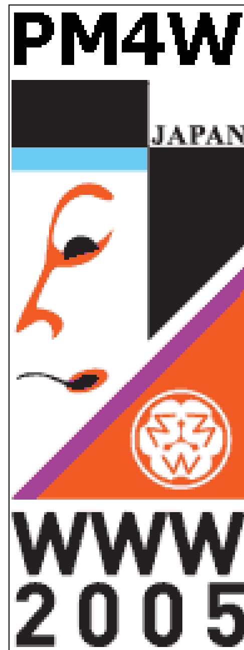
Policy Management for the Web

A workshop held at the
14th International World Wide Web Conference
Tuesday 10 May 2005, Chiba Japan



Lalana Kagal, Tim Finin, and Jim Hendler (Eds.)

Contents

Forward 2

Workshop schedule 3

Call for papers 4

Research papers

Policy Conflict Analysis Using Free Variable Tableaux for Access Control in Web Services Environments, Hiroaki Kamoda, Masaki Yamaoka, Shigeyuki Matsuda, Krysia Broda, and Morris Sloman 5

Methods for Policy Conflict Detection and Resolution in Pervasive Computing Environments, Evi Syukur, Seng Wai Loke, and Peter Stanski 13

Policy Conformance in the Corporate Blog Space, Robert McArthur, Peter Bruza, and Dawei Song 21

Expressing WS Policies in OWL, Bijan Parsia, Vladimir Kolovski, and Jim Hendler 29

Policy-based Access Control for Task Computing Using Rei, Ryusuke Masuoka, Mohinder Chorpa, Zhexuan Song, Yannis Labrou, Lalana Kagal, and Tim Finin 37

Describing the P3P base data schema using OWL, Giles Hogben 44

Position papers

Predicates for Boolean web service policy languages, Anne Anderson 52

Policy Management and Web Services, Greg Pavlik, Tim Gleason, and Kevin Minder 57

Representing Security Policies in Web Information Systems, Felix J. Garcia Clemente, Gregorio Martinez Perez, Juan A. Botia Blaya , and Antonio F. Gomez Skarmeta 61

RDF Query for Policy Management, Eric Prud`hommeaux 67

Application Report: An extensible policy editing API for privacy and identity management policies, Giles Hogben 72

Policy based access control for an RDF store, Pavan Reddivari, Tim Finin, and Anupam Joshi 78

Forward

This workshop brings together researchers interested in the role of explicit, machine interpretable policies to control programs, services and agents on the Web. We believe that such policies will have a role to play in realizing the full potential of the Web as an open, dynamic, and distributed ``universe of network-accessible information''. Policy management provides the openness, flexibility, and autonomy required to regulate this environment as entities can reason over their own policies and the policies of other entities to decide how to behave. Using policies also allows entities to specify expected behavior of entities they interact with. Entities can also adapt to increasingly complex requirements without the need for substantial changes to the structure or implementation through the use of policies. Policy management includes policy specification, deployment, and reasoning over policies, updating and maintaining policies, and enforcement.

The workshop could not have happened without the participation of the program committee. They reviewed the submitted papers, selected those for including in the proceedings and presentation at the workshop, and provided the authors with helpful advice and comments. We thank the committee for their generous contributions.

The program committee included Anne Anderson (Sun Microsystems), Vijay Atluri (Rutgers University), Elisa Bertino (Purdue University), Jeffrey M. Bradshaw (Institute for Human and Machine Cognition), Dan Connolly (World Wide Web Consortium), Naranker Dulay (Imperial College), Tim Finin (University of Maryland Baltimore County), Jim Hendler (University of Maryland College Park), Maryann Hondo (IBM), Benjamin Grosof (Massachusetts Institute of Technology), Anupam Joshi (University of Maryland Baltimore County), Lalana Kagal (Massachusetts Institute of Technology), Jonathan Moffett (University of York), Wolfgang Nejdl (L3S Research Center and University of Hannover), Bijan Parsia (University of Maryland College Park), Filip Perich (Cougaar Software), Stefan Poslad (Queen Mary University of London), Eric Prud'hommeaux (World Wide Web Consortium), Norman Sadeh (Carnegie Mellon University), Kent Seamons (Brigham Young University), Marek Sergot (Imperial College), Akhil Sahai (Hewlett Packard Laboratories), Katia Sycara (Carnegie Mellon University), Dinesh Verma (IBM TJ Watson Research Center), William Winsborough (George Mason University), and Marianne Winslett (University of Illinois, Urbana-Champaign).

Workshop schedule

9:00 Welcome, Jim Hendler

9:15 Transparency vs. Privacy, Daniel Weitzner (W3C)

10:30 Break

11:00 Session One : Conflicts and Conformance

- *Policy Conflict Analysis Using Free Variable Tableaux for Access Control in Web Services Environments* (20 min), Hiroaki Kamoda, Masaki Yamaoka, Shigeyuki Matsuda, Krysia Broda, and Morris Sloman
- *Methods for Policy Conflict Detection and Resolution in Pervasive Computing Environments* 20 min), Evi Syukur, Seng Wai Loke, Peter Stanski
- *Policy Conformance in the Corporate Blog Space* (20 min), Robert McArthur, Peter Bruza, and Dawei Song
- Discussion (30 min)

12:30 Lunch

14:00 Session Two : Web Services and Policy Management

- *Expressing WS Policies in OWL* (20 min), Bijan Parsia, Vladimir Kolovski, and Jim Hendler
- *Predicates for Boolean web service policy languages* (10 min), Anne Anderson
- *Policy-based Access Control for Task Computing Using Rei* (20 min), Ryusuke Masuoka, Mohinder Chorpa, Zhexuan Song, Yannis Labrou, Lalana Kagal, and Tim Finin
- *Describing the P3P base data schema using OWL Services* (10 min), Greg Pavlik, Tim Gleason, and Kevin Minder
- Discussion (30 min)

15:30 Break

16:00 Session Three : Policy Representation

- *Representing Security Policies in Web Information Systems*, (10 min), Felix Garcia Clemente, Gregorio Martinez Perez, Juan Botia Blaya , and Antonio Gomez Skarmeta
- *Describing the P3P base data schema using OWL* (20 min), Giles Hogben
- *RDF Query Requirements for Policy Management* (20 min), Eric Prud`hommeaux
- *Application Report: An extensible policy editing API for privacy and identity management policies* (20 min), Giles Hogben
- Policy based access control for an RDF store (10 min), Pavan Reddivari, Tim Finin, and Anupam Joshi
- Discussion (25 min)

17:30 Discussion for a white paper to sum up results of the workshop, closing remarks

Policy Management for the Web

A Workshop to be held at the 14th International World Wide Web Conference Tuesday 10 May 2005, Chiba Japan

In order to realize the full potential of the World Wide Web as an open, dynamic, and distributed ``universe of network-accessible information'', it is important for web entities to behave appropriately. Policy management provides the openness, flexibility, and autonomy required to regulate this environment as entities can reason over their own policies and the policies of other entities to decide how to behave. Using policies also allows entities to specify expected behavior of entities they interact with. Entities can also adapt to increasingly complex requirements without the need for substantial changes to the structure or implementation through the use of policies. Policy management includes policy specification, deployment, reasoning over policies, updating and maintaining policies, and enforcement. We propose that policy management is required for the web for (i) constraining different kinds of behavior including security, privacy, conversation, and collaboration, (ii) configuration management, (iii) describing business processes, and (iv) establishing trust and reputation. Relevant topics include the following:

- Policy specification, implementation, and enforcement
- Dynamic merging of policies
- Static and dynamic conflict resolution
- Dynamic policy modification
- Formal models for policy verification
- Relationship of trust and reputation to policies
- Business contracts and rules
- Case studies for policy management
- Applicability of XML, RDF and OWL for policy specification
- Obligation management
- Policies for access control, privacy, and collaboration
- Decidability and tractability issues
- Digital Rights Management policies
- Policy engineering
- Enhancing P3P with policies
- User-oriented policy authoring systems

Format and venue. PM4W will be a one day workshop consisting of invited talk(s), presentations of submitted papers, and (probably) a panel as well as time for discussion. The workshop will be held as part of WWW2005 in Chiba, Japan at Nippon Convention Center (or better known as Makuhari Messe). Makuhari Messe is conveniently located halfway between central Tokyo and the New Tokyo International Airport (Narita Airport).

Submission details. We seek two kinds of papers: research papers that report on the results of original research and short papers that articulate a position, describe an application or demonstrate a working language or system. Both research papers and short papers will be included in the workshop proceedings. Research papers should describe original research not published elsewhere and should not exceed eight pages in length. Short papers are expected to be four to six pages. Short position papers should provide insight into the requirements for, or challenges of, developing or applying policies for web-based information systems. Short application papers should describe an implemented novel use of policies in a web-based environment. Short demonstration papers should document a implemented system or language that uses policies. Each submission should indicate the type of paper being submitted: research, position, application or demonstration. See the web site for additional information on format requirements.

Deadlines. Papers must be submitted electronically by 1 February 2005. Decisions will be announced on 15 March and final camera ready copy must be submitted by 15 April.

http://cs.umbc.edu/pm4w/

Chairs

Tim Finin, UMBC
Jim Hendler, UMCP
Lalana Kagal, MIT

Program Committee

Anne Anderson, Sun
Vijay Atluri, Rutgers University
Elisa Bertino, Purdue University
Jeffrey M. Bradshaw, IHMC
Dan Connolly, W3C
Naranker Dulay, Imperial College
Tim Finin, UMBC
Benjamin Grosof, MIT Sloan
Jim Hendler, UMCP
Maryann Hondo, IBM
Anupam Joshi, UMBC
Lalana Kagal, MIT CSAIL
Jonathan Moffett, University of York
Wolfgang Nejdl, L3S , U. Hannover
Bijan Parsia, UMCP
Filip Perich, Cougaar Software
Stefan Poslad, Queen Mary University of London
Eric Prud'hommeaux, W3C
Norman Sadeh, CMU
Kent Seamons, BYU
Marek Sergot, Imperial College
Akhil Sahai, HP labs
Katia Sycara, CMU
Dinesh Verman, IBM TJ Watson
William Winsborough, GMU
Marianne Winslett, UIUC

Invited Talks

Daniel Weitzner of the W3C will give an invited talk on transparency and policy.

Policy Conflict Analysis Using Free Variable Tableaux for Access Control in Web Services Environments

Hiroaki Kamoda
NTT DATA CORPORATION
Tokyo, Japan
kamodah@nttdata.co.jp

Masaki Yamaoka
NTT DATA CORPORATION
Tokyo, Japan
yamaokam@nttdata.co.jp

Shigeyuki Matsuda
NTT DATA CORPORATION
Tokyo, Japan
matsudasg@nttdata.co.jp

Krysia Broda
Imperial College London, UK
k.broda@imperial.ac.uk

Morris Sloman
Imperial College London, UK
m.sloman@imperial.ac.uk

ABSTRACT

Web Services technologies are now an active research area. By integrating individual existing web systems the technology enables the provision of advanced and sophisticated services, such as allowing users to use different types of resources and services simultaneously in a simple procedure. However the management and maintenance of a large number of Web Services is not easy and, in particular, needs appropriate authorization policies to be defined so as to realize reliable and secure Web Services. The required authorization policies can be quite complex, resulting in unintended conflicts, which could result in information leaks or prevent access to information needed. This paper proposes an approach using free variable tableaux for detecting conflicts resulting from the combination of various kinds of authorization and constraint policies used in Web Services environments. The method not only enables static detection of policy conflicts such as modality and static constraint conflicts but also yields information that is helpful for correcting the policies.

Categories and Subject Descriptors

D.2.4 [**Software Engineering**]: Software/Program Verification—*Formal methods, Validation*; D.4.6 [**Operating Systems**]: Security and Protection—*Access controls, Verification*

General Terms

Algorithms, Theory, Verification

Keywords

Access Control, Policy Analysis, Conflict Detection, Free Variable Tableaux, Abduction

1. INTRODUCTION

The recent spread of broadband technology such as DSL and FTTH has led to a rapid increase in the number of Internet users across the world. One of the key technologies is the use of Web systems, often based on the use of HTTP, and although having been in use for many years, it is still one of the most used technologies. In particular, ways of integrating individual web systems to provide advanced services have been suggested (e.g. [21, 27]). Web Services are constructed by statically or dynamically integrating independent web systems using a set of XML standards such as SOAP[26], Universal Description, Discovery and Integration (UDDI)[24] and Web Services Description Language (WSDL)[25]. This enables advanced and sophisticated services to be provided enabling users to perform several procedures simultaneously, resulting in a better overall service.

Copyright is held by the author/owner(s).
WWW2005, May 10–14, 2005, Chiba, Japan.
.

In order to realize reliable and secure Web Services it is important to authenticate and authorize the users appropriately. For instance, to prevent problems such as an information leak, suitable access control is needed for the users who access the resources through Web Services. By using the standard policy description languages such as WS-Policy[5], WSPL[1] and XACML[17], it is possible to realize complicated access control for Web Services. However, the overall structure of these policies can become very complex, reflecting the complexity of the web services and roles involved. There is an increased risk that an administrator mistakenly defines conflicting policies which, if the wrong choice is made, result in information leak or prevent access to critical information in an emergency situation.

We have already proposed a static method for detecting policy conflicts arising in the On Demand VPN Framework[14]. The method is based on free variable tableaux and has the advantage that it gives helpful information for resolving conflicts. In this paper we extend the method beyond simple authorization policies to cope with various kinds of constraints on policies.

The paper is organized as follows: Section 2 introduces the Web Services model for policy analysis, Section 3 presents an outline of conflict detection using free variable tableaux and in Section 4 we illustrate the method to detect and abduce conflicting policies through examples. In Section 5 we describe some related work, and our conclusions and future work are presented in Section 6.

2. WEB SERVICES MODEL AND POLICY

There are many types of use case models for Web Services[11] and in this paper, we assume the “aggregation Web Services model”, in which a single server manages several Web Services accessed by multiple users. This model is mainly used for services such as portal site, market place and one stop services. The features of the model and policies used in it are described in this section. The particular Web Services model used in this paper is shown in Fig.1.

2.1 Web Services Model

The main entities of the Web Services model used here are re-

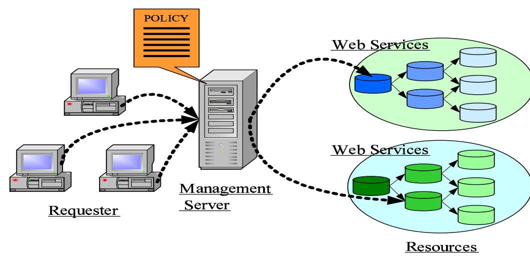


Figure 1: Aggregation Web Services Model

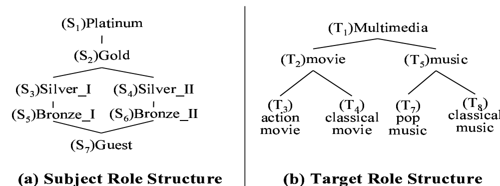


Figure 2: Examples of Role Structures

quester, management server, Web Services and their resources. A management server integrates several Web Services and provides a common services interface for users. A requester sends a request to the management server to use the resources or services provided by the Web Services. A management server checks the request by using the access control policy to see whether it should be granted or not. If it is granted, then the request is transferred to appropriate Web Services to answer the request. The most popular use case of this model is travel agency service example[11]. By using the management server, there is an advantage that requesters can use any Web Services in a similar way.

