for an *AssociatedLocation* instance in the knowledge base whose associated identity i is the user previously identified in line 1. Line 6 is fulfilled when there is not evidence of the localization of this user in the building (there are not *AssociatedLocation* instances present in the knowledge base). Then, when all these antecedents are fulfilled, a call forward will be made in line 8 among the user's mobile phone and the telephone available in her office. Note that this expressiveness cannot be described in standard SWRL and it is needed to improve the call management service.

6.3 Not Exist Quantifier in the Consequent. Removal of Knowledge

NotExists quantifier used on the consequent of a rule entails several issues on the inference process. Firstly, the use of this quantifier ensures that the facts scoped under this quantifier have a truth value $undefined$, and therefore all of them are removed from the KB. Secondly, note again that this removal produces that all the truth values have to be processed as assumptions since their values can change to $undefined$ when this operator is applied. This fact leads to the need of managing rule retraction in the inference process due to it could cause that rules previously fulfilled, now after the execution of another rule, they are not fulfilled anymore and they are retracted (see section 7 for a more detailed explanation).

It is worth mentioning that $DL\text{-}Safe$ context has been extended for this reason and this quantifier can contain unbound variables which have not been previously defined in the antecedent of the rule. The explanation of this extension and the semantics given to these variables will be also explained in section 7.

Hence, by the inclusion of this quantifier in the consequent, the following rule can be expressed as part of our running example. It establishes the call service behavior when a user is in a room located on a different floor from the one where his assigned office is located. In this case, if the user is located in a room where there is a telephone, a call forward is done to the telephone available in the current location and all the previous call forwards are removed.

$$FloorForward : Identity(?i) \wedge AssignedOffice(?ao) \wedge assignedTo(?ao, ?i) \wedge elementAssigned(?ao, ?ra) \wedge Room(?ra) \quad (1)$$
$$\wedge Subsumption(?ss) \wedge Member(?ss, ?ra) \wedge Collection(?ss, ?fa) \wedge Floor(?fa) \quad (2)$$
$$\wedge AssociatedLocation(?al) \wedge locatedElement(?al, ?i) \wedge location(?rb) \wedge Room(?rb) \quad (3)$$
$$\wedge Subsumption(?ss2) \wedge Member(?ss2, ?rb) \wedge Collection(?ss2, ?fb) \wedge Floor(?fb) \wedge swrl : notEqual(?fa, ?fb) \quad (4)$$
$$\wedge AllocatedResource(?ar) \wedge allocatedIn(?ar, ?rb) \wedge resource(?ar, ?pb) \wedge CallDevice(?pb) \quad (5)$$
$$\wedge ServiceAvailableToElement(?sae) \wedge UserOfService(?sae, ?pb) \wedge ServiceProvided(?sae, ?sb) \wedge CallService(?sb) \quad (6)$$
$$\wedge AllocatedResource(?ar2) \wedge allocatedIn(?ar2, ?ra) \wedge resource(?ar2, ?pa) \wedge CallDevice(?pa) \quad (7)$$
$$\wedge ServiceAvailableToElement(?sae2) \wedge UserOfService(?sae2, ?pa) \wedge ServiceProvided(?sae2, ?sa) \wedge CallService(?sa) \quad (8)$$
$$\rightarrow \quad (9)$$
$$\mathbf{NotExists}(CallForward(?x) \wedge from(?x, ?sb) \wedge to(?x, ?y)) \quad (10)$$
$$\wedge CallForward(?cf) \wedge from(?cf, ?bs) \wedge to(?cf, ?as) \quad (11)$$

First line identifies the room associated to each identity in the system. Line 2 obtain the floor in where this room is located. Next two lines are referred to the actual location of the user in the building, room and floor respectively. Lines 5, 6, 7 and 8 catch the call services associated to the call devices available on each room. In case the user is located on a different floor from his own floor, all the previous call forwards are removed, see line 10. Then, a new

call forward will be done in the visitor room, see line 11. Note the usage of this new quantifier in the line 10. In this line there are two different unbounded variables, $?x$ and $?y$. These variables represent all call forwards which have a *to* relationship and they are removed only $from$ the user's telephone room.

6.4 Dominance Operator. Courteous Priority

Dominance operator has been designed following the courteous priority [10]. This kind of priority offers the possibility to establish dominance relationship among rules to decide the order in the rule execution plan. In a monotonic approach, when two or more rules can be fired in a given time, all of them are fired because a rule execution can not invalidate the rest of rules which are later executed. However, in non-monotonicity, one rule execution could lead to the non-fulfillment of other rules. Then, the order can alter the output result. In this case, a method to describe the desired execution order in rules have to be included. To this end, the dominance operator defined in Section 5.1 offers a priority mechanism among rules. This operator is defined as $dominance(Rx, Ry)$ where Rx and Ry are two rule names and the meaning is that Rx has more priority in the execution order than Ry.

