# Towards Personal Privacy Control

[1,2]Susana Alcalde Bagüés, [1]Andreas Zeidler, [2]Carlos Fernandez Valdivielso,
[2]Ignacio R. Matias
[susana.alcalde.ext|a.zeidler]@siemens.com
[carlos.fernandez|natxo]@unavarra.es

[1]Siemens AG, Corporate Technology
Munich, Germany
[2]Public University of Navarra
Department of Electrical and Electronic Engineering
Navarra, Spain

**Abstract.** In this paper we address the realization of personal privacy control in the era of pervasive computing. How could an individual meet his/her expected level of privacy? how could the system guaranty that a user privacy criteria is fulfilled?. For that an elaborate set of requirements for personal privacy is given followed with the implementation of our SenTry policy language.

## 1 Introduction

Privacy is a prime concern in today's information society where personal sensitive data often has to be revealed in many daily tasks. However, laws exist that shall prevent the misuse of sensitive information by enterprises once it has been disclosed. Individuals, though, often are not aware of how to control the dissemination of such data and mostly make decisions casually or on the move. Even in open settings, like the Internet, users control privacy mostly manually and are limited to acknowledging some prefabricated privacy statements. To our believes, for the upcoming era of so-called *Ambient intelligence* [1], which fosters the deployment of heterogeneous Context-Aware Mobile Services (CAMS), such control of *personal privacy* eventually will fall short. The large number of services alone will make a manual per-use authorization of access to personal data (as required by law) an impossible task.

Being different from the enterprise requirements for meeting existing legislation on privacy, personal privacy is about meeting a person's desired level of privacy of the information revealed. In this paper, we address the individual's need for managing privacy "offline"; beforehand of actually being in a particular situation. Therefore, we have elaborated requirements for Personal Privacy Control and applied them in the design of the *SenTry* language, which allows users to generate appropriate User Privacy Policies to automatically govern all accesses to their sensitive data.

The *SenTry* language is presented in this paper as part of our ongoing work focused in the development of the User-centric Privacy Framework (UCPF) [2]. The SenTry language allows the specification of fine-grained constraints on the use of personal data to conform to a user's privacy criteria. Here, the UCPF takes the roll of a trusted *privacy enforcement point* and as the *sentry* of its users' personal privacy supervises the application of policies in the interaction of the user with a CAMS.

The remainder of this document is structured as follows: First, Section 2 provides some background information on privacy policies. After that, in Section 3 requirements for implementing personal privacy are elaborated. Section 4 then details the central features of the SenTry language, which is followed by Section 5 where details of the implementation and simple use cases are presented. Finally, Section 6 indicates the directions of future work and concludes the paper.

## 2 Background

While there are many languages for access control they are rarely adequate for enforcing privacy policies [3] since they in general need of a richer expressivity to delimit accesses and usage of personal information. A policy language for supporting expression and enforcement of a privacy criteria must meet several requirements, which are different by nature if we consider the needs of an enterprise or those of an individual.

In many countries legislation regulates the collection and use of privacy data. It prevents its misuse and demands that enterprises comply with certain privacy practices (directives 95/46/EC and 2002/58/EC in Europe). Therefore, the main requirement of an enterprise from a privacy policy system is that it allows for automatic enforcement of the enterprise privacy statement. Thus, the enterprise reduce the risk of unauthorized disclosure and the risk of misuse of the collected data. A description of the enterprise privacy requirements is given in [4].

There exist a few approaches for the support of automated analysis of a privacy statement, probably the best known are the Platform for Privacy Preferences (P3P) [5], the Enterprise Privacy Authorization Language (EPAL) [6] and the eXtensible Access Control Markup Language (XACML) [7]. P3P is an standard from the World Wide Web Consortium (W3C) that enables websites to express their privacy policies and to compare them with the user's privacy criteria, which in turn can be specified by using A P3P Preference Exchange Language (APPEL) [8]. While APPEL provides a good starting point for expressing user privacy preferences, it cannot support the richness of expressions needed in ambient intelligence scenarios, see Section 3. In [9] user requirements for a privacy policy system are detailed with emphasis in the richness of constraints users might want to apply to control the distribution of their location information. Here, rules are implemented as system components called validators without defining a concrete implementation language, though. Apart from the lack of expressivity, P3P does not address the problem of enforcing a website's privacy policy [4]. The use of P3P alone does not give assurance about the *actual* privacy practices in the backend of the website and whether "obligations" implicitly included in the user privacy preferences such as the purpose or the limit time are respected.