We assume the authorization policies needed for checking the request are defined in terms of subject and target role structures[4, 18]. Policies can propagate up or down the role structures. Furthermore, an authorization policy may be defined in terms of composite actions, which can result in conflicts if separate policies are defined for the various sub-actions. We also assume that we can define obligation policy and kinds of constraint policy, including the Chinese wall policy, separation of duty policy and time constraint policies. These policies are all explained below.

2.2 Features of the Policy

In this section policies that can be defined in the management server are presented.

2.2.1 Roles

Policies are defined by using a *role*, which is a named collection of privileges[9]. A partial order relation is defined among these roles and the graph representation of the relation is called a *role structure*. Individual requesters and resources take on assigned roles. In particular, the role structure corresponding to requesters is called a *subject role structure* (SRS) and that corresponding to resources or services of Web Services is called a *target role structure* (TRS). Examples of these role structures are shown in Fig.2.

2.2.2 Authorization Policy

The most basic policy defined in the management server is an *authorization policy*. There are both positive and negative authorization policies. Examples are:

$$\text{Policy r1}: \quad \texttt{Auth}+(\texttt{Bronze_I}, \texttt{movie}, \texttt{play})$$
$$\text{Policy r2}: \quad \texttt{Auth}-(\texttt{Gold}, \texttt{movie}, \texttt{play})$$

These policies define authorizations between a requester and Web Services that provide multimedia contents. Policy r1 specifies that the subject role `Bronze_I` is allowed to perform the action `play` on the target role `movie` and Policy r2 specifies that the subject role `Gold` is forbidden to perform the action `play` on the target role `movie`. The policies r1 and r2, appear to define authorizations for different subject roles so there should be no problems. However, if these policies are compared with respect to the role structure, then a conflict occurs, which is explained in the next section.

2.2.3 Propagation Policy

The role structures potentially simplify policy specification by allowing *propagation policies*. In general, if a certain subject role r is allowed to perform a particular action, then roles higher than r should also be allowed to perform the action. Conversely, if roles higher than r are not permitted to perform an action, then r should not be permitted to perform the action. These propagation policies are specified as follows:

$$\text{Policy r3}: \quad \texttt{prop}(\texttt{Auth}+, \mathcal{R} \in \texttt{SRS}, \texttt{Up})$$
$$\text{Policy r4}: \quad \texttt{prop}(\texttt{Auth}-, \mathcal{R} \in \texttt{SRS}, \texttt{Down})$$

Policy r3 specifies that `Auth+` policy defined for subject role structure $\mathcal{R}$ propagates upwards through roles. Policy r4 specifies that `Auth-` policy defined for subject role structure $\mathcal{R}$ propagates downward through roles. More concretely, let the role structure shown in Fig.2(a) be $\mathcal{R}$, then Policies r3 and r4 implicitly define the following policies from Policies r1 and r2.

$$\begin{array}{lll} \text{r1_1} & : & \texttt{Auth}+(\texttt{Silver_I}, \texttt{movie}, \texttt{play}) \\ \text{r1_2} & : & \texttt{Auth}+(\texttt{Gold}, \texttt{movie}, \texttt{play}) \\ \text{r1_3} & : & \texttt{Auth}+(\texttt{Platinum}, \texttt{movie}, \texttt{play}) \\ \text{r2_1} & : & \texttt{Auth}-(\texttt{Silver_I}, \texttt{movie}, \texttt{play}) \\ \text{r2_2} & : & \texttt{Auth}-(\texttt{Silver_II}, \texttt{movie}, \texttt{play}) \\ \text{r2_3} & : & \texttt{Auth}-(\texttt{Bronze_I}, \texttt{movie}, \texttt{play}) \\ \text{r2_4} & : & \texttt{Auth}-(\texttt{Bronze_II}, \texttt{movie}, \texttt{play}) \\ \text{r2_5} & : & \texttt{Auth}-(\texttt{Guest}, \texttt{movie}, \texttt{play}) \end{array}$$

Clearly, Policy r1_2 and Policy 2_3 derived from propagation policies r3 and r4 respectively conflict with Policy r2 and Policy r1, since the subject roles named `Bronze_I` and `Gold` have opposite permissions. Policy r1_1 and Policy r2_1 also conflict.

Propagation is a convenient and easy way to specify implicit policies, but it can result in unforeseen conflicts. Note that the concept of the role structure described here is slightly different from the role hierarchies defined in the standard role based access control model[9] in that the propagation is explicitly defined by a propagation policy, rather than being implicit. The direction of the propagation may differ according to the type of policy or the type of role structures. For example, the system administrator may define the subject role structure "upside down" in some situations. For example, in Fig.2 the administrator may define the Guest user as a top and Platinum user as a bottom role, in which case polices should propagate in the opposite directions to those given in Policies r3 and r4. That is, Policy r3 would specify `Down` and Policy r4 would specify `Up`. There also may be a case that only lower role users are permitted to do something. For example we can imagine the situation in which the rank of member status is decided by how many points the member has purchased. In this case members with a role lower than Gold should be permitted to access the service to purchase the points and a positive authorization would be expected to propagate down. We can thus define an explicit propagation policy for each role structure which is more flexible than the implicit propagation in standard role hierarchies.

2.2.4 *Action Composition Policy*

Policies may be defined in terms of more than one action. For example, consider a reservation system, for which the Web Services may provide different types of reservation services. Example policies are:

Policy r5 : $\texttt{Auth}+(\texttt{Bronze_II}, \texttt{TR}, \texttt{rsv_travel})$
Policy r6 : $\texttt{Auth}-(\texttt{Bronze_II}, \texttt{TR}, \texttt{rsv_air})$
Policy r7 : $\texttt{Auth}-(\texttt{Bronze_II}, \texttt{TR}, \texttt{rsv_hotel})$

`rsv_travel`, `rsv_air` and `rsv_hotel` mean, respectively, to send a request for some holiday abroad, to reserve an airline ticket and to reserve a hotel, and `TR` indicates a certain Web Service that provides travel reservation services. At first sight, comparing the three policies r5, r6 and r7, no problems are detected. However, `rsv_travel` is, in fact, a composite action, defined as the following *action composition policy*:

Policy r8 : $\texttt{rsv_travel} = \texttt{rsv_air} \wedge \texttt{rsv_hotel}$

This specifies that two actions `rsv_air` and `rsv_hotel` are needed to complete the request `rsv_travel`. This means that to perform an overseas holiday reservation process the requester must be granted to reserve both an airline ticket and hotel accommodation through the Web Services. Then r5, r6 and r7 become conflicting policies, as policy r5 specifies that `Bronze_II` is allowed to perform an `rsv_travel` action, while the other two policies specify that both the actions `rsv_air` and `rsv_hotel` are prohibited. In this way an action composition may also lead to policy conflicts.

2.2.5 *Obligation Policy*

In addition to the authorization policy described in Section 2.2.2, an obligation policy[8] can be defined. Example policies are:

Policy r9 : $\texttt{Obli}+(\texttt{Play}, \texttt{Guest}, \texttt{fillout}, \texttt{questionnaire})$
Policy r10 : $\texttt{Obli}-(\texttt{Sunday}, \texttt{Guest}, \texttt{login}, \texttt{WS})$

Policy r9 specifies that `Guest` member must fill out the questionnaire after playing the multimedia contents. Policy r10 specifies that `Guest` member must not login to the Web Services named `WS` on Sunday.

2.2.6 *Chinese Wall and Separation of Duty Policy*

A *Chinese wall policy*[6] and *separation of duty policy*[7] defines the constraints for target roles and actions respectively. Note that the original use for the separation of duty policy was to prevent an occurrence of fraud; however, in this paper separation of duty simply means a constraint for any actions. Here we consider examples such as online banking and auction Web Services, for which example policies are:

Policy r11 : $\texttt{CW}(\texttt{Guest}, \{\texttt{Bank_A}, \texttt{Bank_B}\}, \texttt{view_account})$
Policy r12 : $\texttt{SoD}(\texttt{Bronze_I}, \texttt{Auction}, \{\texttt{sell}, \texttt{buy}\})$

Policy r11 specifies that subject role named `Guest` is permitted to view accounts of exactly one of the target roles `Bank_A` or `Bank_B`. Policy r12 specifies that subject role named `Bronze_I` of auction Web Services is permitted to either sell or to buy something through the Web Services, but not both buy and sell simultaneously.

These constraint policies may also lead to other types of policy conflict. For example, the following two positive authorization policies r13 and r14 conflict with Policy r11.

Policy r13 : $\texttt{Auth}+(\texttt{Guest}, \texttt{Bank_A}, \texttt{view_account})$
Policy r14 : $\texttt{Auth}+(\texttt{Guest}, \texttt{Bank_B}, \texttt{view_account})$

The conflict arises because these policies allow the subject role named `Guest` to view accounts of both target roles named `Bank_A` and `Bank_B`. A similar situation can be happen when defining a separation of duty policy.

2.2.7 *Time Constraint Policy*

A time constraint policy can be used to specify the period during which an authorization policy is valid. This constraint is defined in each authorization policy. Here is a multimedia Web Services example:

Policy r15 : $\texttt{Auth}+(\texttt{Gold}, \texttt{movie}, \texttt{play}, [\texttt{00:00}, \texttt{24:00}])$
Policy r16 : $\texttt{Auth}-(\texttt{Guest}, \texttt{music}, \texttt{play}, [\texttt{09:00}, \texttt{17:00}])$

Policy r15 specifies that subject role named `Gold` can play a movie for 24 hours (*i.e.* at any time). Policy r16 specifies that subject role named `Guest` cannot play music between 9:00 to 17:00. A time constraint policy itself doesn't cause a policy conflict. Policy conflicts can happen only if the time periods specified in various policies overlap. An example is given in subsection 4.3.3.

2.3 Policy Conflict

As described in Section 2.2, conflicting policies can result from propagation, action composition and other constraint policies, which cannot be detected by simply comparing authorization policies. We call this type of conflict *implicit conflict*. The problem is that as role structures and the action compositions become more complex, so it becomes more difficult to detect an implicit conflict. In some applications runtime conflict detection methods are not suitable. For example, the information exchanged in medical applications usually contains very sensitive data. Information leak caused by an incorrect policy should never be allowed and contrarily in a medical emergency prevention of access to information resulting from an undetected conflict could have life-threatening consequences. Therefore we need a method that can analyze policies statically before activating a system, in order to detect presence of conflicts, and to provide information to resolve any conflicts detected. In the rest of this paper we present our approach, which is based on free variable tableaux, to satisfy these demands.

3. FREE VARIABLE TABLEAUX

In this section we describe an outline of the conflict detection method based on *free variable tableaux*[10].

It is possible to enumerate all policies derived implicitly by propagation and action composition policies and then to detect an implicit conflict by comparing the original and derived policies. However, this would be computationally expensive and it is still hard to identify the original policies that cause any conflict. The Free Variable Tableaux method allows faster detection of a conflict and also infers the cause of the conflict.

Detection of a conflict effectively requires that a contradiction $\perp$ be derived from a collection of policies $\mathcal{P}$. To prove that C results from Γ (*i.e.* $\Gamma \models C$) is equivalent to showing that the set $\{\Gamma, \neg C\}$ is inconsistent (*i.e.* $\{\Gamma, \neg C\} \models \perp$). The method of free variable tableaux (FVT) can be used to show inconsistency. The FVT method is a sound and complete theorem prover upon which can be built simple abductive reasoning. Moreover, it has optimized implementations. The following two steps are needed to detect a conflict using FVT:

i) each policy is translated into a logical sentence

ii) the FVT method is applied to these sentences to detect any possible conflicts, by detecting inconsistency, and to obtain the information that shows the cause of the conflict.

In other words, all we have to do is to define the following translation mapping ζ from policies to logical sentences, such that conflicting policies become inconsistent sentences in logic.

$$\begin{array}{rccc} \zeta: & \mathcal{P} & \to & \mathcal{L} \\ & \Psi & & \Psi \\ & r & \mapsto & \zeta(r) \end{array}$$

where $\mathcal{P}$ is a set of policies and $\mathcal{L}$ is a set of sentences. Once policies have been translated into logic, a conflicting policy is detected in the same way independent of the language to define the policies, so our approach can easily be applied to various different policy definition languages.

4. FORMALIZATION OF POLICIES

In this section the definition of ζ for some policies is presented.

4.1 Authorization and Obligation Policy

The two most basic policies are an authorization policy and an obligation policy. We first present these policy definitions and their formalizations.

4.1.1 Authorization Policy

An authorization policy (`Auth+`) defines the action A_1 that a subject role S_1 is *permitted* to perform on a target role T_1. A negative authorization policy (`Auth-`) defines the action A_1 that a subject role S_1 is *forbidden* to perform on a target role T_1. These are represented by

$$\texttt{Auth}\pm(S_1,\ T_1,\ A_1).$$

4.1.2 Obligation Policy

An obligation policy (`Obli+`) defines the action A_1 that a subject role S_1 *must* perform on a target role T_1 when an event E_1 occurs. A negative obligation policy (`Obli-`) defines the action A_1 that a subject role S_1 *must not* perform on a target role T_1 when an event E_1 occurs. These are represented by

$$\texttt{Obli}\pm(E_1,\ S_1,\ T_1,\ A_1).$$

4.1.3 Formalization

The translation mapping ζ of authorization policies and obligation policies is defined as follows.

$$\begin{array}{rl} \zeta(\texttt{Auth+}(S_1,\ T_1,\ A_1)) & := \forall x(E_x \to P(S_1, T_1, A_1)) \\ \zeta(\texttt{Auth-}(S_1,\ T_1,\ A_1)) & := \forall x(E_x \to \neg P(S_1, T_1, A_1)) \\ \zeta(\texttt{Obli+}(E_1, S_1, T_1, A_1)) & := E_1 \to O(S_1, T_1, A_1) \\ \zeta(\texttt{Obli-}(E_1, S_1, T_1, A_1)) & := E_1 \to R(S_1, T_1, A_1) \end{array}$$

In the above translations, the predicate P can be read as "subject role S_1 is permitted to carry out action A_1 on target role T_1" and predicate O as "subject role S_1 must carry out action A_1 on target role T_1" and R as "subject role S_1 must not carry out action A_1 on target role T_1". The atom E_x says that event x occurs. Then, for example, the second and third translations can be read, respectively, as "for any event E_x, S_1 is forbidden to carry out A_1 on T_1" and "if event E_1 occurs then S_1 must carry out action A_1 on target role T_1".

Finally, there needs to be two axioms that relate P, O and R *i.e.* an obligation policy requires an authorization policy to permit the action and it contradicts a negative obligation policy:

$$\begin{array}{lcl} \text{Ax1} & : & \forall s, t, a(O(s,t,a) \to P(s,t,a)) \\ \text{Ax2} & : & \forall s, t, a(\neg(O(s,t,a) \wedge R(s,t,a))) \end{array}$$

Ax1 is used to detect conflicts involving both authorization and obligation policies and Ax2 is used to detect conflicts between positive and negative obligation policies.

4.2 Propagation and Action Composition Policy

4.2.1 Propagation Policy

As shown in Section 2.2.3, an authorization policy is defined by using a role that has a partial order relation and a propagation policy defines how an authorization policy propagates in accordance with the partial order. The syntax of the propagation policy is as follows.

$$\texttt{prop}(\texttt{Auth}+|-, \texttt{SRS}|\texttt{TRS}, \texttt{Up}|\texttt{Down})$$

`SRS` and `TRS` stand, respectively, for the subject and target role structures to which the propagation policy is applied. `Up` and `Down` define the direction of the propagation, where `Up` means that the policy propagates upward through the partial order from the least element, and `Down` means that the policy propagates downward from the greatest element.