It is worth mentioning that the dominance operator has been designed to establish relative dominances rather than absolute ones. In the absolute approach, a rule contains a given priority, e.g. numerical priority, whereas in the relative approach, a relationship is created between two rules. In absolute dominance approaches it is required that the administrator knows the whole rule set in order to decide the assigned priority of each rule. Then, by inserting new rules in the domain, the absolute priorities could require changes to accommodate the priorities of such new rules. Contrarily, relative dominance is less sensitive against the arrival of new rules. This fact is due to the definition of the dominance is performed by relationships among pairs of rules and further rules can be relatively ordered with relation to a subset of the already existing ones. This relative approach is clearly better for the Semantic Web where new information is constantly discovered, inserted and updated. In this case, relative dominance is more suitable for dynamic changes on the rules set. Furthermore, this kind of dominance applies transitivity among priorities to determine the priority associated to a rule. For example, given the priority relationships $dominance(Rx, Ry)$ and $dominance(Ry, Rz)$, they imply $dominance(Rx, Rz)$. As explained in Section 5.2, rules without a defined priority are assigned the minimal priority and they are the latest to be executed.

In our running example, let us suppose that Bob is located in a room in Floor16 (it is a different floor from the one where his room is located). Suppose also that Bob's room telephone is out of order (the service call is not provided therein). In this case, two different rules can be fired: $BobPhoneFailure$ and $FloorForward$. Thus, if **Dominance**($BobPhoneFailure, FloorForward$) is established , then the call forwarding to the Bob's mobile phone is done by the dominance of the rule $BobPhoneFailure$. Otherwise, if the dominance is inversely done, **Dominance**($FloorForward, BobPhoneFailure$), then the call forward will be done to the telephone located in the room in which Bob is located in this moment. Note that the dominance relationship has influence on the results obtained and it needs to be taken into account in the inference process.

6.5 Mutex Operator

$Mutex$ operator is used in order to establish a blocking relationship between two rules. Thus, when one of them is already executed, the other cannot be executed (it is *blocked*). This kind of relationship enables to get a fine grain control in the rule execution. Thus, $Mutex(Rx, Ry)$ states that Ry will not be executed in case that Rx was already executed. In order to define

a symmetric exclusion between two rules, a double mutex relationship has to be asserted: $Mutex(Rx, Ry)$ and $Mutex(Ry, Rx)$.

In the running example of Section 4, this double $Mutex$ relationship can be established between **Mutex**($FloorForward, BobPhoneFailure$) rules and vice versa, **Mutex**($BobPhoneFailure, FloorForward$). Then, only one rule is fired and the other one will be blocked. Note that the usage of this operator can be combined with the dominance operator so as to determine which one of these two rules must be executed, leaving blocked the other one. Moreover, note also that mutex and dominance operators can be used in order to control the decidability in the inference process since they enable to block and schedule the rule execution plan, which is a suitable feature for the inference process.

7 Non-monotonic Inference Process

Usually, inference processes over SWRL have been performed by means of rule-based engines, which in turn, are essentially an implementation of a Rete-based [6] algorithm or another pattern matching algorithm. Some examples of SWRL-suitable implementations are Jena [18], Pellet [30] or Jess [7], among others. All of them are using a Rete-based algorithm to perform the inference process of SWRL rules. The Rete algorithm is an efficient pattern matching algorithm that creates a network of nodes in memory where each node correspond to a condition (antecedent atom) of a rule. Then, the path from the root node to the leaf nodes defines a complete antecedent part of a rule. Additionally, each node has a memory of facts which satisfy that pattern (working memory). Then, when a fact or combination of facts causes all of the patterns for a given rule to be satisfied, a leaf node is reached and the corresponding rule is triggered. Hence, when a rule is triggered, all the atoms of the body are processed performing the action associated to these ones. Moreover, since SWRL is a declarative language, the associated action to these atoms is limited to process the KB, asserting and removing facts.

The Rete-based algorithm exhibits the following characteristics: i) it reduces or eliminates certain types of redundancy through the use of node sharing in the network. ii) it allows avoiding complete re-evaluation of all facts each time changes are made in the working memory. Instead, the inference process only need to evaluate the changes (deltas) to working memory. iii) it allows for efficient removal of knowledge when facts are retracted from working memory.

The intention of this section is not to provide a new algorithm to build a highly efficient Rete network or to provide a full description of any version of this algorithm because this is out of the scope of this proposal. In case the reader is more interested in the algorithm to built a Rete network, [6], [33] and [28] provide different algorithms to implement this network node efficiently. However, this section tries to explain which are the changes that has been done to a Rete-based algorithm in order to enable the inference process over the expressiveness provided in this proposal. The following subsection will describe how to perform the different adaptations over this algorithm.

7.1 Different types of α-nodes

Rete network is usually composed by α-nodes and β-nodes also called left and right nodes or 1-input and 2-inputs nodes, respectively. On the one hand, α-nodes perform the pattern matching between the atom of the antecedent of the rule (pattern) and the facts available on the KB based on simple conditional tests. On the other hand, β-nodes are in charge of joining conditions in order to minimize the size of the network and to increase the performance and optimization (the latter are optional and used only for optimization purposes, for this reason they are left aside in this proposal).

In general, traditional Rete network manage one type of α-node to perform the pattern matching of facts (refered as α_+-node). However, in order to establish a Rete network suitable for the Semantic Web, we have designed new different types of α-nodes in this proposal. In particular, each new type of α-node performs a different test condition during the pattern matching. Table 4 described the different α-node proposed and the condition test associated. All of them are new ones except the α_+-node provided in the original Rete network.