EPAL and XACML are two platform-independent languages that support the definition and enforcement of privacy policies and obligations. IBM submitted EPAL 1.2 to the W3C in November, 2003, for consideration as a privacy policy language standard (still pending). XACML 2.0 is an XML-based language designed primarily for access control and extended for privacy. It has been accepted as an OASIS standard and widely deployed. In [3] a comparison of both languages shows that EPAL offers only a subset of the functionalities of XACML. Nevertheless, while XACML has been developed for

some time and have reached a high level of standardization, it has only started to take possible privacy constraints on information management into account. Which may be enough for enterprise privacy enforcement but it lacks of some important features to enforce personal privacy. In next Section we present those requirements that from of point of view a policy language used to express and apply personal privacy should meet.

In the following we assume that a user privacy criteria is defined as a set of rules, which are included in a User Privacy Policy, and each rule contains three logical sections namely: *applicability*, *effect* and *condition*. The applicability part contains a set of predicates used to determinate whether the rule applies to a given request. The condition section is an optional set of predicates that consist of boolean combinations of functions. If the rule satisfies a request all the predicates included in the applicability and condition section must evaluate to true. The specification of the result of enforcing a particular rule is included in the effect predicates.

## 3 Requirements for implementing Personal Privacy

A privacy policy system used to define and enforce a user's privacy criteria must provide a way of describing the different elements involve in an interaction between an individual and a CAMS, together with the environment in which such interaction could occur or no, and perform accordingly. We now outline seven requirements, which have guided the design of our policy system and particularly the SenTry language.

***Centralized privacy enforcement***. In order to offer a controlled distribution of sensitive user data we need a trusted policy system that centralizes the collection of context-related information of a user. SeTry policies are *context-aware* in the sense that their evaluation involves checking the user's context against the privacy policies available. Leaving the enforcement of such privacy policies to a third party entity would not be advisable since it would involve the disclosure of sensitive information for its evaluation. We provide then a *user privacy enforcement point* (UCPF), which controls all accesses to privacy-relevant information. Figure 1 represents the role of the UCPF in the new model of protected interaction chain, with the following entities: i) A *Target* is the tracked individual and the source of any *Resource* (location, calendar, situation, medical data, etc); ii) A CAMS compiles the resource and carries out the *Action* (purpose of the interaction); iii) A *Subject* is the user of the service and the final recipient of the data; iv) A Context Provider acts as an intermediate entity, responsible for collecting, managing and for disseminating context; v) The UCPF enforces the user private policy and provides a protected interaction with the service.

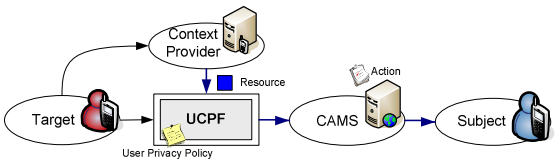



**Fig. 1.** Interaction Chain with a CAMS

***User aware***. A privacy policy system must be aware of a user needs allowing her to define fine-grained privacy policies to accomplish her expected level of privacy. It should be avoided situations in which the target has a passive role limited to accept or reject a CAMS privacy statement, as is the case of some previous approaches e.g., PawS [10]. In the interaction with the new generation of services a user should have an active role establishing how her personal relevant data is used; starting from the management and enforcement of privacy policies to a post-disclosure control of the data.

***Context awareness***. As mentioned before, SeTry policies are *context-aware*, which means that apart from those typical restrictions on the entities of Figure 1: constrains on multiple recipients (service, subject), on the purpose of the data collection (action) or on the requested resource; the SenTry language supports the inclusion of constrains on a user's context as well as on his/her peers' context to restrict when a rule applies, allowing personal and environment factors to influence e.g., whether or not she is working. Thus, for making real the idea of the UCPF roaming among various context providers a sound standard model for context representation is needed.

***Semantic awareness***. Obviously, the interaction with pervasive services and user-customized applications demands some awareness of the underlying semantics. Therefore, the policy language used for reasoning on context information has to be expressive and aware of the underlying meaning at the same time. This recommends the use of a semantic representation model for privacy policies and context data, which also will provide a common semantic frame for the collaboration of the different entities involved in the interaction chain.