The syntax of the propagation policy allows the following eight types of propagation policies to be defined.

$$\begin{array}{lcl} \text{prop1} & : & \texttt{prop}(\texttt{Auth+}, \mathcal{R} \in \texttt{SRS}, \texttt{UP}) \\ \text{prop2} & : & \texttt{prop}(\texttt{Auth-}, \mathcal{R} \in \texttt{SRS}, \texttt{Down}) \\ \text{prop3} & : & \texttt{prop}(\texttt{Auth+}, \mathcal{R} \in \texttt{SRS}, \texttt{Down}) \\ \text{prop4} & : & \texttt{prop}(\texttt{Auth-}, \mathcal{R} \in \texttt{SRS}, \texttt{UP}) \\ \text{prop5} & : & \texttt{prop}(\texttt{Auth+}, \mathcal{R} \in \texttt{TRS}, \texttt{UP}) \\ \text{prop6} & : & \texttt{prop}(\texttt{Auth-}, \mathcal{R} \in \texttt{TRS}, \texttt{Down}) \\ \text{prop7} & : & \texttt{prop}(\texttt{Auth+}, \mathcal{R} \in \texttt{TRS}, \texttt{Down}) \\ \text{prop8} & : & \texttt{prop}(\texttt{Auth-}, \mathcal{R} \in \texttt{TRS}, \texttt{UP}) \end{array}$$

More than one propagation policy or no propagation policy can be defined as required. These eight propagation policies are translated into the following four sentences.

$$\begin{array}{l} \zeta(\text{prop1}) = \zeta(\text{prop2}) := \\ \quad \forall x, y, z, a(P(x,y,a) \wedge H_{\mathcal{R}}(z,x) \to P(z,y,a)) \\ \zeta(\text{prop3}) = \zeta(\text{prop4}) := \\ \quad \forall x, y, z, a(P(x,y,a) \wedge H_{\mathcal{R}}(x,z) \to P(z,y,a)) \\ \zeta(\text{prop5}) = \zeta(\text{prop6}) := \\ \quad \forall x, y, z, a(P(x,y,a) \wedge H_{\mathcal{R}}(y,z) \to P(x,z,a)) \\ \zeta(\text{prop7}) = \zeta(\text{prop8}) := \\ \quad \forall x, y, z, a(P(x,y,a) \wedge H_{\mathcal{R}}(z,y) \to P(x,z,a)) \end{array}$$

where $H_{\mathcal{R}}(i,j)$ is a predicate stating that $i \in \mathcal{R}$ is a "senior" role of $j \in \mathcal{R}$ (*i.e.* i is greater than j in the partial order of $\mathcal{R}$). If you use the fact that $(A \wedge B) \to C$ is equivalent to $(\neg C \wedge B) \to \neg A$, you can easily prove that, for example, prop1 and prop2 policies are translated into the same sentence and similarly for the other cases shown above.

4.2.2 Action Composition Policy

An action composition policy is a policy that defines the relationship among actions operated in Web Services. The syntax of the action composition policy is defined by n actions $A_1, \cdots, A_n$,

$$A_1 = \Gamma(A_2, \cdots, A_n)$$

where Γ is a Boolean combination of $A_2, \cdots, A_n$. The mapping ζ for the action composition policy is defined as follows.

$$\begin{array}{l} \zeta(A_1 = \Gamma(A_2, \cdots, A_n)) \\ \quad := \forall x, y(P(x,y,A_1) \leftrightarrow \Gamma(P(x,y,A_2), \cdots, P(x,y,A_n))) \end{array}$$

4.3 Other Constraint Policies

In this section we present a definition of the mapping ζ for a Chinese wall[6], separation of duty[7] and time constraint policy as examples of other constraint policies. There are two types of separation of duty - static and dynamic [22], however, we discuss only static separation of duty and its conflicts in this paper.

4.3.1 Chinese Wall Policy

We specify the syntax of a Chinese wall policy for a set of targets $\{T_1, T_2\}$.

$$\text{cw1} : \texttt{CW}(all, \{T_1, T_2\}, all)$$

This Policy cw1 defines two mutually exclusive target roles. Namely, all subject roles can perform all actions for exactly one of the two targets $\{T_1, T_2\}$. The mapping ζ of this Chinese wall policy is defined as follows.

$$\zeta(\text{cw1}) := \forall x, y \neg(P(x, T_1, y) \wedge P(x, T_2, y))$$

If a Chinese wall policy must be defined for specific subject role or action in place of arbitrary ones, one can replace the arbitrary values x or y in the above formalization by a specific subject role or action such as S_1 or A_1.

4.3.2 Separation of Duty Policy

We specify the syntax of a separation of duty policy for a set of actions $\{A_1, A_2\}$.

$$\text{sod1} : \texttt{SoD}(all, all, \{A_1, A_2\})$$

This Policy sod1 specifies that two actions are mutually exclusive *i.e.*, all subject roles can perform exactly one of the actions $\{A_1, A_2\}$ for all target roles. The mapping ζ of this separation of duty is defined as follows.

$$\zeta(\text{sod1}) := \forall x, y(\neg(P(x, y, A_1) \wedge P(x, y, A_2)))$$

If a separation of duty policy must be defined for a specific subject or target role then replace the arbitrary values x or y in the above formalization by a specific subject or target role.

Note that more complex variations of the Chinese wall and separation of duty policies can be easily formalized. For example, a separation of duty for any finite set of mutually exclusive actions can be formalized by including additional predicates of the form $P(x, y, A_i)$ in Policy sod1. However, in this paper, we restrict the discussion to two mutually exclusive actions for simplicity.

4.3.3 Time Constraint Policy

A time constraint policy defines the time or period during which a policy becomes valid. In general a temporal logic may be best suited formalize the time constraint policy. However in this paper, by keeping to a simple time constraint policy, we present a formalization using first order logic.

Let $I_1, I_2, \cdots, I_n$ be a set of points that is defined on a time axis T, where $I_1 < I_2 < \cdots < I_n$. A time constraint for an authorization policy is specified as follows by a period $[I_a, I_b]$, $a \leq b$.

$$\texttt{Auth}\pm(S_1, T_1, A_1, [I_a, I_b])$$

An `Auth+` policy specifies that during the time period $[I_a, I_b]$ the subject role S_1 is permitted to perform the action A_1 on target role T_1. An `Auth-` policy specifies that during the time period $[I_a, I_b]$ a subject role S_1 is forbidden to perform the action A_1 on target role T_1. The translation mapping ζ for these time constraint policies is defined as follows.

$$\begin{aligned}
&\zeta(\texttt{Auth+}(S_1, T_1, A_1, [I_a, I_b])) \\
&\quad := \forall t(T(t, I_a, I_b) \rightarrow P(S_1, T_1, A_1, t)), \quad (I_a \leq I_b) \\
&\zeta(\texttt{Auth-}(S_1, T_1, A_1, [I_a, I_b])) \\
&\quad := \forall t(T(t, I_a, I_b) \rightarrow \neg P(S_1, T_1, A_1, t)), \quad (I_a \leq I_b)
\end{aligned}$$

where the predicate $T(t, I_a, I_b)$ can be read as a time t is contained in the time period $[I_a, I_b]$ and $P(S_1, T_1, A_1, t)$ can be read as subject role S_1 is allowed to perform an action A_1 on target role T_1 at time t. A positive authorization policy and negative authorization policy that are defined with a time constraint may lead to a conflict if their time periods overlap. To detect this type of conflict there needs to be two additional axioms.

$$\begin{aligned}
&\text{Ax3} : \quad \neg\exists x, y(T(t, I_x, I_{x+1}) \wedge T(t, I_y, I_{y+1}) \wedge x \neq y)) \\
&\text{Ax4} : \quad \forall x < \forall y(T(t, I_x, I_y) \leftrightarrow \textstyle\bigvee_{k=x}^{y-1} T(t, I_k, I_{k+1}))
\end{aligned}$$

Ax3 defines that at most one *unit time period* is always valid. Ax4 defines that $T(t, I_x, I_y)$ can be divided into a set union of unit time periods $T(t, I_x, I_{x+1}) \vee T(t, I_{x+1}, I_{x+2}) \vee \cdots \vee T(t, I_{y-1}, I_y)$.

5. CONFLICT DETECTION

In this section we show that our approach can detect a conflict and abduce the cause by using some examples.

5.1 Modality Conflict

Lupu et al. [16] mentioned that the following combinations of authorization and obligation policies may cause a modality conflict.

$$\{\texttt{Auth+}/\texttt{Auth-}\}, \{\texttt{Obli+}/\texttt{Obli-}\}, \{\texttt{Obli+}/\texttt{Auth-}\}$$

By using the mapping ζ defined in Section 4.1.3 and the tableaux method, every combination of modality conflict can be detected. As an example, in Fig.3 we show the result of analyzing the pair `Obli+/Auth-` and in particular that the following policies conflict.

$$\begin{aligned}
\text{Policy r17} \quad &: \quad \texttt{Obli+}(E_1, S_1, T_1, A_1) \\
\text{Policy r18} \quad &: \quad \texttt{Auth-}(S_1, T_1, A_1)
\end{aligned}$$

In the FVT, if inconsistent sentences can be made to appear in the same path, then the path is closed (indicated by a horizontal line in Fig.3). If all paths are closed, then the given sentences are conflicting.

A tableaux is developed as a tree, such that every piece of data is analyzed in every branch of the tree, unless a branch should already become conflicting. The analysis starts from the premise that the data are not conflicting and derives a contradiction, namely that all possibilities resulting from the assumption lead to contradiction. A datum is analyzed by considering the possible truth values of its constituents. For example, a sentence of the form $A \rightarrow B$ is true either if $\neg A$ is true or if B is true. This leads to two possibilities, represented in the tableaux by two branches. Basic rules to build a tableaux are presented in Table 1. A sentence $\forall x(E_x \rightarrow B)$ is true for each instance of the variable x. In the FVT method, a free variable is substituted for x, say x_1, to give the free variable instance $E_{x_1} \rightarrow B$, which is analyzed as above. That is, it is true either if $\neg E_{x_1}$ is true or if B is true. In the first branch of Fig.3, we can see that if E_1 is true the branch will close. Abduction allows us to assume the occurrence of event E_1, which is then available as an assumption in the other branches. In particular, it allows for the second branch to be closed, in the case x_1 is bound to 1. The third branch closes by use of Ax1. The final outcome of the analysis is that if event E_1 occurs then there can be a conflict for pairs of the form `Obli+/Auth-`. Other types of modality conflicts can be detected by the FVT method in a similar way.

Table 1: Tableaux Rules

[$\wedge$]	[$\vee$]	[$\rightarrow$]	[$\leftrightarrow$]	[$\neg$]
$A \wedge B$ A B	$A \vee B$ A B	$A \rightarrow B$ $\neg A$ B	$A \leftrightarrow B$ A $\neg A$ B $\neg B$	$\neg\neg A$ A
[$\neg\wedge$]	[$\neg\vee$]	[$\neg\rightarrow$]	[$\neg\leftrightarrow$]	[close]
$\neg(A \wedge B)$ $\neg A$ $\neg B$	$\neg(A \vee B)$ $\neg A$ $\neg B$	$\neg(A \rightarrow B)$ A $\neg B$	$\neg(A \leftrightarrow B)$ A $\neg A$ $\neg B$ B	A $\neg A$ close

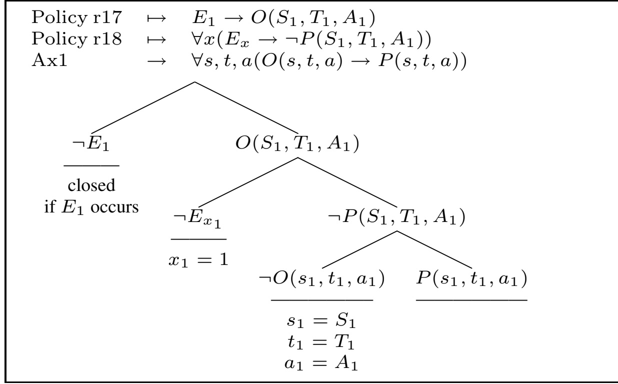


Figure 3: Modality Conflict

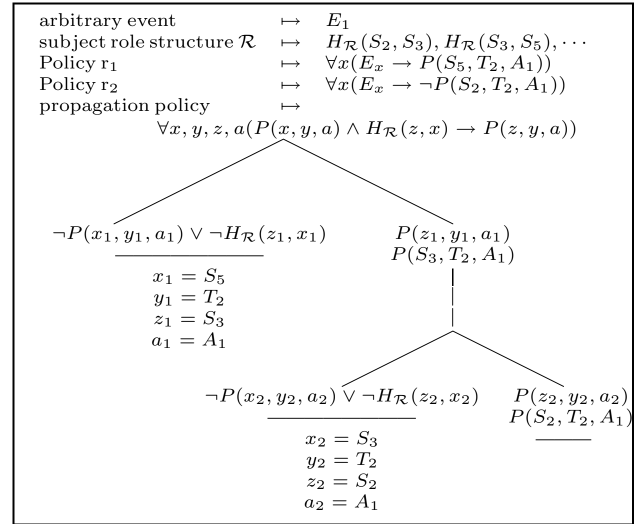


Figure 4: Conflict Caused by Propagation

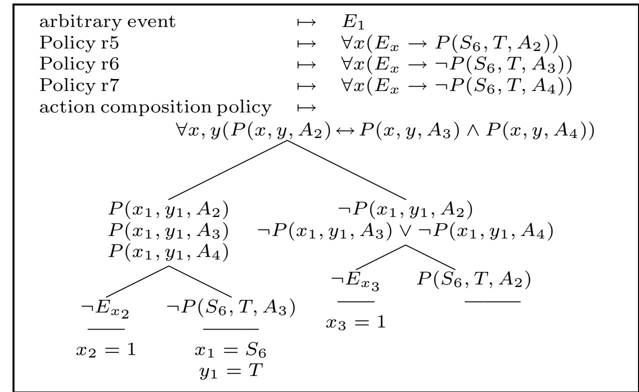


Figure 5: Conflict Caused by Action Composition

5.2 Conflict Caused by Propagation

We show that Policies r1 and r2 described in Section 2.2.2 are conflicting with respect to the propagation policy. Policies r1 and r2 are translated into the following sentences by using the definitions described in Section 4.1.3 and notations described in Fig.2.

$$\zeta(\text{Policy r1}) = \forall x(E_x \rightarrow P(S_5, T_2, A_1))$$
$$\zeta(\text{Policy r2}) = \forall x(E_x \rightarrow \neg P(S_2, T_2, A_1))$$

where A_1 stands for `play`. In this case, as a positive authorization policy should propagate upwards and a negative one should propagate downwards, we use the following type of propagation policy formalization.

$$\forall x, y, z, a(P(x, y, a) \wedge H_{\mathcal{R}}(z, x) \rightarrow P(z, y, a))$$

The result of analyzing policies r1 and r2 using the FVT method is shown in Fig.4. Since a conflict only happens if an event occurs, we assume an arbitrary event E_1 occurs. To simplify the diagram some details are omitted, however, all tableaux including latter examples have been worked through in detail. For example, in the first branch of Fig.4, if the variables $\{x_1,\ y_1,\ z_1,\ a_1\}$ are given the values $\{S_5,\ T_2,\ S_3,\ A_1\}$, then the branch contradicts with the assumption $H_{\mathcal{R}}(S_3, S_5)$ and Policy r2. From the tableaux we deduce that these policies conflict with each other and that the conflict is caused by the propagation $\{S_2, S_3, S_5\}$.