α-node Type	SWRL Elements	Pattern Matching (P)	Condition Test
α_+-node	$C(x)$	(?x rdf:type C)	$S(P) = T$
	$D(z)$	(?x rdf:type D)	
	$P(x, y)$	(?x P ?y)	
	$Q(x, y)$	(?x Q ?y)	
	$sameAs(x, y)$	(?x owl:sameAs ?y)	
	$differentFrom(x, y)$	(?x owl:sameAs ?y)	
α_--node		(?z rdf:type owl:NegativeProperty)	$S(P) = F$
	$not(P(x, y))$	(?z rdf:subject ?x)	
	$not(Q(x, y))$	(?z rdf:predicate [P or Q])	
		(?z rdf:object ?y)	
α_u-node	$NotExist(A, B, ...Z)$	(A)	$S(P) = U$
		(B)	
		...	
		(Z)	
α_b-node	$builtIn(r, z1, ..., zn)$	N/A	$S(R(z1, ..., zn)) = T$

Table 4. Types of α-nodes proposed

The first column of the table 4 contains the name of different types of α-nodes proposed. Note the existence of three different α-nodes, each one associated to the evaluation of a concrete truth value (T, F, U) and one additional α-node to process built-in functions. The second column shows all the SWRL elements that produces the insertion of a new node of this type in the Rete network. For example, in case a rule contains $not(Q(x, y))$ as atom on the antecedent part, a new α_--node is inserted on the node network. The third column describes the pattern that is processed in the α-node for the SWRL element associated. This pattern is directly related to the manner in which the OWL interpretations are stored in the KB. So, the patterns associated to the α_+-node and α_b-node were described in the OWL [3] standard. The ones related to α_--node were described in the recently OWL 2 [20] proposal. Note that the pattern contains a

$$PorQ$$

, this means that will be P or Q depending on the SWRL elements to be inserted in the Rete network, $P(x, y)$ or $Q(x, y)$, respectively. The pattern related to α_u-nodes does not entail any other additional pattern matching since this node is inserted when a quantifier is processed and this quantifier has a term under its scope (see A,B,...,Z in the table 4, they are the atoms

associated to this terms). Therefore, the pattern associated to the atoms of the term are the patterns that produce the activation of the $\alpha - node$. Finally, the forth column establish the condition test that should pass the pattern in order to fulfil the α-node where S function is the one labelled as 4. Note that these α-nodes cover all the truth values available under the range of S function. This fact turn this set of α-nodes suitable to be used on a rule-engine adapted to perform inference processes in the Semantic Web.

7.2 Extending DL-Safe Context

DL-Safe context has been presented in [21] in order to restore decidability in the inference process associated to SWRL rules. This proposal imposes some limitations on the design of SWRL rules for ensuring the decidability. Mainly, DL-Safe imposes that a variable can be used in the consequent, if and only if, it has been previously declared in the antecedent of the rule. This imposition prevent the appearance of unbound variables in the consequent part because of this appearance could produce undecidability in the inference process.

Undecidability related to the usage of unbound variables is due to management done over these variables during the inference process. Usually, when an unbound variable z is available in a consequent of a given rule, this one is automatically bound to a new *individual ID* each time the rule is activated, causing the insertion of the new facts in the KB. This new fact could cause that the antecedent of this rule is fulfilled again and then it would enter in an infinite execution loop which in turn, it leads to undecidability in the inference process.

However, current DL-Safe context is not suitable in OWL 2 where the representation of a negative property assertion in the consequent of a rule force the appearance of an unbound variable in the consequent part of the rule. As the reader can see in the table 4, the pattern matching associated to $\alpha\_$-node (related to negative property assertions such as $not(P(x,y))$) requires that $?z$ variable have to be used in the pattern matching in order to produce the fulfilment of this node. Then, when this assertion is done in the consequent part of the rule, this new $?z$ variable produces a violation of this DL-Safe context since it is being used a non previously undeclared variable in the antecedent part.

For this reason, an extension of the DL-Safe context has been promoted. This extension enables to use unbound variables available in $\alpha\_$-nodes. The unbound variables related to negative property assertions has been syntactically hidden. Hence, $?z$ variable previously described does not appear in the SWRL syntax related to elements $not(P(x,y))$ and $not(Q(x,y))$ and for this reason, this variable can not be used in any other place. Consequently, this will not cause undecidability in the inference process due to this variable is not used in any other place and therefore, the side effects can be controlled.

Regarding $NotExists$ quantifier, it is also promoted the extension of DL-Safe context to enable the definition of unbound variables in atoms that are under the scope of this quantifier. Note that this quantifier produces the removal of knowledge from the KB. Then, in case unbound variables appears in these atoms, the following semantics is applied: this variable matchs with all *individual ID* available in the KB. Formally, being z the unbound variable under the scope of a $NotExists$ quantifier and a any *individualID* in the KB, then $\forall a \in KB, I(z) = a$ being I the function previously introduced in section 5.2 which map variables to individuals. By adopting this extension of the DL-Safe context, there is not any modification in the content of the KB and for this reason decidability is also preserved.

7.3 Conflict Set

The conflict set is composed by the set of fulfilled rules in a given time during the inference process. After the creation of the conflict set, the Rete algorithm should decide what is

the next rule to be executed. In the original SWRL inference process, the order in the rule execution was not taken into account because the output in the inference process could not be altered by the execution order of the rules. Now, when *non-monotonicity* is adopted, it might be taken into account since the output in the inference process could be affected by the execution order.