***Post-disclosure awareness***. Privacy control should not be limited to a pre-disclosure phase typical of access control. Once the data has been disclosed, in order to still keep a target's privacy, is vital to have mechanisms to delimit the extension of the action granted e.g., a target may want to control the limit time of the action, or to restrict any secondary use of the data transmitted. We propose the use of *Obligations* [11] to bind an entity (service, subject) to perform a predefined action in a future on a particular object. We include *Positive Obligation Rules* to trigger a negotiation process between the UCPF and a CAMS before any resource is revealed. As a result the CAMS might agree on holding an obligation with the UCPF's user, allowing post-disclosure control.

***Transformations***. We introduce the concept of Transformations [12] to allow users to better specify their privacy preferences. Basically, we define Transformations as any process that the tracked user may define over a specific piece of context information to limit the maximum accuracy to be revealed, e.g coordinates accuracy max 500 m, or filter calendar items labeled as private. We have created the *Positive Transformation Rule* that includes a Transformation together with the grant permission, which must be performed before delivering the requested resource.

***Constraints on Active interactions***. Any interaction with a CAMS can be classified as *active* or *passive*. In the active case the user is actively using the service, some action from the service is queried, e.g. where to find the closest-by Italian restaurant. To compile an appropriate answer the CAMS asks his current location in return. The interaction is passive if the user receives a request from the CAMS without previously requesting the service e.g., if a colleague asks the Friend-finder application where a user currently is located, the disclosure of the user's location does not necessarily involve that he gets

any benefit in return. The introduction of constraints to limit a disclosure on whether or no a service request is based on an active interaction, brings much greater privacy control (an example is presented in Section 5).

## 4 *SenTry* language

The SenTry language (SeT) is built on top of the Web Ontology Language (OWL) and the Semantic Web Rules Language (SWRL) as a combination of instances of our user-centric privacy ontology (SeT ontology) and SWRL rules. The SeT ontology describes the classes and properties associated with the policy domain in OWL DL using a unique XML namespace. OWL provides considerable expressive power for modeling a domain knowledge. However, it presents some limitations, which mainly stem from the fact that is not possible to capture relationships between a certain property an another in the domain [13]. For instance, we cannot define with only OWL that if a person *hasName* "Pablo" and *hasGroup* "UPNA" the property *hasAccessMyLocation* should take the value "True". A possible way to overcome this restriction is to extend OWL with a semantic rule language. An important step has been given with the definition of the Semantic Web Rules Language, based on a combination of OWL DL and OWL Lite sublanguages with RuleML.

The SeT ontology has three main constructors namely the *Policy*, the *Service Request* and the *User Request*. They together model the elements involved in an interaction of a user with a CAMS. The Policy scheme is shown in Figure 2, a policy collects the privacy preferences of an entity identified with the property *onTarget*. In our model, the system holds a unique policy per target, which contains a collection of rules associated with the property (*hasRule*).

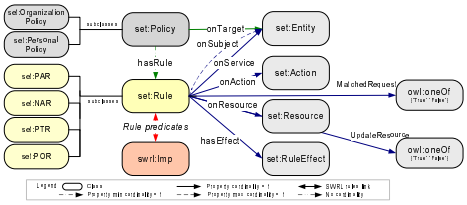


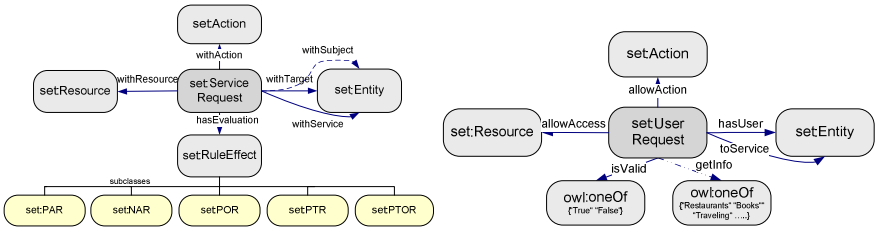

**Fig. 2.** Classes and Properties of a Policy

A policy in SeT is divided into two disjoint subclasses *PersonalPoliy* and *OrganizationPolicy*. Organizational policies covers the need of companies (organization) to manage specified context of a predefined group of individuals (employees, clients, etc). A person as part of an organization has to adhere, usually based on a contractual relationship, to some privacy policy which is orthogonal to her own. The Personal policy, scope of this paper, is mainly about how individuals control the use of personal information in everydays life.