5.3 Conflict Caused by Action Composition

Next we show that Policies r5, r6 and r7 described in Section 2.2.4 become conflicting due to an action composition policy.

First the policies are translated by using the definitions described in Section 4.1.3:

$$\zeta(\text{Policy r5}) = \forall x(E_x \rightarrow P(S_6, T, A_2))$$
$$\zeta(\text{Policy r6}) = \forall x(E_x \rightarrow \neg P(S_6, T, A_3))$$
$$\zeta(\text{Policy r7}) = \forall x(E_x \rightarrow \neg P(S_6, T, A_4))$$

where $A_2,\ A_3,\ A_4$ are `rsv_travel`, `rsv_air` and `rsv_hotel` respectively. Second, the action composition policy given in Section 2.2.4,

$$\texttt{rsv_travel} = \texttt{rsv_air} \wedge \texttt{rsv_hotel}$$

is translated as follows.

$$\forall x, y(P(x, y, A_2) \leftrightarrow P(x, y, A_3) \wedge P(x, y, A_4))$$

The result of analyzing these policies using the FVT method is shown in Fig.5. To simplify the diagram some details are omitted. We can recognize that these policies conflict with each other since all branches are contradictory.

5.4 Conflict Caused by Constraint Policy

5.4.1 *Conflict Caused by Chinese Wall Policy*

In this section we show that the FVT method can also detect a static conflict of a Chinese wall policy. The following Policies r19, r20 and cw1 are examples of the conflict.

Policy r19 : $\texttt{Auth+}(S_1, T_1, A_1)$
Policy r20 : $\texttt{Auth+}(S_1, T_2, A_1)$
Policy cw1 : $\texttt{CW}(all, \{T_1, T_2\}, all)$

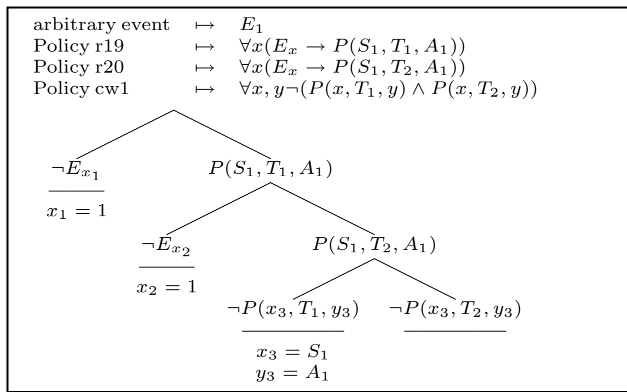


Figure 6: Conflict Caused by Chinese Wall Policy

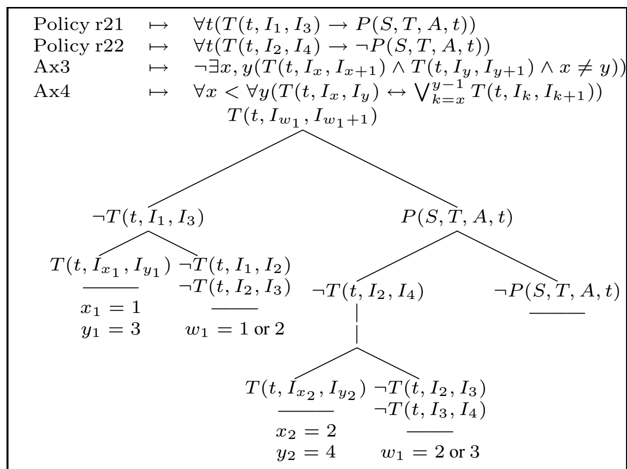


Figure 7: Conflict Caused by Time Constraint Policy

Policy cw1 specifies that exactly one of the targets T_1 or T_2 can be accessed. However, according to the Policy r19 and r20, subject role S_1 can access both targets; that is, these three policies are conflicting. The result of analyzing these policies using the FVT method are shown in Fig.6. Again we can recognize that these policies conflict with each other since all branches are contradictory. A static conflict of separation of duty policy can also be detected by the FVT method in a similar way.

5.4.2 *Conflict Caused by Time Constraint Policy*

As a last example we show that a conflict caused by time constraint policy defined in Section 4.3.3 can be detected using the FVT method. We use the following example Policies r21 and r22.

$$\text{Policy r21} \;:\; \texttt{Auth+}(S, T, A, [I_1, I_3])$$
$$\text{Policy r22} \;:\; \texttt{Auth-}(S, T, A, [I_2, I_4])$$

These policies conflict with each other because the time periods $[I_1, I_3]$ and $[I_2, I_4]$ are overlapping for the same subject role, target role and action.

The result of analyzing these policies using the FVT method is shown in Fig.7. In Fig.7 we assume that an event t occurs in some time unit $[I_{w_1}, I_{w_1+1}]$, where w_1 is to be determined. To simplify the diagram some details are omitted. From the result we can not only detect that these are conflicting but also abduce that the conflict occurs during the time period $[I_2, I_3]$ since we can get $w_1 = 2$ by combining the result $w_1 = \{1, 2\}$ and $w_1 = \{2, 3\}$. Namely, to resolve the conflict we need to eliminate the overlapping period $[I_2, I_3]$ from the Policies r21 and r22.

6. RELATED WORK

P.C.K.Hung [12] mentions a conflict of interest, which is used to define a Chinese wall policy, and separation of duties for a Web Services environment. Also R. Bhatti et al.[4] proposes a policy description language, called X-RBAC, developed to realize role based access control in Web Services environment. Moreover, most policy description languages, for example XACML[17] and Ponder[8], can define time constraint policy. However, none of the methods seem to refer to policy conflict.

There are some static conflict detection methods discussed in the literature. For example, Ribeiro et al.[20] present a method to detect some inconsistent rules logically and statically, whilst S. Jajodia et al.[13] describe a method which detects conflicts by using derivation rules. Also M. Strembeck [23] presented a method to detect a static separation of duty conflict caused by propagation. However, these approaches do not provide information about the cause of the conflict.

Several approaches to detect and resolve conflicting policies can be found. Lupu et al. [16] discuss that conflicts may occur due to the overlap of the domains to which subjects and objects belong. A method to resolve the conflict by using priorities based on the relationship of these domains is proposed. However, their approach uses an implicit propagation policy defined by the domain structure and does not deal with composite actions.

Other approaches mention conflicts that occur due to the hierarchical structure of the underlying organization and the associated propagation policies. Several methods to prevent such conflicts by precedence are proposed. For example, S. Jajodia et al.[13] propose to resolve conflicts by using default rules such as “deny override”. In XACML[17], *Deny-overrides*, *Permit-overrides* and *First-applicable* can be defined as default rules. The novel technique presented in [3] works by inferring rule priorities based on the role structure. However, these approaches may not always yield the result that an administrator really intends; for instance, even in a single system, the priorities of the various rules may differ depending on whether the situation is normal or an emergency. Therefore, before the system starts working, either conflicting rules should be statically detected and notified together with the reasons to the administrator, who should then specify a method to resolve them, or an application specific precedence policy is required.

7. CONCLUSION AND FUTURE WORK

In this paper we have presented an approach to statically detect a conflicting policy for an aggregation Web Services environment by using free variable tableaux, which is a sound and complete theorem prover that can be used to show inconsistency and upon which can be built abductive reasoning. It is realized by translating each access control policy into logic. Our method can detect not only modality conflicts but also constraint conflicts such as propagation, Chinese wall, time constraint and so on, all in a uniform way. We have also ensured the usability of the approach by showing how the conflicting policy can be detected and the conflicting information that is very helpful to resolve the policy can be obtained. As the tableaux method is sound and complete, it is guaranteed that all conflicting policies can be detected. Moreover, it has the additional advantage that it can be applied to various policies written in different policy definition languages.

In the near future, we will try to extend our method into three directions:

i) **Extension :** Our method could be applied not only for policies introduced in this paper but also for other types of policies; for example a delegation policy or a devolution policy may be needed in an e-government environment. We will consider the formalization of these policies. Moreover, we are investigating how techniques such as temporal logic and event calculus [15] could be included into the method to cope with more complicated time constraints such as cyclical events.

ii) **Evaluation :** We will evaluate the computational complexity of the method and compare it with other similar approaches for detecting conflict.

iii) **Implementation :** There are tools named leanTAP [2] or leanCoP [19], which are implementation of the free variable tableaux. We will extend this to include abduction and use it to develop a tool that detects conflicting policies, written in such as Ponder [8] or XACML [17].

8. ACKNOWLEDGMENTS

We would like to thank Alessandra Russo, Naranker Dulay, Emil Lupu, Arosha Bandara and Shuichiro Yamamoto for their many helpful comments and suggestions.

9. REFERENCES

[1] A. H. Anderson. An Introduction to the Web Services Policy Language (WSPL). In *Proceedings of the Fifth IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY'04), June 07 - 09, 2004, New York, USA*, pages 189–192. IEEE Computer Society, June 2004.

[2] B. Beckert and J. Posegga. leanT^AP: Lean Tableau-based Deduction. *Journal of Automated Reasoning*, 15(3):339–358, 1995.

[3] S. Benferhat, R. E. Baida, and F. Cuppens. A Stratification-based Approach for Handling Conflicts in Access Control. In *SACMAT '03: Proceedings of the eighth ACM symposium on Access control models and technologies, Como, Italy*, pages 189–195. ACM Press, June 2003.

[4] R. Bhatti, J. B. D. Joshi, E. Bertino, and A. Ghafoor. Access Control in Dynamic XML-based Web-Services with X-RBAC. In *Proceedings of the International Conference on Web Services, ICWS '03, June 23 - 26, 2003, Las Vegas, Nevada, USA*, pages 243–249. CSREA Press, June 2003.

[5] D. Box, F. Curbera, M. Hondo, C. Kaler, D. Langworthy, A. Nadalin, N. Nagaratnam, M. Nottingham, C. von Riegen, and J. Shewchuk. Web Services Policy Framework (WS-Policy) Version 1.01. June 2003. http://www-106.ibm.com/developerworks/library/specification/ws-polfram/.

[6] D. F. C. Brewer and M. J. Nash. The Chinese Wall Security Policy. In *Proceedings of the IEEE Symposium on Security and Privacy, May 01 - 03, 1989, Oakland, California, USA*, pages 206–214. IEEE Computer Society, May 1989.

[7] D. D. Clark and D. R. Wilson. A Comparison of Commercial and Military Computer Security Policies . In *Proceedings of the IEEE Symposium on Security and Privacy, California, USA*, pages 184–194. IEEE Computer Society, April 1987.

[8] N. Damianou, N. Dulay, E. Lupu, and M. Sloman. The Ponder Policy Specification Language. In *Proceedings of the Policy 2001: Workshop on Policies for Distributed Systems and Networks, Bristol, U.K.*, pages 18–39. Springer-Verlag LNCS 1995, January 2001.

[9] D. F. Ferraiolo, R. Sandhu, S. Gavrila, D. R. Kuhn, and R. Chandramouli. Proposed NIST standard for role-based access control. *ACM Transactions on Information and System Security*, 4(3):224–274, August 2001.

[10] M. Fitting. *First Order Logic and Automated Theorem Proving*. Springer, second edition, 1996.

[11] H. He, H. Haas, and D. Orchard. Web Services Architecture Usage Scenarios. *W3C Working Group Note*, February 2004. http://www.w3.org/TR/2004/NOTE-ws-arch-scenarios-20040211/.

[12] P. C. K. Hung. From Conflict of Interest to Separation of Duties in WS-Policy for Web Services Matchmaking. In *Proceedings of the 37th Annual Hawaii International Conference on System Sciences (HICSS'04), Track 3, January 05 - 08, 2004, Hawaii*, page 30066b, January 2004.

[13] S. Jajodia, P. Samarati, and V. S. Subrahmanian. A Logical Language for Expressing Authorizations. In *Proceedings of the 1997 IEEE Symposium on Security and Privacy, May 04 - 07, 1997, Oakland, California, USA*, pages 31–42. IEEE Computer Society, 1997.

[14] H. Kamoda, A. Hayakawa, M. Yamaoka, S. Matsuda, K. Broda, and M. Sloman. Policy Conflict Analysis Using Tableaux for On Demand VPN Framework. *Proceedings of the the First International Workshop on Trust, Security and Privacy for Ubiquitous Computing (TSPUC 2005), Taormina, Sicily, Italy*, June 2005.

[15] R. A. Kowalski and M. J. Sergot. A Logic-based calculus of events. *New Generation Computing*, 4(1):67–95, 1986.

[16] E. C. Lupu and M. Sloman. Conflicts in Policy-Based Distributed Systems Management. *IEEE Transactions on Software Engineering*, 25(6):852–869, November 1999.

[17] OASIS. eXtensible Access Control Markup Language (XACML) Version 1.1. *OASIS Standard*, July 2003.

[18] OASIS. Core and Hierarchical Role Based Access Control (RBAC) profile of XACML, Version 2.0. Committee Draft 01, November 2004. http://www.oasis-open.org/.

[19] J. Otten and W. Bibel. leanCoP: Lean Connection-Based Theorem Proving. *Journal of Symbolic Computation*, Volume 36, pages 139–161. Elsevier Science, 2003.

[20] C. Ribeiro, A. Zúquete, P. Ferreira, and P. Guedes. Security Policy Consistency. *Technical Report, INESC*, June 2000.

[21] Y. Sakata, K. Yokoyama, and S. Matsuda. A Method for Composing Process of Non-deterministic Web Services. In *Proceedings of the IEEE International Conference on Web Services (ICWS'04), California, USA*, pages 436–. IEEE Computer Society, June 2004.

[22] R. Sandhu. Separation of Duties in Computerized Information Systems. In *Proceedings of the IFIP WG11.3 Workshop on Database Security, U.K.*, September 1990.

[23] M. Strembeck. Conflict Checking of Separation of Duty Constraints in RBAC - Implementation Experiences. In *Proceedings of the Conference on Software Engineering (SE2004), Austria*, pages 224–229, February 2004.

[24] UDDI Organization. UDDI Specification. *Version 3.0, Published Specification*, 2002. http://www.uddi.org/.

[25] W3C. Web Services Description Language (WSDL) 1.1. March 2001. http://www.w3.org/TR/wsdl.

[26] W3C. SOAP Version 1.2. June 2003. http://www.w3.org/TR/soap/.

[27] H. J. Wang, H. K. Cheng, and J. L. Zhao. Web Services Enabled E-Market Access Control Model. *International Journal of Web Services Research*, 1(1):21–40, 2004.

Methods for Policy Conflict Detection and Resolution in Pervasive Computing Environments

Evi Syukur
SCSSE, Monash University, Australia
evis@csse.monash.edu.au

Seng Wai Loke
SCSSE, Monash University, Australia
swloke@csse.monash.edu.au

Peter Stanski
Telstra Lab, Australia
peter.stanski@stanski.com

ABSTRACT

Recently, there has been increasing work in using policy in pervasive systems. Policy is a relatively new field and much work is still required to explore designs, concepts, and architecture for using policy in pervasive computing environments. In this paper, we briefly introduce the concepts and design of a policy based pervasive system, using Mobile Hanging Services as an example. The main aim of this paper is to investigate several techniques that can be used to statically or dynamically detect and resolve conflicts in pervasive systems. We discuss the conflict detection and resolution techniques in the system as a case study.