Thus, the Rete algorithm establishes the execution order according to the information provided by the usage of the *Dominance* operator. Then, after the rules have been ordered according to the dominance information, all the rules affected by the usage of the $Mutex$ operator is removed from the conflict set. In case in which there is some rule with the same priority, the order is determined by means of the rule name. This order based on the rule name has been decided to provide a deterministic approach rather than a indeterministic random order. Finally, the first rule in the ordered list is selected to be executed in the inference process. Note that rule retraction is considered as negative execution of rules and for this reason, these retracted rules are also inserted in the conflict set and ordered according to their priority. This fact is explained in the section 7.4.

7.4 Rule Retraction

The inclusion of the capability to remove knowledge ($NotExists$ quantifier) in SWRL language leads to the necessity of revising the knowledge base after the each rule execution, that is, it leads to the non-monotonicity. Then, during this revision of knowledge some rules previously activated, now may be not fulfilled because of some of the facts that causes their activation have been deleted. In that case, these rules have to be retracted in order to keep the KB consistent. Rule retraction is referred as the action for which the inferred knowledge produced by firing a given rule is dropped.

In order to achieve the rule retraction, let's define an inverse atom A^- of a given atom A, as the atom that cause the drop of the facts produced by the execution of the atom A. Thus, being C the set of all the atom of the consequent part of the rule R, a rule retraction R^- is defined by the execution of the inverse atoms of the all the atoms in C. Formally, $\forall A \in C/\ E(A^-)$ being E the function that executes a given atom. Thus, a rule retraction produces that the inverse rule R^- is inserted in the conflict set. The table 5 shown all the atoms available in the proposed language and the inverse atoms associated to be executed in the rule retraction. For example, if an atom $C(x)$ have to be retracted then $NotExists(C(x))$ might be executed.

Atom (A)	Inverse Atom (A^-)
$C(x)$	$NotExists(C(x))$
$D(z)$	$NotExists(D(z))$
$P(x,y)$	$NotExists(P(x,y)$
$Q(x,y)$	$NotExists(Q(x,y)$
$sameAs(x,y)$	$NotExists(sameAs(x,y))$
$differentFrom(x,y)$	$NotExists(differentFrom(x,y))$
$not(P(x,y))$	$NotExists(not(P(x,y)))$
$not(Q(x,y))$	$NotExists(not(Q(x,y)))$
$NotExist(A,B,...Z)$	$A,B,...,Z$
$builtIn(r,z1,...,zn)$	$Builtin(r\_,z1,....,zn)$

Table 5. Relationship between rule atoms and its inverses for performing rule retractions

During the rule retraction, the execution of the inverse atoms have to be done using the

same facts bound to the variables that previously was causing the firing of the rule. In Rete algorithm, the facts that produced the activation of a node in the Rete network are stored in the own node in the so-called working memory. Then, this information could be retrieved lately in order to perform the execution of these inverse atoms with the variables bound accordingly.

Note in the table 5 that $NotExists$ quantifier has as inverse atom $Exists$ operator and vice versa. Moreover, it is worth mentioning another change required in the inference process for implementing rule retracting. This is the extension of the definition of the built-in functions allowed in the consequent. Thus, $r\_$ built-in is defined as the built-in function that produces the drop of the action produced by the builtin r. For example, built-in *add* could have as inverse *subtract*. Hence, each built-in available in SWRL might have associated the action to be taken in case its retraction have to be done.

Rule retraction implies again the appearance of undecidability in the inference process. In particular, let's suppose two rules defined as: $Rx : A \Rightarrow B^-$ and $Ry : B^- \Rightarrow A^-$ being A, B rule atoms and A^-1, B^- their inverse atoms. Then, just inserting A in the KB, Rx is activated, and then, Ry is also activated. After that, Rx is retracted and consequently, Ry is also retracted leading to the initial state. This will cause an infinite loop in the rule execution plan which in turn, will lead in the undecidability of the inference process.

In order to control this issue, it is provided some operators related to the management of the execution of the rules. In particular, *Dominance* and $Mutex$ operators enable to perform a fine control in the execution plan of the rules. This control is used to drive the behaviour of the inference process according to the administrator preferences and it can be used to establish the decidability in the inference process since these operator enable to block any loop during the inference process. For example, the previous example can be controlled just defining $Mutex(Ry, Rx)$ operator. This one causes that Ry will not be fired and the loop will be controlled.

8 Complete Scenario

This section shows all the rules described in the previous sections working together. These rules contain all the elements provided in the new expressiveness provided in this proposal. All of them are executed in the inference process described in the section 7. As a result, the main intention of this section is to remark the necessity of the proposed expressiveness extension for modelling realistic scenarios.