Please note that Figure 2 depicts two different rule classes the *set:Rule* part of the SeT ontology and the *swrl:Imp*, which belongs to SWRL language. Both classes make up the three logical sections of a policy rule; The individuals of the applicability and

the effect section are created with the SeT ontology by using one out of four rule subclasses for each set:Rule, namely the Positive Authorization Rule (PAR), the Negative Authorization Rule (NAR), the Positive Transformation Rule (PTR) and the Positive Obligation Rule (POR). Each set:Rule explicitly delimits the transmission of a target's resource with the properties *onSubject*, *onService*, *onAction* and *onResource*. The result of enforcing a user's rule is specified with the property *hasEffect*. Due to the mentioned limitations on the OWL language we have included SWRL rule instances to reason about those individuals provided by the SeT ontology, primarily in terms of classes and properties. While individuals are modeled with OWL, the rule predicates of each section (applicability, effect and condition ) are expressed with SWRL.

Our second constructor is the service request, it holds the information needed to check the applicability section of a rule. The *policy decision point* [14] of the UCPF is fired to evaluate a given instance of the Service Request class. The following properties of a service request shown in Figure 3(a): *withTarget*, *withService*, *withAction* and *withResource* has a minimum cardinality for each class of one. The value given by *withTarget* selects the policy in the system, the rest of values together with the property *withSubject* are used to check whether the rules included in the selected policy match or no. The property *withSubject* has a maximum cardinality of one, we support rules defined on general aspects of a service without detailing the final recipient. A rule applies if all the values included in the service request match those given in the rule by *onSubject*, *onService*, *onAction* and *onResource*. If the applicability predicates (SWRL rules) evaluate to true then the condition section finally determines whether the rule effect affects to the given request.




(a) Service Request (b) User Request

**Fig. 3.** Classes and Properties

Since a policy contains multiple rules and since rules may evaluate to different results given the same service request e.g., the PAR returns "Grant", the NAR "Deny", the PTR "Grant with transformation" and the POR "Grant with obligation". The system must determinate potential rule conflicts, see Section 5, before determinating the result to be returned from the policy evaluation. The final result is then assigned to the request thought its property *hasRuleEvaluation*, which may include an extra value the PTOR: a grant permission that contains a transformation and obligations.

The last constructor used in SeT is the User Request. This element is used to model *active interactions* between a user and a third-party service. A user identified with the property *hasUser* actively can request a service (*toService*) and compromise to allow the action defined with *allowAction* on the resource contained in the propierty *allowRe-*

*source*, which is needed to compile the requested service. We explicitly distinguish between rules that affect an active interaction with a service by evaluating an instance of the class User Request against an individual of the Service Request.

## 5 Implementing a SenTry Policy

Our PDP component has been developed in JAVA on top of the Java Expert System Shell (Jess). Once the PDP gets a evaluation request message, triggers the execution of the Jess rule engine, which then accesses the policy repository to retrieve the policies applicable for the current situation, which are evaluated based on the provided Service Request instance. Our policy repository is integrated by all the different individuals created on the SeT ontology and the SWRL rule predicates. Along with rules in the Jess language, the Jess engine accepts rules formulated in SWRL [15], which are translated into Jess rules and facts with the Java API SWRL factory.

Each time that a user defines a new rule, a new instance of the set:Rule is created together with the needed SWRL predicates. In our system rule predicates are first expressed with SWRL and then automatically translated to Jess. The SWRL rules are divided in three groups, those that are make up with *Filter predicates*, with *Static predicates* or with *Dynamic predicates*. In common with many other rule languages, SWRL rules are written as antecedent-consequent pairs. In SWRL terminology, the antecedent is referred to as the rule body and the consequent is referred to as the head. The head and body consist of a conjunction of one or more atoms (rule predicates). At present, SWRL does not support more complex logical combinations of atoms that the conjunction, represented with the symbol $\wedge$ in the examples presented below.