Categories and Subject Descriptors

D.2.11[**Software Engineering**]:Software Architectures; H.3.4[**Information Storage and Retrieval**]:Systems and Software; K.6.3[**Management of Computing and Information Systems**]:Software Management.

General Terms

Design, Performance and Management.

Keywords

Policy, Conflict Detection, Conflict Resolution, Web Services, Context, Mobile device, and Pervasive System.

1. Introduction and Motivation

Pervasive computing has a broad view of utilizing computing devices everywhere in the environment and at any time [1]. The idea is that a mobile or non-mobile user can communicate with embedded or non-embedded computing devices, which are invisibly integrated into the environment as soon as s/he steps into that particular space. To date, we have seen a number of pervasive computing systems that have been developed and many of them share similar concepts, although the details of each concept may be different one from another, depending on the target domain of the pervasive system. These basic concepts of the pervasive system are the notions of entities, spaces, services, mobile devices, workstations and contexts.

Recently, there has been increasing work in designing policy based pervasive systems. In our case, policy is used to express a set of rules to govern and control the behaviours of entities in accessing services in specific contexts. Having the additional policy mechanisms in pervasive systems would certainly benefit the user. For example, it allows the users to constrain and control the behaviors of foreign entities operating in his/her environment, and it is used for humans to tell a system what task to do automatically within a certain situation [11]. However, there are some challenges in developing such a system. One of the main challenges we focus on in this paper is detecting and resolving conflicts in an efficient and appropriate manner as they arise in the context of using policies to control mobile services. Conflicts often arise as a result of the differences in policy specifications: e.g., one allows the user to start the service but another prohibits the user from doing so. From our study, we experienced that in pervasive systems, the possibility of conflict occurrence is higher than in other systems (i.e., a distributed system). This is mainly due to a number of contexts and services used, and the mobility of entities, in which, the entity can move freely from one geographical space to another and the entity carries its own rules on how the service should be executed in the designated place.

Due to a number of possible conflicts that may occur in a pervasive environment and each of these conflicts may need different detection and resolution strategies (due to its source of occurrence), we may require a number of techniques to detect and resolve the conflict efficiently. The research presented in this paper attempts to tackle the above issues in our framework for Mobile Hanging Services (MHS). MHS supports policy mechanisms by having and publishing policy software components as Web services. We also propose several techniques for conflict detection and resolution in our pervasive system. We then compare these techniques by considering several aspects of the system such as:

a. System performance - how long it takes to detect or resolve the conflict. The shorter time it takes to detect or resolve the conflict, the faster it is to respond to the user's request (hence, minimizing the user wait time).
b. Implementation - how easy it is to implement such techniques.
c. Accuracy - how often we need to update the conflict detection or resolution result.
d. Does it accommodate all conflicts that may happen in the future?

The rest of this paper is organised as follows. In section 2, we give an overview of the policies in our pervasive system including several possible sources and types of conflicts. In section 3, we describe several general techniques used for conflict detection. In section 4, we discuss general strategies used to resolve the conflict. In section 5, we present a case study: a campus based mobile services system using policy (a MHS application). In section 6, we discuss in detail each of the proposed conflict detection and resolution techniques and compare them. In section 7, we present related work. In section 8, we draw overall conclusions and present future work.

2. Background

This section discusses a definition of policy, followed by an overview of various possible sources of conflicts in pervasive computing environments.

2.1 Definition of Policy

The purpose of the policy is to constrain the behaviours of entities in particular contexts and to ensure that their behaviours (actions performed) are aligned with the rules of the system. A policy language in a pervasive environment can be enriched by

Copyright is held by the author/owner(s).
WWW 2005, May 10--14, 2005, Chiba, Japan.

supporting various kinds of normative notions [3,12]. Three basic deontic logic notions that we focus on are:

- Right (R) refers to a permission (positive authorization) that is given to the entity to execute a specified action on the service in the particular context.
- Obligation (O) is a duty that the entity must perform in a given context.
- Prohibition (P) is a negative authorization that does not allow the entity to perform the action as requested in the given context.

2.2 Policy Conflict Sources and Types

In the pervasive MHS system, we may assign different policy specifications to each entity depending on the role that s/he has. Assigning different policy specifications to each user in the system is a way to limit and control the user's behaviours. However, this could also lead to a conflict as the conflict arises due to some differences including:

(a) Policy space modality conflict: conflict occurs as the space (i.e., can be the system space or room space) assigns different specifications on what an entity can do with the service i.e., one allows the user to start the service (i.e., a system) and another prohibits the user from starting the service (i.e., a room) or a room obligates to start a service and at the same time, the user is obligated by the system to stop the service.

These differences lead to a potential or actual conflict that needs to be resolved. In our definition, a potential conflict refers to a conflict that has not happened yet at the time the system detects that such a conflict can happen, as the context or condition for the conflict to occur has not been met. The potential conflict can be further classified into two different types: possible potential conflict and definite potential conflict.

The *possible potential conflict* is a conflict where the possibility of the occurrence is less than the definite potential conflict. This conflict may still not happen even in the right user contexts of location and time. For example, a system allows the user to "start any service" but the room only allows the user to "start media player service". "Any" here means all services which are available for the user in that context. It includes the media player service and some other services in the context. The conflict only occurs if the user starts any service other than the media player service. The conflict will not occur if the user starts the media player service. Hence, we categorize this conflict as a potential conflict with the type *possible*. The *definite potential conflict*, on the other hand, refers to a conflict that will definitely occur if the user is in the right context. For example, a system allows the user to "start media player service" but the room prohibits the user from "starting this service". Once the user is in the right context, this definite conflict will become an actual conflict, as one allows the user and the other prohibits the user.

b) Role conflict: it occurs due to the differences in the privilege that the entity has. For example, one user (with higher privilege) can execute more types of services at any time and any place compared to other users (with lower privilege) who can only execute certain number of services at certain place and time. In our system, the level of privilege is determined based on the level of positions or roles that the user has. As each entity has a different level of privileges, a user with higher level of role may override the execution of the shared service that has been started earlier by a user with lower role. This then leads to a conflict.

c) Entities conflict: it occurs if two or more users have different policy specifications or intentions of what to perform on the service that is running on the same shared resource device. For example, one user wants to start a music service but another user wants to stop this music service which is currently running on the same target machine.

3. Policy Conflict Detection

In this section, we briefly describe goals of conflict detection, followed by several strategies used to detect conflicts in a pervasive computing environment.

3.1 Goals of Conflict Detection

The primary goal of detecting a conflict is to investigate several possible sources of conflicts and types that may occur within the system. Knowing that there is a potential conflict would allow the system to accommodate the conflict resolution earlier. Hence, by the time it occurs, the system is ready with the resolution result. There are also several sub-goals of conflict detection:

a. to group the conflicts based on its type i.e., a possible potential conflict or a definite potential conflict (see section 2.2). This is useful to decide on when to resolve the conflict.
b. to analyse the probability of the conflict occurrence (i.e., normally a possible potential conflict has lower possibility of occurrence compared to a definite potential conflict).
c. to investigate the best technique for conflict detection based on the sources and types of the conflict.
d. to predict the number of occurrences of the conflicts; hence, we can assign the best technique to detect and resolve this particular conflict.
e. to predict the probability of potential conflicts which will become actual conflicts. This is useful to decide when to resolve the conflict. For example, if we can predict that the potential conflict never happens, the conflict resolution for this type of conflict may not be necessary.

3.2 Conflict detection strategy

It is imperative to make a clear distinction on when and where to perform the conflict analysis (conflict detection and resolution), as it can be computationally intensive, time and resources consuming. By analyzing several possible sources of conflicts that may happen in pervasive environments, we propose two different techniques to detect a conflict.

1. Static conflict detection

Static conflict detection aims to detect all types of potential conflicts (possible or definite) which clearly could cause conflicts from the policy specification. This static conflict detection is performed offline on the client side or on the server side. Performing the static conflict detection on the client side is less desirable as it slows down the conflict detection process. This is due to some constraints i.e., limited resources, power and processing speed on the mobile device. The only advantage is the conflict detection result is there on the mobile client side as the user needs it (hence, it does not have to be transferred to the client device). On the other hand, performing static conflict detection on the server side has more advantages compared to the client side i.e., the server (normally a desktop PC) has larger memory size and faster processing speed, and so, can detect the conflict faster. The result can then be pushed onto the mobile client when done.

With static conflict detection, we also need to decide on types of conflicts that we need to detect i.e., whether we only want to detect conflicts which are clearly specified in the policy specification (*predicted potential conflict*) or we want to detect some other conflicts which are not conflicts yet from the policy specification, but they could lead to conflicts if one or more entities are in the space at the right contexts (*unpredicted potential conflicts*). To include all unpredicted potential conflicts

will certainly speed up the performance in responding to the user's requests (as it has detected all possible conflicts). The only drawback is it may use up a lot of system resources (i.e., memory and processing speed), as it has to detect the conflict based on all possible combinations of entities, contexts and services that the system has. Moreover, some of the conflict detection results may never be used as the entities may never be in a context as predicted (hence, the conflict may never occur).

Another issue that needs to be taken into consideration is to decide on how often the cached detection result needs to be updated (i.e., if we cache the conflict detection result for future re-use). The detection results may be outdated as perhaps, there are more users registering with the system or some users have modified their policy specifications. To address this issue, several approaches can be incorporated: (a) frequently (i.e., every 5 minutes), (b) periodically (i.e., every Monday) (c) only when the system detects that the user has modified the policy specification or when there is a new user registered with the system.

2. Dynamic conflict detection

Unlike static conflict detection, dynamic conflict detection is performed at run time by dynamically detecting all unpredicted potential conflicts between a number of entities in the given contexts. As dynamic conflict detection is performed some time at run time, the system needs to decide on when to trigger this detection module. We propose five different strategies on when to dynamically detect a conflict.

a. Reactive model

As it is reactive, this dynamic conflict detection is only triggered when there is an explicit request from users i.e., when the user clicks on any action name (start, stop, pause, resume, or submit) from a mobile device to request an action on the service. The detection is done as soon as the system detects that there is a request from the user. If there is a request, the system then collects all the entities' context information and reactively detects the conflicts between those entities in the given context.

This technique is best in the situation with only a few requests from an entity. It takes some time to detect conflicts if there are many requests from the entities. In addition, the detection is only limited to the current location, day, and time, which are related to the requested action and only between the requested user against all other users in the room (not all users in the system).

b. Proactive model

Proactive conflict detection tends to implicitly and automatically detect the conflict by sensing the user's current context i.e., when the user moves in or out of the room. This technique is best used in the situation where performance is paramount. The proactive conflict detection detects all the potential conflicts that may occur in the given context and may cache the result for future re-use. The proactive technique is also considered as pessimistic conflict detection. We are pessimistic that there will be a conflict between those entities in the room, as each entity may try to perform different actions for the same shared service. Hence, the system proactively catches all potential conflicts that may occur in the given context. In addition, this technique is considered useful only if the participating entities (i.e., users) are still in the same context where the conflict is predicted to happen. If one of these entities has moved to a different location, the predicted potential conflict may no longer be an actual conflict (as this type of conflict only occurs if two or more entities which have different specifications on the same target service are still in the same space).

Moreover, there are two issues that need to be addressed in order to increase the accuracy of dynamic conflict detection result:

- What happens if in the middle of process of detecting a conflict, another user comes in? Will the system continue with the detection process? If it continues, it then has to re-compute the result after some time, as it is already outdated.
- What happens if the user has left the space and this user is already in the conflict detection list result? Do we need to remove him/her from the list? What happens if s/he comes back to the space after some time? We need to know when to remove users from the list. Also, there is a problem, if we keep all the results in the memory, as the server may be overloaded with outdated results and perhaps, there is no longer a conflict between users (as one of the conflicted users is no longer in the space).

c. A combination of reactive and proactive models

A combination of these techniques is useful when we want the system to act proactively in a certain situation i.e., in an examination room, a seminar room and in a certain place, it acts reactively i.e., in the individual room. This is mainly because, at a public place, there are many users coming in and out of the place, therefore, it is useful to employ a proactive conflict detection technique here. At the individual room, usually, only the owner with some other visitors that may not perform many activities, hence, we detect the conflict reactively. A decision to choose whether to perform a proactive or reactive behavior can be based on: **(i) the location** i.e., proactive in public place and reactive in the individual place (i.e., a user's office). **(ii) the day and time** i.e., on Monday at any place, proactively detects the conflict, because, it may be a busy day and many students come to the University or at the shopping centre, there may be a lot of visitors visiting the mall, but, other days, we detect the conflict reactively. **(iii) the number of users in the location**. For example, if the system detects there are more than five users in the location, a proactive behaviour is used. However, if there are less than five users, the system then detects the conflict reactively.

d. Predictive model

Predictive model detects the conflict based on the user's history file. By analyzing the user's history file, the system can predict the user's movement and the person that the user is going to meet. For example, from the history file, user A is always going to room B and meeting user B on Wednesday at 12PM. Based on this information, the system may want to compute the conflict detection proactively between these users (user A and B) at room B. This technique is considered useful only if the system prediction is correct (i.e., the user always does the same activity as listed in the history file). However, if the user's movement and activity are not anticipated by the system (i.e., the user is moving to a different room and meeting different people), there will be a delay in responding to the user's request. This is due to the conflict detection result which has been previously computed is irrelevant to the user's current context. Hence, the system will need to re-detect the conflict based on the user's current location, day, time and people that s/he is meeting.

4. Policy Conflict Resolution

When there is a potential or actual conflict detected by a conflict detection module, it becomes necessary to resolve the conflict. Several aspects discussed in this section are the goals of the conflict resolution, when and how to resolve conflicts, as well as when to update the conflict resolution result.

4.1 Goals of Conflict Resolution

The primary purpose of conflict resolution is to resolve all types of conflicts in minimum amount of time, and so, minimizes the user wait time. Several sub-goals of conflict resolution are:

a. to investigate several techniques on how to resolve the conflict based on its sources and types.
b. to decide when it is the best time to resolve the potential or actual conflicts.
c. to monitor whether the conflict resolution result satisfies both of the conflicted entities. If the conflict resolution result does not satisfy the conflicted entities, we need to think of the best solution that will benefit both of these entities i.e., allowing the conflicted entities to challenge the system and resolving the conflict by taking into account the user's current situations.
d. to decide on how often the conflict resolution module needs to be re-computed.
e. to analyse whether the conflict resolution result is useful (i.e., the conflict resolution result will be used at run time, as the predicted potential conflict becomes an actual conflict).

4.2 Techniques to resolve the conflict

We propose several conflict resolution techniques to handle possible conflicts that may occur in pervasive systems. Some additional resolution techniques or further refinements of each of the following resolution techniques are required depending on the target pervasive domain. This paper discusses only the major conflict resolution techniques which can be used across pervasive systems that employ and share the basic pervasive concepts as discussed earlier in introduction. These resolution techniques are **(a) Role hierarchy overrides policy.** The role hierarchy overrides policy is used if the conflict occurs between users who have different roles, in which a user with a higher role can override the policy that belongs to the user with a lower level of role. **(b) Space holds precedence over visitor.** This technique is used if a conflict occurs between a user and a room. For example, the system permits a user to start a service at room A, but room A prohibits the user from starting this service. If there is a conflict, the room (representing its owner) always wins, regardless of the levels of role of the visitor. **(c) Obligation holds precedence over rights.** This technique is used if a conflict occurs between an obligation and the right. An obligation always wins over the right. For example, a user is permitted by the system to start a media player service, but a room obligates the user to stop this service.