The scenario is composed of several parts previously described in this paper. First, the running example exposed in section 4. This scenario represents an intelligent building with several floors and employees which have different kinds of phone services. These services are managed in order to perform call forwards among them. The scenario is completed with the following additional information. Regarding the initial localization of the users across the building, there are not location information available for Bob. Alice is initially located in her room by means of an *AssociatedLocation* association among *Alice* and *AliceRoom*. Regarding the simulation of the localization of Bob, rules $B1$,$F4$,$F16$,RA, RB and *BreakService* exposed in the same section are used to simulate the Bob's presence in different places. The complete definition of the *BreakService* rule is the following one:

$$BreakService : Identity(Bob) \wedge AssignedOffice(?ao) \wedge assignedTo(?ao, Bob) \wedge elementAssigned(?ao, ?r) \wedge Room(?r) \quad (1)$$
$$\wedge AllocatedResource(?ar) \wedge allocatedIn(?ar, ?r) \wedge resource(?ar, ?cd) \wedge CallDevice(?cd) \quad (2)$$
$$\wedge ServiceAvailableToElement(?sae) \wedge UserOfService(?sae, ?cd) \wedge CallService(?cs) \quad (3)$$
$$\rightarrow \quad (4)$$
$$NotExists(ServiceProvided(?sae, ?cs)) \wedge Not(ServiceProvided(?sae, ?cs)) \quad (5)$$

Line 1 retrieves the room assigned to Bob identity. Next line obtains the call device available in that room. The call services offered by this call device are obtained in line 3. For each call service, this rule causes the disabling of that call service (see line 5). This line replaces the *serviceProvided* property used to establish an association toward devices and services by the negative property. This negative property indicates the device is not providing the service associated (out of order). Note that the $BreakService$ can not be expressed in standard SWRL since it contains remove and negative operators.

Moreover, the scenario is completed with the rules explained in section 5: $BobPhone\text{-}Failure$, $UserMissing$ and $FloorForward$. As a brief summary, while $UserMissing$ performs a call forward to users' mobile not located in the building, $BobPhoneFailure$ performs a call forward to the user's mobile in case his room telephone is out of order. Additionally, $FloorForward$ is carrying out a call forward to the room in which users are located in case this room is located in a different floor from his own one.

There are one additional consequent atom that should be included in the $BobPhone\text{-}Failure$ rule in order to represent a real scenario. That is the removal of all the previous existent call forwards. This atom is inserted to give a realistic aspect to the scenario, in which at maximum, one call forward will be performed over a phone service. To this end, just the line 10 of the $FloorForward$ is also inserted in the $BobPhoneFailure$ rule. Moreover, a dominance relationship from $FloorForward$ over $BobPhoneFailure$ is taken into account in the scenario (this relationship was explained on section 6.4). Additionally, a mutex relationship between $FloorForward$ and $BobPhoneFailure$ is supplied in the scenario (this one was introduced in section 6.5).

The scenario is inserted in the proposed inference process in order to manage the call forwards on the intelligent building. This insertion describes the inference process step by step. The execution plan carry out in this process is shown in the figure 2. The figure is composed of a set of boxes. Each one represents the state of the knowledge base. The first box represents the initial KB. On each box, the CS (Conflict Set) is specified indicating the fulfilled rules. Boxes are linked by means of arrows. Each arrow indicates that a rule of the conflict set is fired/retracted. In case the label is cross out the rule is retracted, otherwise it is fired. This arrow ends with a new knowledge base (box). Bold facts represent the new facts with respect to the previous step and in case they are cross out, this means that have been removed from the KB. Note that all the facts representing the scenario exposed in figure 1 and the localization of Alice in her room are not present in the initial KB represented in figure 2 for clarity.

Firstly, since Bob is not present in the building, the rule $UserMissing$ (see n.1 in figure 2) is fulfilled and fired. This execution causes the insertion of a $CallForward(cf1)$ from the Bob's office phone and the Bob's mobile phone. When Bob is getting back to the building, rule $B1$ (see n.2 in figure 2) is fired to simulate the sensor detection and a new $Associated\text{-}Location(al1)$ is inserted in the knowledge base (now, Bob is located on the building). Thus, the $UserMissing$ rule is invalidated due to the existence of $AssociatedLocation(al1)$ upon Bob identity. This invalidation causes the retraction of the $UserMissing$ (see n.3 in figure 2) rule and in turn, that causes the deletion of the $CallForward(cf1)$ among the Bob's office phone and the Bob's mobile phone.

Now, Bob decides to visit his friend Alice using the lift to the sixteenth floor. Then Bob is detected by the floor sensor firing $F16$ (see n.4 in figure 2) when he leaves the lift at floor sixteen. This detection does not cause the fulfilment of any rule apart from the inclusion of an $AssociatedLocation(al2)$ instance according with the localization information. By entering