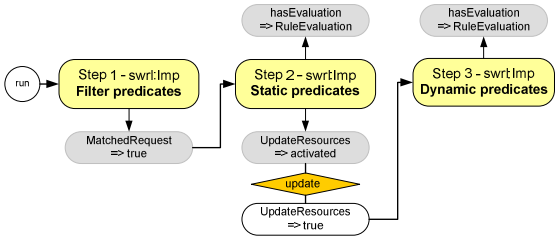



**Fig. 4.** Policy Evaluation Process

The SWRL predicates included on a user's rule are enforced consecutively in three steps as is shown in Figure 4. In the step 1 the Filter predicates of Figure 5(b) are evaluated. They are a set of common conditions to all the user's rules, there is only a SWRL rule- step 1 shared by all the instances of the set:Rule class. If all the Filter predicates are true for a given set:Rule, the *MatchedRequest* property is set to "true" and the SWRL rule- step 2, which includes the Static predicates associated to that particular set:Rule is triggered. The Static predicates might include conditions base on multiple recipient and/or the active interaction constraint.

The evaluation of the Static predicates leads (if true) to two possible results: i) there is not Dynamic predicates associated with the matched set:Rule. Thus, the SWRL rule-

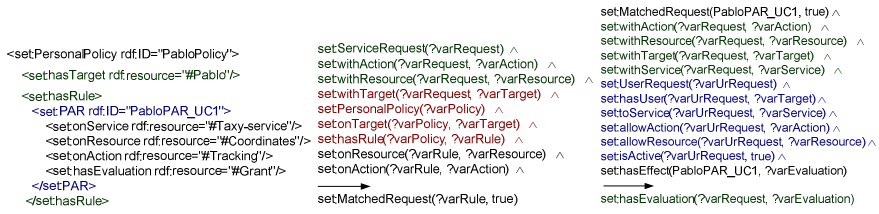
step 2 asserts the appropriate instance of the *RuleEvaluation*; ii) there exist Dynamic predicates defined on the set:Rule and the evaluation of the Static predicates has the effect of setting the *UpdateResources* property of the set:Rule instance from “false” to “activated”. Once the second step is finished an internal process checks if there are any instance of a set:Rule with the property *UpdateResources* equal to “activated” and updates the list of dynamic resources from the respective context providers. Finally, it sets the *UpdateResources* to “true”. The step number 3 of the evaluation process decides about Dynamic predicates. They might include time constraints, constraints on the target’s context, or/and on a target peer’s context. As consequent it returns a *RuleEvaluation*. The last part of this process consist on applying a combining algorithm on the *RuleEvaluation* instances returned in the process described. It combines rule effects generated in the second and third step. By default, the system generates NAR rules, only when a user explicitly define a PAR is possible to permit a requested action. We use a *grant overrides* combining algorithm to resolve conflicts between rules, together with inheriting transformations and obligation from PTR and POR rule effects.

### 5.1 Examples

We now introduce two simple use cases that show how a SenTry policy is implemented. The use case 1 includes only two SWRL rules (step 1 and 2), while the use case 2 has also Dynamic predicates (step 3). The step 1, Figure 5(b), is common to both use cases. It filters the policy with target “Pablo” and the rules within that policy, which specify the same resource and action that the given instance of the service request.

Use case 1: *Pablo uses his mobile phone to call a taxi at the end of day, he wants the taxi company to determine his location automatically to ensure a smooth pickup, but he does not want them to be able to trace him once the journey is over.*

In the above use case the target, Pablo, allows being tracked during a limited time slot. While the request for the taxi service is still *active*. Thus, we have included in the rule- step 2 (Figure 5(c)) a unique constraint: the active interaction constraint. This rule checks if there exist an active User Request instance with the flag *isActive* equal to true, for a given Service Request and in that case returns a grant permission, defined in the owl rule 5(a).

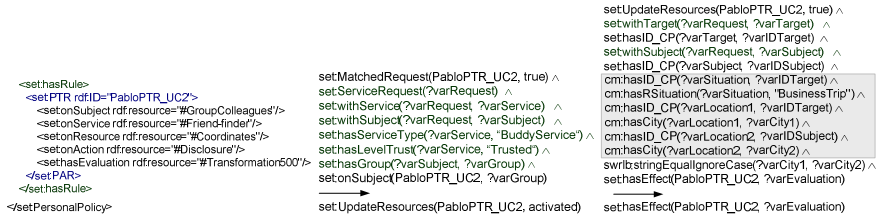



(a) set:Policy (b) SWRL rule (step 1) (c) SWRL rule (step 2)