4.3 When to resolve the conflict

We propose two strategies on when to resolve the conflict in pervasive computing environments.

a. At the time when a conflict is detected

This is a *pessimistic conflict resolution* technique. We are pessimistic that some or all detected potential conflicts will become actual conflicts. Hence, the system resolves all conflicts immediately as soon as the system detects them. Depending on the conflict detection technique that the system employs, with this technique, the conflict can be resolved offline (i.e., when users are not in the space yet) or at run time. For example, if we employ a static conflict detection technique, the conflict resolution of all potential conflicts is done offline, as soon as they are detected. However, if the system employs a dynamic conflict detection (i.e., a reactive technique), the conflict resolution is only performed at run time, as the conflict is only detected at run time.

In addition, with this technique, we can further choose which conflicts to resolve based on its type such as: **(i) Resolve only a definite potential conflict:** The technique here resolves only a definite potential conflict, as we are sure that it will become an actual conflict once the entities are in the right contexts for the conflict to happen and resolve the possible potential conflict only when the contexts for the conflict to happen are met. This technique does not anticipate all resolution results. Hence, it may experience a delay in responding to the user's request, especially if the possible potential conflict happens to be an actual conflict at run time. **(ii) Resolve both possible and definite potential conflicts**. The system can also choose to resolve both types of conflicts as soon as they are detected. These potential conflicts are solved, though they have not happened yet to be actual conflicts. This technique would minimize the user wait time, as it has resolved all predicted conflicts prior to become actual conflicts. However, if none of the predicted conflicts become actual conflicts, it may waste the system resources.

b. At the time when the potential conflict becomes an actual conflict (normally at run time)

This is an *optimistic conflict resolution* technique. We resolve the potential conflicts just when they become actual conflicts. We do not resolve these potential conflicts, just when we detect them, as we are optimistic that the conflicts that we have detected may or may not become actual conflicts. This is due to several factors such as the user may not be in the context where the conflict is detected to happen or the user does not execute the service in the specific context (i.e., specific location, day and time) where a conflict can arise (although, it is clearly a conflict from the policy specification). For example, the user is allowed by the system to start a media player service at any day, however, the room only allows the user to start this service on Monday only. We are optimistic that the conflict here will not happen, unless the user starts the service on any other days (other than Monday).

4.4 How often to update the conflict resolution result

It also would be good to cache the conflict resolution result for future re-use. The question here again, we need to decide on how often the cached result needs to be updated. One simple solution is to update each time the conflict detection module is re-computed (when the cached conflict detection result is updated).

5. Case Study

This section discusses in detail on how policy specification, conflict detection and resolution strategies are used in pervasive computing environments. One sample prototype that we have developed is a campus based policy system within MHS.

5.1 MHS on Campus

As discussed earlier in introduction, our definition of a pervasive computing environment consists of entities, spaces, services, mobile devices, workstations and contexts. The details of each of these concepts depend on the target pervasive system and its environment. For example, an entity in a campus domain refers to a student, a lecturer and a head of school, however, in a shopping mall domain, it could mean something different i.e., a customer and a seller. In this section, we mainly focus on the pervasive concepts and policy specifications in a campus domain. We describe each of these concepts as follows:

a. **Entities.** Entities here refer to mobile users which are always on the move (move from one geographical space to another). Three types of entities in our system are a student, a lecturer, and a head of school. By default, our **s**ystem imposes certain **r**ights (denoted by sRe), **o**bligations (sOe) and **p**rohibitions (sPe) to each of these **e**ntities depending on the role that the entity has and the physical space that the entity is visiting. In addition, each of the **e**ntities in the system can also impose a certain **o**bligation to the

system (eOs), created via a user policy application that we have. In summary, each of the entities in the system will have:

sRe_i, sOe_i, sPe_i and e_iOs

Note: i denotes a specific user i.e., user **i**.

b. **Spaces.** Spaces here can be a physical room that is represented by a geographical location e.g., room B558. The room entity has its own policy that can be used to restrict the visitors' behaviors or actions on mobile services in the room. Generally, the room's policy is created by the owner of the room. The public place in our system (e.g., tea room, corridor, or seminar room) is owned by the system. Hence, the public policy is created by the system (i.e., a developer/system administrator).

c. **Services.** A service refers to a software tool that is enlisted as users need it and it helps users to accomplish the tasks by downloading the service application or mobile code onto a target machine (i.e., a mobile device or a desktop PC machine). We have two types of services in our system: a shared resource service e.g., Mobile VNC [10] and Mobile Media Player applications [11], in which the service is downloaded onto a shared desktop machine and it can be controlled and accessed by all legitimate users from their mobile devices in that specific location. A non-shared resource service, on the other hand, is a service that is downloaded to and compiled in the user's mobile device (e.g., a Mobile Pocket Pad Service [9] and is only accessible by that user).

d. **Mobile Devices** i.e., handheld devices which display service interface and can execute service processes.

e. **Workstations**. It can be a normal desktop PC where services can be executed (run) or a server that hosts all context-aware and policy related components.

f. **Contexts**. Contexts are conditions that must be met before a list of services can be displayed on the mobile device or before the user's request to perform an action is approved. In our work, contexts consist of a user's identity, location, day and time.

5.1.1 Architectural Design

Our policy software components handle the user's request to perform some actions on the service. The request can be start, stop, pause and resume the service. This section provides a high level architecture and description of these parts of our mobile policy based framework (see Figure 1 below).

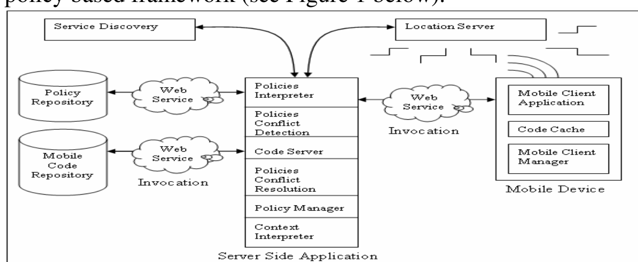


Figure 1: High Level Architecture of MHS Policy Framework

The details of each of our context-aware software components have been discussed in [9]. We now describe each of our policy software components: **(a) Mobile client query manager (on the mobile client side).** It handles the request from the user and sends this request to the policy manager. **(b) Policy manager (on the server side).** Policy manager manages the interaction between the mobile client and the server, in which the mobile client sends a request to the policy manager and the policy manager computes the request and returns the result back to the client. The result is either allowing or disallowing the mobile user to perform the action. **(c) Policy interpreter (on the server side).** The policy interpreter component specifies a set of rights, prohibitions and obligations which are useful for the user in the particular contexts. **(d) Policy conflict detection module.** The policy conflict detection detects lists of potential or actual conflicts that may occur between entities in the system. **(e) Policy conflict resolution module.** The policy conflict resolution module handles conflicts between entities in the system.

5.1.2 Prototype Implementation Details

We present our prototype implementation where we have implemented some of the conflict detection and resolution techniques discussed in previous sections. Our MHS system consists of users with mobile devices who are always on the move, a web service that determines the user's current location, and policy software components which handle a user's request to perform an action on a particular service.

As for conflict detection, our system employs a combination of static and dynamic conflict detections. Static conflict detection is performed offline on the server side and statically checks the entity's policy specification to detect the policy space modality conflicts (i.e., between a policy specification from a system to the user and from a room to the user). The policy space modality conflict may occur here, as the system may permit the user to "start the service", but the room prohibits the user.

We also cache this static conflict detection result for future re-use. When the system detects there is a new user added or there is a user modified his/her policy, our static conflict detection module then updates the cached result. Our dynamic conflict detection further detects the conflict at run time (i.e., a conflict between users). We only detect conflicts between users at run time, as in pervasive systems, the user is always on the move and the movement is unpredictable; hence, we do not know where the user is going to and whom s/he is meeting. Therefore, it would be good to detect this type of conflict dynamically at run time (just when the users are already in the space).

Before further checking for conflicts between users, dynamic conflict detection first detects the type of the service that a user would like to perform. If it is a shared resource service, then the dynamic conflict detection needs to check whether there is a conflict between one user's policy against another user's policy. Checking between users' policies are required for shared services only, as the service is running on the shared machine that allows any legitimate user to control the execution of the service from his/her mobile device. If it is a non-shared resource service, the dynamic conflict detection does not need to further check the conflict between users as the non-shared resource service does not involve other users (only between a user and the room). As we have already detected the conflict between a user and a room statically, the dynamic conflict detection for the non-shared service then just reads from the cached detection file.

Once the checking on the type of the service is done, the dynamic conflict detection then needs to read and process the cached result to find out whether the user is permitted by the system to perform the specified action. If so, then it checks whether the user is permitted to perform the action by the room. If the user is permitted, the system then continues to perform dynamic checking whether there is a conflict between users if the specified action is performed. We use a combination of static and dynamic conflict detections in order to speed up the conflict analysis and processing time. Hence, it will reduce the user wait time. Employing only a single conflict detection strategy i.e., only a static or dynamic conflict detection would slow down the system

performance. In addition, our system also resolves all potential conflicts as soon as they are detected. Resolving the conflict only when it becomes an actual conflict will result in delay in responding to the user's request.

It would be preferably to detect and resolve the conflict statically (offline). However, due to undiscovered all potential conflicts at this time, as some of the conflicts may only occur if a number of entities are in the contexts, run time conflict detection and resolution are also necessary. However, there is still a challenge here in deciding on what types of conflicts should be handled statically or dynamically when considering the aspects of system resources and performance. For example, detecting and resolving all conflicts statically can certainly improve the system performance (as the system has anticipated all potential conflicts with their resolution results). However, detecting and resolving all conflicts statically also has a drawback, in which, it may use up a lot of system resources and may waste the resources, especially if the predicted conflicts never become actual conflicts (the detection and resolution results are never used). This area is still an ongoing work that needs to be further explored in the future.

5.1.3 Performance Results

The framework has given promising results in obtaining a list of policies which are useful to the user, detecting and resolving the conflict both offline and at run time. The evaluation starts from the Web service call to get a user's policy up to resolving the conflict and deciding whether the user is permitted to perform the action on the specified service. The evaluation aspects of our system are described in Figure 2 below.

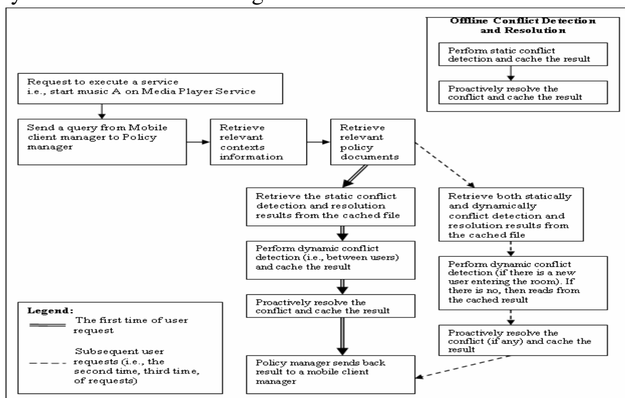


Figure 2: Evaluation aspects

In our evaluation and testing, results were collected for five times of requesting the system to execute the same action with the same service name at the same contexts i.e., a mobile user requests to start a media player service with a particular song name on Saturday, between 12-2PM at B558 room. We measure each of the evaluation aspects above for five times of policy execution, assuming the number of policies in the location are the same throughout the execution i.e., two policies exist in the location – a user's policy and a room's policy. There is also one conflict found between a user and a room, in which a room does not allow a user to perform such a service on Saturday, between 12-2PM at B558 room.

The evaluation results are illustrated in Figure 3 below. These figures were obtained on an iPAQ emulator that is running on the laptop using wireless Wifi network for internet connection. From Figure 3, we can see that the time required to call the Web service: send a query from a client to policy manager, retrieve context information, retrieve relevant and parse policy document, read from the cached results and send back result to the mobile client manager decreases for the 2nd, 3rd, 4th, and 5th times of web service calling. The first call of the Web service takes longer time, as the system needs to compile and download the local host Web service proxy object to the device.

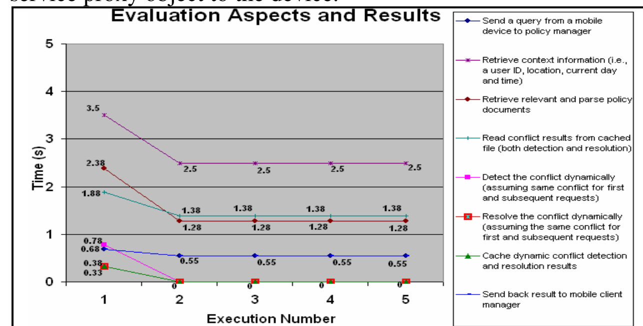


Figure 3: Experimental results

The proxy object allows the Web service to be treated like other .NET classes. The 2nd and subsequent calls to the web service will have much shorter times as it reuses the service proxy object already on the local mobile device. The amount of time required to perform static conflict detection and resolution at compile time is 3.18s (=1.17+0.48+1.05+0.48). Here, the static conflict detection component first detects whether the system gives a user permission to execute the service. If the system permits the user, we then continue checking with the room's policy (i.e., whether the room permits the user to execute the service). Here is the formula to detect and resolve the conflict statically.

$$T_{\text{static conflict analysis(s)}} = T_{\text{detect a conflict statically}} + T_{\text{cache the conflict detection result}} + T_{\text{proactively resolve the conflict}} + T_{\text{cache the conflict resolution result}}$$

If there is a permission given by the room, we continue checking it against other users' policies or a room's obligation (conflict detection at run time). Here, it takes 0.78s to detect a conflict at run time. We found one conflict between a user and a room's obligation. The dynamic conflict detection module here only checks the conflict against a room's obligation, as we only have one user in the location. Checking against a room's obligation is necessary because the room also imposes a certain duty to the user. Hence, we want to ensure that there is no conflict between the user's action and the room's obligation.

This static and dynamic conflict detection results are also cached on the server for future re-use. The second and subsequent policy execution of the same action, service and contexts will just read from the cached file (assuming there is no user moving in or out of a place). Therefore, the dynamic conflict detection time for subsequent policy executions here is zero. Having static conflict detection would help to minimize the user wait time by detecting all potential conflicts between a user and room offline. Detecting such conflict at run time would consume lots of time. Hence, it is recommended to detect it statically, although some of the conflict detection results may not be useful as some of the users may not be in the context as predicted.