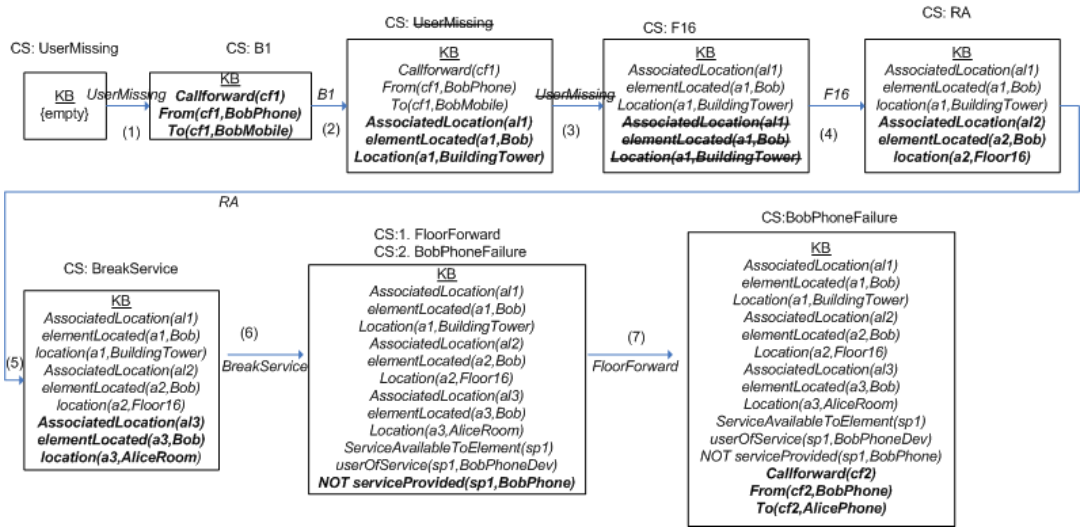

Fig. 2. The execution plan for the intelligent building scenario

on the Alice's room, the room sensor detects Bob and it is fired the RA rule (see n.5 in figure 2). Again, RA produces the insertion of a new localization information represented by means of an $AssociatedLocation(al3)$ instance between Bob and Alice's room. At the same time, the Bob's room phone is getting out of order by means of rule $BreakService$ which replaces the $serviceProvided$ associated to the Bob's room phone property by $Not(serviceProvided(sp1))$ (see n.6 in figure 2)

Bob is now located in the Alice's room having his room phone out of order. In that case, more than one rule is fulfilled at the same time. Concretely, $FloorForward$ (defined in section 6.3) and $BobPhoneFailure$ (defined previously in this section) rules are candidates to be executed. By the dominance established in the scenario, $FloorForward$ dominates $BobPhoneFailure$ and this one is executed. The $FloorForward$ rule is fired first (see n.7 in figure 2) causing the appearance of a call forward from the Bob's room phone to the Alice's room phone ($CallForward(cf2)$) . After that, $BobPhoneFailure$ still remain fulfilled and the mutex operator avoid the $FloorForward$ to be executed.

As reader can notice, Bob is receiving all his calls independently of his location. For example, when he is located at his office, or out of the building or even in the office of his friend, he continues receiving calls by means of the management of the call forward service.

This scenario remarks the necessity for describing rules with more expressiveness than the provided in standard SWRL. Most of the rules used in this scenario cannot be represented in the standard SWRL language. Therefore, by supporting the description of these rules in this proposal, some lacks of expressiveness have been overcome. The scenario has been successfully tested and validated in the implemented inference process offering an added value to the business since really the employee is more available to the customer and other colleagues by the use of call forwards.

9 Implementation

As a proof of concept, a prototype implementation of the non-monotonic inference processes explained in section 7 has been developed. This implementation is named JNOMO [b] inference engine. JNOMO stand for *Jena Nonmonotonic Extension* and it is a free, open source project released under GPL license.

Jena [18] is a Semantic Web framework which, among other features, contains a Horn-like rule engine based on an implementation of a Rete network. This rule engine is called *GenericRuleEngine*. JNOMO has extended this Jena engine by providing a new Rete network implementation.

The current *GenericRuleEngine* has a Rete network which only enables to process α_+-nodes and α_b-nodes (see Section 7). Moreover, this engine does not have any kind of scheduling in the conflict set and it does not support rule retractions. This implementation has been extended by JNOMO with a new implementation of this Rete network which is able to process all the α-nodes proposed in Table 4. This extension enables to perform scheduling in the conflict set by means of the usage of the information of the $Mutex$ and *Dominance* operators included in the proposed SWRL syntax given in Section 5.1. Additionally, this new Rete network implementation enables to perform rule retractions enabling the execution of the inverse atoms associated to the rule that have to be retracted.

Regarding rule syntax, the *GenericRuleEngine* has its own rule syntax called Jena rule syntax. Then, a simple parser has been built in order to perform a translation from SWRL XML syntax to this format. Moreover, this rule syntax has been extended in order to manage all the expressiveness extension provided in this proposal, including negative property assertions, $NotExists$ quantifier and retractable built-in functions. Table 6 shows the extension provided to the Jena rule syntax in order to enable the description of these operators.

SWRL Elements	Jena Rule Syntax
$C(x)$	$(C\ rdf:type\ ?x)$
$D(z)$	$(D\ rdf:type\ ?z)$
$P(x,y)$	$(?x\ P\ ?y)$
$Q(x,y)$	$(?x\ Q\ ?y)$
$sameAs(x,y)$	$(?x\ owl{:}sameAs\ ?y))$
$differentFrom(x,y)$	$(?x\ owl{:}differentFrom\ ?y))$
$not(P(x,y))$	$(?z\ rdf{:}type\ owl{:}NegativeObjectProperty))$ $(?z\ rdf{:}subject\ ?x)$ $(?z\ rdf{:}predicate\ P)$ $(?z\ rdf{:}object\ ?z)$
$not(Q(x,y))$	$(?z\ rdf{:}type\ owl{:}NegativeDatatypeProperty))$ $(?z\ rdf{:}subject\ ?x)$ $(?z\ rdf{:}predicate\ Q)$ $(?z\ rdf{:}object\ ?z)$
$NotExist(A,B,...Z)$	$notExist(A,B,...,Z)$
$builtIn(r,z1,...,zn)$	$r(z1,....,zn)$
$dominance(Rx,Ry)$	$dominance(Rx,Ry)$
$mutex(Rx,Ry)$	$mutex(Rx,Ry)$