**Fig. 5.** Pablo’s Rule - Use Case 1

Use case 2: *When Pablo is on a business trip, he likes to meet old colleagues that may coincide in the city. He allows to be located by the peer group Colleagues, but only if the potential subject is in the same city and with a maximum accuracy of 500 m.*

The use case 2 requires constraints based on multiple subjects and dynamic constraints. The rule- step 2, Figure 6(b), checks the type of service just as the group of the subject. Here, Pablo allows the disclosure of his coordinates with a maximum accuracy of 500m (PTR - transformation 500) to the group colleagues. The rule- step 3 applies constraints based on the target situation (Business trip) and limits the disclosure to circumstances in which target and subject are in the same city, Figure 6(c). This rule includes predicates expressed over individuals of the context model ontology (cm), situation and coordinates. We assume an standard context model, which is linked with the SeT ontology throught the property *hasID-CP* per person and per context provider.




(a) set:Policy (b) SWRL rule (step 2) (c) SWRL rule (step 3)

**Fig. 6.** Pablo's Rule - Use Case 2

## 6 Conclusions and Outlook

The paper presents a set of user requirements based ... Obligations–¿ Future work, negotiation protocol Only Personal Policy -¿ integration of organizational privacy and priorization User Interface: We are currently implementing the first prototype of the SenTry user interface to allow users to define policies and rules in a natural way. Application in the IST CONNECT project as part of the CoPO ontology

## References


1. IST Advisory Group (ISTAG). Ambient Intelligence: From Vision to Reality. For participation in society and business. Luxembourg: Office for Official Publications of the European Communities, September 2003.
2. Susana Alcalde Bagüés, Andreas Zeidler, Carlos Fernandez Valdivielso, and Ignacio R. Matias. Sentry@home - leveraging the smart home for privacy in pervasive computing. *To appear in the International Journal of Smart Home*, 2007.
3. Anne Anderson. A Comparison of Two Privacy Policy Languages: EPAL and XACML. Technical report, Sun Microsystems Laboratories, Technical Report TR-2005-147, September 2005.
4. Marnix Dekker, Sandro Etalle, and Jerry den Hartog. *Security, Privacy and Trust in Modern Data Management. Chapter 25: Privacy Policies*. Springer-Verlag, 2007.
5. Lorrie Cranor, Marc Langheinrich, Massimo Marchiori, and Joseph Reagle. The platform for privacy preferences 1.0 (P3P1.0) specification. W3C Recommendation, April 2002.

6. P. Ashley, S. Hada, G. Karjoth, C. Powers and M. Schunter. Enterprise Privacy Authorization Language (EPAL 1.2) Specification `http://www.zurich.ibm.com/security/enterprise-privacy/epal/`, November 2003.
7. OASIS standard. eXtensible Access Control Markup Language. Version 2, February 2005.
8. Marc Langheinrich, Lorrie Cranor, and Massimo Marchiori. Appel: A P3P Preference Exchange Language. W3C Working Draft, April 2002.
9. Ginger Myles, Adrian Friday, and Nigel Davies. Preserving privacy in environments with location-based applications. *IEEE Pervasive Computing*, 2(1):56–64, 2003.
10. Marc Langheinrich. A privacy awareness system for ubiquitous computing environments. In *Proceedings of the 4th International Conference on Ubiquitous Computing*, pages 237–245. LNCS No. 2498, Springer-Verlag, September 2002.
11. Lalana Kagal and Tim Finin. Modeling Conversation Policies using Permissions and Obligations. *Journal of Autonomous Agents and Multi-Agent Systems*, December 2006.
12. Susana Alcalde Bagüés, Andreas Zeidler, Carlos Fernandez Valdivielso, and Ignacio R. Matias. A user-centric privacy framework for pervasive environments. In *OTM Workshops (2)*, pages 1347–1356. LNCS, 2006.
13. Ian Horrocks, Peter F. Patel-Schneider, Sean Bechhofer, and Dmitry Tsarkov. OWL rules: A proposal and prototype implementation. *J. of Web Semantics*, 3(1):23–40, 2005.
14. R. Yavatkar, D. Pendarakis, and R. Guerin. RFC2753 - A framework for policy-based admission control, January 2000.
15. Martin O'Connor et all. Supporting Rule System Interoperability on the Semantic Web with SWRL. In *In Proceedings of the 4th International Conference on The Semantic Web ISWC 2005*. LNCS 3729, 2005.