As we employ a proactive conflict resolution strategy (resolving conflicts as soon as the system detects them) for both static and dynamic conflict detection, the system takes shorter time to resolve some other detected dynamic conflicts at run time. It takes 0.33s to resolve the dynamic conflict for the first time a service is called. Our system also caches the dynamic conflict resolution result on the mobile device. Hence, the second and subsequent requests of the conflict resolution for the same conflict that has the same action name, target service and contexts would just read from the cached file. In addition, the time it takes to

cache the results (i.e., conflict detection and resolution results) at run time is 0.38s (for the first time requesting the service). As there is no conflict occurring for subsequent requests, there is no result that needs to be cached (0s time to cache results for subsequent requests). Finally, we present a formula to calculate the time required to request to perform an action on a shared or non-shared resource service till the system responds back to the user. This formula is illustrated as follows:

$$T_{\text{user wait time(s)}} = T_{\text{send a query from a mobile client to a policy manager}} + T_{\text{retrieve context information}} + T_{\text{retrieve and parse relevant policy documents}} + T_{\text{read conflict results from a cached file (both detection and resolution)}} + T_{\text{detect a conflict dynamically (if any)}} + T_{\text{resolve a conflict dynamically (if any)}} + T_{\text{cache results (if any)}} + T_{\text{send back result to the mobile client manager}}$$

Based on the formula above, we can conclude that the worst-case scenario for the user wait time is the first time of requesting the service, which takes 10.61s (= 0.68 + 3.5 + 2.38 + 1.88+ 0.78 + 0.33 + 0.38 + 0.68). The 3.5s is the total time to retrieve context information. It takes 3s to get a user's current location using Ekahau location tracking system via a Web service call and 0.5s to retrieve a user identity, current day, and time. The 3s Ekahau delay can be eliminated, if we assume the user is still in the same location (for the first and subsequent requests), and so, the system does not need to re-detect the user's current location.

The best case scenario i.e., the minimum time delay to get a response back from the policy manager is in any execution which is not the first. In such a case, the delay time is 6.26s (=0.55 +2.5 + 1.28 +1.38 + 0 + 0 + 0 + 0.55) – assuming the location context for subsequent requests are still the same. The delay time to detect subsequent requests decrease to 6.26s, because, the Web service calls in subsequent requests, re-use the local proxy object, which has been downloaded and compiled previously and also the subsequent requests do not require to perform dynamic conflict detection and resolution (only read from the cached file) as the conflict is the same as in the first run.

6. Discussion

We observe that each of the proposed conflict detection and resolution techniques has its own advantages and disadvantages, such as: 1) Static conflict detection: it accommodates all potential conflicts that may happen in the future (hence, it will speed up the performance in responding to the user's requests), is simple to develop and relatively easy to maintain. However, this technique only suits if the number of entities in the system is not too many and policy specification and number of entities in the system are relatively static. More entities mean more policy specifications which mean more policies to compare. Allowing entities to modify his/her policy specification at run time or having a new user registered, requires the system to update the static conflict detection result which has been previously computed. Hence, it will use up a lot of resources and may be quite tedious, as it has to re-detect the conflict between all entities in the system. Moreover, some of the conflict detection results may never be used as the entities may never be in the context as they are predicted - hence, the predicted potential conflict never becomes an actual conflict.

2) Reactive based Dynamic Conflict Detection: this technique takes shorter time to detect all potential conflicts in the given context as it only checks the conflict between the requester and number of users in the room. It is also simple to develop and maintain and suits any situation (i.e., static/dynamic policy specifications or entities) as the conflict detection is triggered reactively i.e., when there is a request from a user to perform an action on the service. The main drawbacks of this technique are long delays in detecting and resolving the conflict between entities, as the system only starts to detect and resolve the conflict when there is a request from the user. Moreover, detecting the conflict based on the user's request may not be a good idea as one user may request (click on the action name) more than once in a minute i.e., user A clicks on the start button twice and user B clicks on the stop button three times, hence, the system needs to execute the conflict detection for five times.

3) Proactive based Dynamic Conflict Detection: this technique accommodates all potential conflicts in the given context (hence, reduces the user wait time), use less system resources (memory and CPU processing) compared to the static conflict detection technique, as it only detects conflicts between entities which are in the same context (not all entities in the system). It is also considered easy to develop and suits for any situation with static or dynamic policy specifications or entities. However, the system maintenance can be challenging, as we need to know the best time to update the conflict detection result (when to proactively detect a conflict) i.e., when the system detects that there is a new user moves in or out of the space, frequently every 5 seconds, or when the system detects there are more than certain number of users in the space such as more than two users in the room.

4) A combination of Reactive and Proactive based Dynamic Conflict Detection: this is an ideal technique among all other conflict detection techniques. It accommodates all potential conflicts in the given space by using a combination of reactive and proactive techniques. It can be proactive in some situations and reactive in others, and so, can further reduce the system resources (memory and CPU processing). It also suits in any situation (with static or dynamic policy specification, entities, services and contexts). This technique is also easy to implement. The only issue here is we need to decide when and under which situation a proactive or reactive behaviour should be performed.

5) Predictive based Dynamic Conflict Detection: this technique is much more complex to develop and maintain and does not accommodate the user's unpredictability.

In addition, we found that the potential conflicts which are detected at run time by using a reactive technique have higher possibility to become actual conflicts compared to other techniques (i.e., a proactive or predictive technique). This is mainly because in reactive technique, the detection is only performed when there is a request from a user and the detection is looking for conflicts only for the current day, time and location (hence, if there is a conflict found, the contexts for the conflict to occur must have been met). On the other hand, a proactive technique proactively detects all potential conflicts between users although the contexts for the conflict to occur have not been met.

Moreover, for conflict resolution, the best technique is to have a proactive conflict resolution strategy that immediately resolves the conflicts as soon as the system detects them. This technique anticipates all potential conflicts that may happen between entities in the future. Hence, it improves the system performance and certainly minimizes the user wait time. However, some of the conflict resolution results may not be useful as some of the detected potential conflicts may never happen at run time.

7. Related Work

This section provides a brief overview about the research work that has been done to date that also concentrates on exploring different strategies used to detect and resolve conflicts in policy systems. Some earlier policy work in pervasive systems are Rei [3], Spatial Policies [4] and Policy for Agent Mobility work [8]. In addition, only few work done to date explores different strategies of policy conflict detection and resolution. A notable

project is a work done in [5,6,7] that explores different techniques used for conflict detection and resolution in enterprise and management policy based systems. Our conflict detection and resolution techniques to some extent have similar philosophy to this project. The only difference is the target environment, we focus on pervasive systems which have services, entities, contexts, mobile devices, workstations and spaces.

As our system is designed for pervasive computing environments, in which users are always on the move and often require immediate response from the system of their requests, the sources and types of conflicts found in our system are also different from the one in [5,6,7]. This then leads to some differences in designing and implementing the conflict detection and resolution techniques. For example, we have conflicts on permissions, obligations and prohibitions between mobile users, as well as between a mobile user and the space. In contrast to [5,6,7], they do not take into account the mobility of users and the notions of services, and so, the conflicts found in the system are mostly between non-mobile users who are trying to access system or a user's resources information. In addition, our pervasive system tends to focus more on the system performance that aims to deliver the service, detecting and resolving conflicts in minimum amount of time.

8. Conclusions and Future Work

This paper has presented a design, model and architecture of a policy based framework in pervasive environments. We have proposed several techniques or strategies for conflict detection and resolution. We also have implemented and tested our policy system with some of conflict detection and resolution strategies on the mobile emulator that runs on an 802.11b wireless network. While implementing some of the conflict detection and resolution strategies, we discovered that each of the proposed strategies both for conflict detection and resolution offers some advantages and disadvantages. The suitability of each strategy is dependent on the system situations (i.e., number of entities, physical rooms, contexts, types of services and target services that the system employs), the system goals (i.e., it aims for high performance, so requires a comprehensive and more complex conflict detection (i.e., a predictive model) and resolution modules), and types of conflicts that the system attempts to detect or resolve (i.e., we tend to detect all policy space modality conflicts statically).

Moreover, we have experienced that using a combination of static and dynamic conflict detection helps to improve the system performance (minimize users wait time), rather than only using a single detection technique (i.e., static only or dynamic only). We also found that resolving all potential conflicts (possible or definite conflicts), as soon as they are detected, would certainly reduce the delay in responding to the user's request, and so improve system performance.

A number of aspects of future work that need to be further analysed, explored and developed are: a) Continue working on proactive and predictive conflict detection strategies. b) Allowing users to modify their policy specifications dynamically at run time. c) Apply our policy concepts (i.e., designs, conflict and detection and resolution strategies) in different pervasive environments or domains i.e., a museum gallery, shopping mall, airport. d) Monitor the probability of potential conflict occurrence e) study the nature and complexity of each conflict found in pervasive systems, also finding out how much of memory, CPU cycles required to detect and resolve conflicts both statically and at run time.

9. References

[1] Weiser, M., "The Computer for the 21st Century", *Scientific American*, 9 1991.

[2] Chen, G. and Kotz, D. (2000), "A Survey of Context-Aware Mobile Computing Research", Dartmouth Computer Science, *Technical Report TR2000-381*.

[3] Kagal, L., Finin, T. and Joshi, A., "A Policy Language for a Pervasive Computing Environment, *Proc. of IEEE 4th International Workshop on Policies for Distributed Systems and Networks*, Italy, June 2003.

[4] Scott, D., Beresford, A. and Mycroft, A., "Spatial Policies for Sentient Mobile Applications", *Proc. of IEEE 4th International Workshop on Policies for Distributed Systems and Networks*, Italy, June 2003.

[5] Dunlop, N., Indulska, J. and Raymond, K., "Dynamic Policy Model for Large Evolving Enterprises", *Proc. 5th IEEE Enterprise Distributed Object Computing Conference*, Seattle, Sept 2001.

[6] Dunlop, N., Indulska, J. and Raymond, K., "Dynamic Conflict Detection in Policy-Based Management Systems", *Proc. 6th IEEE Enterprise Distributed Object Computing Conference*, Lausanne, Sept 2002

[7] Dunlop, N., Indulska, J. and Raymond, K., "Methods for Conflict Resolution in Policy-Based Management Systems", *Proc. 7th IEEE International Enterprise Distributed Object Computing Conference*, Brisbane, Sept 2003, pp 98-109.

[8] Montanari, R., Lupu, E. and Stefanelli, C., "Policy-based dynamic reconfiguration of mobile-code applications", *IEEE Magazine*, July 2004.

[9] Syukur, E., Cooney, D., Loke, S.W. and Stanski, P., "Hanging Services: An Investigation of Context-Sensitivity and Mobile Code for Localised Services", *Proc. of the IEEE International Conference on Mobile Data Management*, USA, Jan 2004, pp.62-73.

[10] Syukur, E., Loke, S.W. and Stanski, P., "The Mobile Hanging Services Framework for Context-Aware Applications: the Case of Context Aware VNC", *Proc. WIS Workshop*, Portugal, April 2004.

[11] Syukur, E., Loke, S.W. and Stanski, P., "A Policy based framework for Context Aware Ubiquitous Services", Proc. of the Embedded Ubiquitous Computing Conference, Japan, August 2004, LNCS, vol. 3207, Springer-Verlag, pp.346-355, 2004.

[12] Mally, E. "The Basic Laws of Ought: Elements of the Logic of Willing", 1926.

Policy Conformance in the Corporate Blog Space

R.McArthur
Distributed Systems Technology Centre
Brisbane,Australia
mcarthur@dstc.edu.au

P.D.Bruza
Distributed Systems Technology Centre
Brisbane,Australia
bruza@dstc.edu.au

D.Song
Distributed Systems Technology Centre
Brisbane,Australia
dsong@dstc.edu.au

ABSTRACT
This paper describes part of a solution to the interpretation of human-readable policy documents into semi-automatic conformance checking. Using a socio-cognitively motivated representation of shared knowledge, and applying appropriate inference mechanisms from a normative perspective, a mechanism to automatically detect potentially non-conforming blog entries is detailed. Candidate non-conforming blog entries are flagged for a human to make a judgement on whether they should be published. Analysis of data from a public corporate blog is analysed and results suggest the methodology has merit.

Categories and Subject Descriptors
I.7. [Document and Text Processing], H.4. [Information Systems Applications]

General Terms
Algorithms, Management, Experimentation, Human Factors, Theory

Keywords
Semantic space, policy conformance, blog, knowledge management

1. INTRODUCTION
Managers of organisations have long tried to control what is the official word of the body versus what an employee personally has presented. The (generally) open nature of the WWW has meant an increasing desire for control by some managers, while others have realised the need for a different way of working – for example, the open source software movement. Management in this new way isn't laissez faire, it respects the possibilities of more openness but still has control, often through loosely worded policy rather than the heavy legal jargon. This approach can be characterised as being more carrot than stick.

Sun Microsystems has recently created a standard blog space[1] available to all employees, visible to the world. From Tim Bray's website, on the 6 June 2004[2]:

> *It's been running for some time, and it's stable enough now to talk about in public: blogs.sun.com is a space that anyone at Sun can use to write about whatever they want. The people there now are early adopters; there's an internal email going out to the whole company Monday officially reinforcing that blogging policy, encouraging everyone to write, and pointing them at blogs.sun.com.*

The Sun Policy on Public Discourse[3] is written for people. It encourages blogging stating "*As of now, you are encouraged to tell the world about your work, without asking permission first (but please do read and follow the advice in this note).*" Because of the implications of the policy, and the particularities and importance of wording and intentionality, we have reproduced it in entirety in Appendix A. Appropriate parts are quoted in the following sections.

This paper describes part of a solution to the interpretation of human-readable policy documents into semi-automatic conformance checking. Using a socio-cognitively motivated representation of shared knowledge, and applying appropriate inference mechanisms, a mechanism to automatically detect potentially non-conforming blog entries is detailed. Candidate non-conforming blog entries are flagged for a human to make a judgement on whether they should be published. Figure 1 shows the workflow.

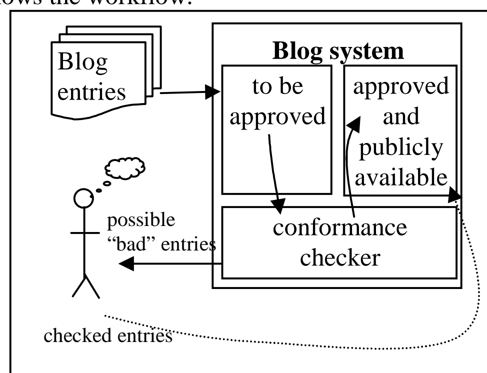


Figure 1: Workflow

The benefits are a significant lessening of work for humans to evaluate each blog entry. Instead, only a subset is required to be vetted by a person.

The remainder of this paper describes the approach taken in more detail, starting with the notion of normative disconformance and applying semantic spaces to blog data,

[1] http://blogs.sun.com/

[2] http://www.tbray.org/ongoing/When/200x/2004/06/06/BSC

[3] http://www.tbray.org/ongoing/When/200x/2004/05/02/Policy (note this was so over the time of this study but may have changed)

thence to experimental results of examining Sun's blog data with respect to one element of its policy.

2. OPERATIONALISING NORMATIVE DISCONFORMANCE

Let N be a normative model comprising principles (or standards) $S_1,\ldots S_n$. Let B be a piece of augmentative behaviour. Let B disconform with principle S_i. If S_i is genuinely normative then B is a mistake (at a minimum) [1].

We believe that Sun's problem with checking compliance of blog content can be considered conceptually from a normative perspective. With respect to Sun, read "mistake" as a breach of policy.

Implementing this requires firstly a computational variant of the normative model N, as well as an (semi-) automated procedure for determining (or estimating) disconformance.

Cognitive science distinguishes between three models of cognitive performance:

1. the normative model N that sets standards of rational performance, irrespective of the (computational) cost of compliance;
2. the prescriptive model P which attenuates the standards to make them executable; and
3. the descriptive model D which is a law governed account of actual performance.

Sun's policy can be considered as a high level prescriptive model. It is assumed that the human moderators apply quite some background knowledge B in order to determine or surmise disconformance.