Table 6. Expressiveness Mapping between SWRL language and Jena rule syntax

[b] JNOMO is available at *http://sourceforge.net/projects/jnomo*

The parsing of the Jena rule syntax creates a Rete network composed of α-nodes according to the parsed token, a list data structure containing all the rules ordered by priority calculated during the parsing of the dominance operators and a list of mutex exclusions. The implementation of each of these α-nodes provides the semantics described in Table 4. Once the rule parsing has been done, the Rete network is created in order to perform the inference process. Then, each time a fact is inserted in the network, all the rules are evaluated in order to create the conflict set. Such a conflict set is composed of rule activations and retractions. Afterward, the conflict set is ordered by means of the data structures of the priorities and exclusions. Eventually, in case that the conflict set is a non-empty set, the first rule is executed/retracted. Otherwise, the next fact is processed.

Note that rule retraction requires the inverse atoms to be inserted in the Rete network with the same facts bound to the atoms that caused the activation of the retracted rule. To this end, a data structure has been implemented keeping the association of all the facts that activated a rule (called *working memory*). Then, when a rule has to be retracted, its correspondent activation facts could be retrieved from this structure to bind its variables accordingly.

JNOMO has been implemented as a plug-in for the Jena framework. The reason to implement it as plug-in instead of a stand-alone application resides in the underlying technologies used for its development. JNOMO has been developed using the Java endorsement mechanism. This technology allows overriding Java classes with other versions of these same classes. Then, by changing the order in which libraries appear in the classpath, other version of the same classes can be executed. Then, by loading the JNOMO library before the Jena ones, all the legacy systems will be able to use this new rule engine without any modification or recompilation in the code.

Note that it is not provided any kind of performance results or performance comparison in this proposal. The main reason is due to the best of our knowledge, there is not any other rule engine implementations which enable the evaluation of rules with this level of expressiveness.

10 Conclusions and Future Work

This proposal has shown the necessity of a new expressiveness extension to the SWRL language by means of a real scenario. Such an extension is aimed to define rules which could contain a $NotExists$ quantifier which enables to ask about the nonexistence of facts in the KB or remove knowledge from it. Moreover, this extension also allows for the use of the Not operator to manage negative property assertions, recently incorporated to OWL 2. Additionally, the proposed extension permits to establish priorities and exclusions relationships among rules. The semantics and inference process associated to this expressiveness extension have been also explained, describing an extension of the Rete algorithm suitable to be used in Semantic Web technologies. Finally, all the theoretical aspects have been validated by means of a free and open source implementation of the proposed inference algorithm. This implementation takes into account the semantics of the proposed operators and its performs their associated inference processes.

Regarding the possible future work, it is expected to analyze the inclusion of other types of expressiveness extensions in SWRL. For example, expressiveness extension for dealing with temporal-related or uncertainty-related operators. In relation to non-monotonicity, another expected step is a theoretical analysis aimed to extend this proposal in order to allow for abductive reasoning or reasoning by default in the inference process over SWRL rules. Another interesting line of work is the integration of the non-monotonic extensions provided in OWL language [14, 9] with the extension described in this paper. This improvement could lead to

a new SHOIQK expressiveness (K is referred to the K operator used in autoepistemic logics). Another expected future direction resides in incorporating conflict detection and resolution in the Rete algorithm. In this sense, a conflict appears when a rule causes an inconsistency in the knowledge base, thus requiring actions to restore its consistency.

Acknowledgement

Authors thank to the the Seneca Foundation for the post-doctoral grant 15714/PD/10 sponsoring Jose M. Alcaraz Calero and for the Funding Program for Research Groups of Excellence 04552/GERM/06. This paper has been also partially funded by the project RECLAMO (Virtual and Collaborative Honeynets based on Trust Management and Autonomous Systems applied to Intrusion Management) with code TIN2011-28287-C02-02 and funded by the Ministry of Science and Innovation of the Spanish Government.

11 References

1. Franz Baader. *The Description Logic Handbook: Theory, Implementation and Applications*. Cambridge University Press, 2003.
2. Sean Bechhofer. Hoolet. http://owl.man.ac.uk/hoolet/, 2005.
3. Sean Bechhofer, Frank van Harmelen, Jim Hendler, Ian Horrocks, Deborah L. McGuinness, Peter F. Patel-Schneider, Lynn Andrea Stein, and Franklin W. Olin. OWL Web Ontology Language reference. W3c recommendation, W3C, 2004.
4. Tim Berners-Lee, James Hendler, and Ora Lassila. The Semantic Web. *Scientific American*, 2001.
5. Corinna Elsenbroich, Oliver Kutz, and Ulrike Sattler. A case for abductive reasoning over ontologies. In *Workshop Proceedings of OWL: Experiences and Directions Conference*, 2006.
6. Charles L. Forgy. *Rete: a fast algorithm for the many pattern/many object pattern match problem*, chapter Rete: a fast algorithm for the many pattern/many object pattern match problem, pages 324–341. IEEE Computer Society Press, 1991.
7. Ernest Friedman-Hill. *Jess in Action*. Manning Publications Co., 2003.
8. Felix J. Garcia, Gregorio Martinez, Andres Munoz, Juan A. Botia, and Antonio F. Gomez. Towards semantic web-based management of security services. *Springer Annals of Telecommunications*, 63(3-4), 2008.
9. Stephan Grimm and Boris Motik. Closed World Reasoning in the semantic web through epistemic operators. In *Workshop Proceedings of OWL: Experiences and Directions Conference*, 2005.
10. Benjamin N. Grosof. Prioritized conflict handling for Logic Programs. In *Logic Programming: Proceedings of the 1997 International Symposium*, 1997.
11. Ian Horrocks, Peter F. Patel-Schneider, Harold Boley, Said Tabet, Benjamin Grosof, and Mike Dean. SWRL: A semantic web rule language combining OWL and RuleML. W3c member submission, W3C, May 2004.
12. U. Hustadt, B. Motik, and U. Sattler. Reasoning in description logics with a concrete domain in the framework of resolution. In *Proceeding of the 9th International Conference on Knowledge Representation and Reasoning*, 2004.
13. Minsu Jang and Joo chan Sohn. Bossam: an extended rule engine for OWL inferencing. In *Proceedings of RuleML 2004*, volume 3323, pages 128–138, Nov 2004.
14. Yarden Katz and Bijan Parsia. Towards a Nonmonotonic extension to OWL. In *Workshop Proceedings of OWL: Experiences and Directions Conference*, 2005.
15. Pavel Klinov. Pronto: A practical probabilistic description logic reasoner. In *Proceeding at International Workshop on Uncertainty*, 2010.
16. Vladimir Kolovski, Bijan Parsia, and Yarden Katz. Implementing OWL defaults. In *Workshop Proceedings of OWL: Experiences and Directions Conference*, 2006.
17. Yue Ma, Pascal Hitzler, and Zuoquan Lin. Algorithms for paraconsistent reasoning with OWL. In *Proceeding at 5th European Semantic Web Conference*, June 2008.

18. Brian McBride. Jena: Implementing the RDF model and syntax specification. In *Proceeding at Semantic Web Workshop (WWW)*, 2004.
19. Craig McKenzie, Peter Gray, and Alun Preece. Extending SWRL to express fully-quantified constraints. *Rules and rule markup languages for the semantic*, 332:139–154, 2004.
20. Boris Motik, Peter F. Patel-Schneider, and Ian Horrocks. OWL 2 Web Ontology Language: Structural specification and functional-style syntax. W3c working draft, W3C, April 2009.
21. 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, 2005.
22. Ignacio Nieto, Juan A. Botia, and Antonio F. Gomez-Skarmeta. Information and hybrid architecture model of the OCP contextual information management system. *Journal of Universal Computer Science*, 12(3):357–366, 2006.
23. Jeff Z. Pan, Giorgos Stoilos, Giorgos Stamou, Vassilis Tzouvaras, and Ian Horrocks. f-SWRL: A fuzzy extension of SWRL. In *Prooceding at International Conference on Artificial Neural Networks*, 2005.
24. Peter F. Patel-Schneider. A proposal for a SWRL extension towards First-Order Logic. Technical report, W2C, 2005.
25. Peter F. Patel-Schneider, Patrick Hayes, and Ian Horrocks. Owl web ontology language semantics and abstract syntax. W3c recommendation, W3C, http://www.w3.org/TR/owl-semantics/, 2004.
26. Alfonso Sanchez-Macian, Encarna Pastor, Jorge E. de Lopez Vergara, and David Lopez. Extending SWRL to enhance mathematical support. *Web Reasoning and Rule Systems LNCS*, 4524:358–360, 2007.
27. SCBIR. Protg. SWRL temporal builtins. http://protege.cim3.net/cgi-bin/wiki.pl?SWRLTemporalBuiltIns, Nov 2007.
28. Florian Schmedding, Nour Sawas, and Georg Lausen. Adapting the rete-algorithm to evaluate f-logic rules. *Advances in Rule Interchange and Applications*, 4824:166–173, 2007. LNCS.
29. Kunal Sengupta, Adila Krisnadhi, and Pascal Hitzler. Local closed world semantics: Grounded circumscription for owl. In *Proceesing at The 10th International Semantic Web Conference*, 2011.
30. Evren Sirin, Bijan Parsia, Bernardo C Grau, Aditya Kalyanpur, and Yarden Katz. Pellet: A practical OWL-DL reasoner. In Elsevier, editor, *Web Semantics: Science, Services and Agents on the World Wide Web*, volume 5, pages 51–53, 2007.
31. Chunming Rong Mark Musen Tomasz Wiktor Wlodarczyk, Martin O'Connor. Swrl-f - a fuzzy logic extension of the semantic web rule language. In *Proceeding at the 9th International Semantic Web Conference*, 2010.
32. Dmitry Tsarkov and Ian Horrocks. FaCT++ description logic reasoner: System description. In Springer, editor, *Proc. of the Int. Joint Conf. on Automated Reasoning*, volume 4130 of *Lecture Notes in Artificial Intelligence*, pages 292–297, 2006.
33. Ian Wright and James Marshall. The execution kernel of rc++: Rete*, a faster rete with treat as a special case. *International Journal of Intelligent Games and Simulations*, 2(1):36–48, 2003.