It seems unlikely that a sufficiently large training set of disconforming blog entries can be acquired, therefore a supervised learning approach is almost certainly not appropriate for detecting disconformance. We take a different approach. Certain words, or phrases, in the prescriptive model flag concepts that are key to a particular standard. These can be considered as pseudo-queries with which blog entries can be retrieved and ranked.

It is well known from the field of information retrieval that short queries are typically imprecise descriptions of the associated information need. More effective retrieval can be obtained via automatic query expansion the goal of which is to "guess" related terms to the query at hand. The word "guess" is used deliberately here as the system is ignorant of the actual information need.

Considered in this light, query expansion is a manifestation of abduction. The goal is to abduce related terms to the pseudo-query which are relevant to the intention behind the pseudo-query. If the query expansion mechanism abduces poorly, retrieval precision will decline, a consequence of which is that disconformant blog entries will not be highly ranked in the retrieval ranking. In this article, we will employ a query expansion mechanism which abduces expansion terms by computing the information flow between concepts in a high dimensional semantic space. Query expansion experiments carried out in a traditional information retrieval setting have shown information flow to be promising, particularly for short queries [2].

3. SEMANTIC SPACES

Nonaka and Takeuchi [3] produced an important and viable knowledge creation system in 1995. We have instantiated their notion of an externalisation mode in which tacit knowledge is made explicit and "*The semantic aspect of information* [as against the syntactic] *is more important for knowledge creation, as it focuses on conveyed meaning*.", with Freyd's [4] work on *shareability* which posited

> "*a dimensional structure for representing knowledge is efficient for communicating meaning between individuals. That is, a small dimensional structure with a small number of values on each dimension is argued to be especially shareable, which might explain why such structures are observed*." (Pp.198-9)

The combination of the explicit-tacit knowledge mode with the dimensional representation is further strengthened by Gärdenfors' three level socio-cognitive model of cognition [5]. He argues that meanings of words come from conceptual structures in people's heads – they emerge from the conceptual structures harboured by individual cognition together with the linguistic power structure within the community. Of his three levels of representation, symbolic, conceptual and associationist (sub-conceptual), it is the middle, conceptual, level that is of relevance for this paper.

People write blog entries to communicate. In all communication, there are both explicit and tacit parts to the message. Ducheneaut and Bellotti [6] found that:

> *Persistent talk* [in email] *provides the context for the solitary activity of viewing the content to which it relates…However, during our interviews we, in fact, saw many more examples of imprecise references that were immediately understood than long, drawn-out, explicit and literal descriptions or references.*" and
> "*…email conversations are grounded in sufficient mutual understanding to allow very brief, sketchy and implicit references to succeed without posing significant problems in interpretation.*"

Compliance analysis of blog entries with respect to any policy, whether perfectly formed or not, is always dependent on the language used in the entry. Explicit mention of keywords is unlikely to uncover the range of candidate non-compliant entries that make up blog data in the "real world", and will most likely result in poor recall and precision (concepts from information retrieval).

Our previous work [7,8,9] has shown the efficacy of a socio-cognitively based dimensional structure-a semantic space-as a knowledge representation framework. Although there are a number of algorithms for populating such a space, we will briefly describe one, HAL, below. We will then discuss ways of using the semantic space in the context of compliance and blog data.

3.1 Creating the representation - HAL

Hyperspace Analogue to Language (HAL) is a model and technique to populate a semantic space [10,11]. HAL produces vectorial representations of words in a high dimensional space that seem to correlate with the equivalent human representations [12]. For example, word associations computed on the basis of HAL vectors seem to mimic human word association judgments. HAL is "*a model that acquires representations of meaning by capitalizing on large-scale co-occurrence information inherent in the input stream of language*".

Words from communication–blogs–are represented in dimension structures through HAL. The space comprises high dimensional vector representations for each term in the vocabulary. Briefly, given an n-word vocabulary, the HAL space is a nxn matrix constructed by moving a window of length l over the corpus by one word increments ignoring punctuation, sentence and paragraph boundaries. All words within the window are considered as co-occurring with each other with strengths inversely proportional to the distance between them. After traversing the communication corpus, an accumulated co-occurrence matrix for all the words in a target vocabulary is produced: the semantic space.
More formally, a concept[4] c_i is a vector representation: $c_i = \langle w_{c_i p_1}, w_{c_i p_2}, \ldots w_{c_i p_n} \rangle$ where $p_1, p_2, \ldots, p_n$ are called dimensions of c_i, n is the dimensionality of the HAL space, and $w_{c_i p_i}$ denotes the weight of p_i in vector of c_i. A dimension is termed a property if its weight is greater than zero. A property p_i of a concept c_i is a termed quality property iff $w_{c_i p_i} > \partial$, where ∂ is a non-zero threshold value. Let $QP(c)$ denote the set of quality properties of concept c.

3.2 Combining concepts

Concept combination is important as combinations of words in may represent a single underlying concept, for example, *Sun's share price*. An important intuition in concept combination is that one concept can dominate the other. For example, the term "Sun" can be considered to dominate the term "price" because it carries more of the information in the phrase. Given two concepts $c_1 = \langle w_{c_1 p_1}, w_{c_1 p_2}, \ldots w_{c_1 p_n} \rangle$ & $c_2 = \langle w_{c_2 p_1}, w_{c_2 p_2}, \ldots w_{c_2 p_n} \rangle$, the resulting combined concept is denoted $c_1 \oplus c_2$. The following concept combination heuristic is essentially a restricted form of vector addition whereby quality properties shared by both concepts are emphasized, the weights of the properties in the dominant concept are re-scaled higher, and the resulting vector from the combination heuristic is normalized to smooth out variations due to differing number of contexts the respective concepts appear in.

Step 1: Re-weight c_1 and c_2 in order to assign higher weights to the properties in c_1.

$$w_{c_1 p_i} = \ell_1 + \frac{\ell_1 * w_{c_1 p_i}}{\underset{k}{Max}(w_{c_1 p_k})} \quad \text{and} \quad w_{c_2 p_i} = \ell_2 + \frac{\ell_2 * w_{c_2 p_i}}{\underset{k}{Max}(w_{c_2 p_k})}$$

$$\ell_1, \ \ell_2 \in (0.0, \ 1.0) \text{ and } \ell_1 > \ell_2$$

For example, if $\ell_1 = 0.5$ and $\ell_2 = 0.4$, then property weights of c_1 are transferred to interval [0.5, 1.0] and property weights of c_2 are transferred to interval [0.4, 0.8], thus scaling the dimensions of the dominant concept higher.

Step 2: Strengthen the weights of properties appearing in both c_1 and c_2 via a multiplier α; the resultant highly weighted dimensions constitute significant properties in the resultant combination.

$$\forall (p_i \in QP(c_1) \wedge p_i \in QP(c_2)) \ \mid \ w_{c_1 p_i} = \alpha * w_{c_1 p_i},$$
$$w_{c_2 p_i} = \alpha * w_{c_2 p_i}, \text{ where } \alpha > 1.0$$

Step 3: Compute property weights in the composition $c_1 \oplus c_2$:

$$w_{(c_1 \oplus c_2) p_i} = w_{c_1 p_i} + w_{c_2 p_i}, 1 \le i \le n$$

Step 4: Normalize the vector $c_1 \oplus c_2$. The resultant vector can then be considered as a new concept, which, in turn, can be composed to other concepts by applying the same heuristic.

In order to deploy the information flow model in an experimental setting, the pseudo-queries have to analysed for concept combinations. In particular, the question of which concept dominates which other concept(s) needs to be resolved. As there seems to be no reliable theory to determine dominance, a heuristic approach is taken in which dominance is determined by multiplying the query term frequency (*qtf*) by the inverse document frequency (*idf*) value of the query term. More specifically, query terms can re ranked according to *qtf**idf*. Assume such a ranking of query terms: $q_1, \ldots, q_m$ ($m > 1$). Terms q_1 and q_2 can be combined using the concept combination heuristic described above resulting in the combined concept $q_1 \oplus q_2$, whereby q_1 dominates q_2 (as it is higher in the ranking). For this combined concept, its degree of dominance is the average of the respective *qtf**idf* scores of q_1 and q_2. The process recurses down the ranking resulting in the composed query "concept" $((..(q_1 \oplus q_2) \oplus q_3) \oplus \ldots) \oplus q_m)$. This denotes a single vector from which query models can be derived. If there is a single query term ($m = 1$), it's corresponding normalized HAL vector is used for query model derivation.

As it is important to weight pseudo-query terms highly, the weights of query terms which appeared in the initial query were boosted in the resulting query model by adding 1.0 to their score. Due to the way HAL vectors are constructed, it is possible that an initial query term will not be represented in the resulting query model. In such cases,

[4] The term "concept" is used somewhat loosely; it can be envisaged as "term" in the traditional IR sense

the query term was added with a weight of 1.0. Pilot experiments show that the boosting heuristic performs better than the use of only query models without boosting.

3.2 Using the semantic space – information flow

Barwise & Seligman [13] have proposed an account of information flow that provides a theoretical basis for establishing informational inferences between concepts. For example,

share, price |- SUN

illustrates that the concept "SUN" is carried informationally by the combination of the concepts "share" and "price". Said otherwise, "SUN" *flows* informationally from "share" and "price". Such information flows are determined by an underlying information state space. A HAL vector can be considered to represent the information "state" of a particular concept (or combination of concepts) with respect to a given corpus of text. The degree of information flow between "satellites" and the combination of "space " and "program" is directly related to the degree of inclusion between the respective information states represented by HAL vectors. Total inclusion leads to maximum information flow. Inclusion is a relation ⊆ over the concept space.

Definition 1 (HAL-based information flow)

$$i_1, \ldots, i_n \mid\!- j \text{ iff degree}(\oplus c_i \subseteq c_j) > \lambda$$

where c_i denotes the conceptual representation of token i, and λ is a threshold value. (For ease of exposition, $\oplus c_i$ will be referred to as c_i because combinations of concepts are also concepts).

Note that information flow shows truly inferential character, i.e., concept j is not necessarily a dimension of the $\oplus c_i$. The degree of inclusion is computed in terms of the ratio of intersecting quality properties of c_i and c_j to the number of quality properties in the source c_i:

$$\text{degree}(c_i \subseteq c_j) = \frac{\sum_{p_l \in (QP(c_i) \wedge QP(c_j))} w_{c_i p_l}}{\sum_{p_k \in QP(c_i)} w_{c_i p_k}}$$

In terms of the experiments reported below, the set of quality properties $QP_i(c_i)$ in the source HAL vector c_i is defined to be all dimensions with non-zero weight (i.e., ∂ > 0). The set of quality properties $Qj_i(c_j)$ in the target HAL vector c_j is defined to be all dimensions greater than the average dimensional weight within c_j**.** These definitions for determining the quality properties in the source concept c_i and target concept c_j were determined via pilot studies in information flow computation.

2.3 Deriving query models via information flow

Given the pseudo-query $Q=(q_1,...,q_m)$ drawn manually from a standard S in the prescriptive model P, a query model can be derived from Q in the following way:

- Compute degree($\oplus c_i \subseteq c_t$) for every term t in the vocabulary, where $\oplus c_i$ represents the conceptual combination of the HAL vectors of the individual query terms $q_i, 1 \le i \le m$ and c_t represents the HAL vector for term t.
- The query model $Q' = \langle t_1 : f_1, \ldots, t_k : f_k \rangle$ comprises the top k information flows

Observe that the weight f_i associated with the term t_i in the query model is not probabilistically motivated, but denotes the degree to which we can infer t_i from Q in terms of underlying HAL space.

4. BLOG DATA

Blog data, as input to computational analysis as distinct from human comprehension, is inherently "dirty": it can consist of anything from a URL, presumably as aid to the memory of the author and often with a longer title explaining something, or it can be a long-winded polemic in the first person. Nardi et al [17] found that people blog for (at least) five reasons – documenting one's life, providing commentary and opinions, expressing deeply felt emotions, articulating ideas through writing, and forming and maintaining community forums. While humans find it relatively easy to navigate the morass, find interesting elements and determine the worth of data, comparatively this is almost impossible for current computer systems.

A vital element is a filter to identify "interesting" blog entries which would be used to populate the semantic space(s). "Interesting" is determined by the particular person doing the searching, or the particular problem. For example, if the question is one of compliance—is a particular blog entry compliant with Sun's policies—the filter would provide very different entries than if an individual were interested in a particular Sun product.

It is feasible to produce filters which could identify the five+ (non-exclusive) motivations as only some of these are relevant to policy conformance checking. It is also important to filter the difference between a wilful breaking of policy and an inadvertent one.

Many such situational-based filters are possible. The focus of these experiments was to apply one such filter to the blog entries. Note that for checking of blog entry compliance, the filter may be enacted *prior* to blog entry publication (as in Figure 1) or afterwards. While the method we describe could be used in both ways, we envisage that human invigilators would prefer to peruse candidate entries at certain times during the day rather than being interrupted for each possibility. This is of course offset by the desire to preserve the currency of the entries.

4.1 Experimental data

We examined all entries from the Sun blog RSS feed from 19 July to 9 August (22 days) 2004. There were 1507 RSS entries at an average of 68.5 per day (2.8 per hour); the minimum was 17 entries on July 31st. However, on two days-the 26th and 27th of July-there were 404 and 140 entries respectively. This was due to discussion about a new product about to be released ([16] have some further insights into these phenomena). Figure 1 charts the entries over time.

The size of the vocabulary (stop words removed) was 24,841 words. As we only examined entries from the RSS feed, we were not able to account for comments submitted to existing blog entries, and other associated text that did not appear in the RSS. Where available, this could augment the analysis.

It is important to note that no set of disconforming data was provided. We do not have details of any blog entries that were filtered prior to publication, and cannot guarantee that those that we worked with are all still extant. All experimental work was conducted on blog entries that were publicly available at the time. We do not know if Sun would consider any particular entry we have discussed disconformant. In this way, although our analysis lets us work unfettered by internal prejudice, we may miss nuances that an internal assessor would not.

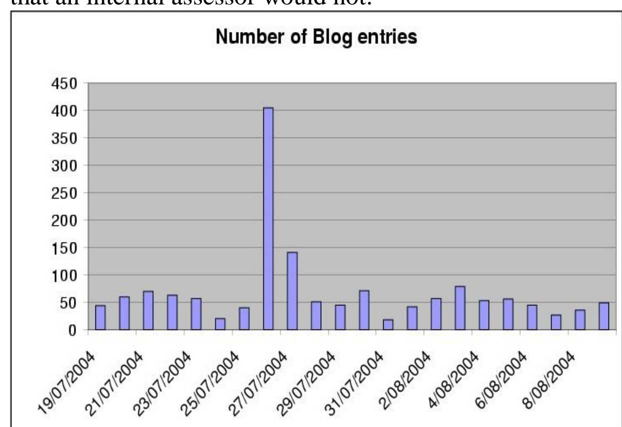


Figure 2: Number of blog entries over experiment time

5. ANALYSIS

To provide a flavour of the data in the form of semantic spaces, two tables show the results of computations creating semantic spaces over the entire collection: table 1 shows the words with the largest number of dimensions (ie. words used in many contexts); table 2 shows the "largest" explicit dimensions of the "sun" vector.

Table 1: "Sun" vector top dimensions

sun	1008	back	387
solaris	662	entry	319
new	651	things	314
java	651	great	313
open	556	dtrace	300
good	516	software	295
work	474	blog	277
people	461	code	266
system	420	linux	265
don	415	...	...
source	397		

Table 2: Top "sun" vector dimensions (cols 1-2) and nearest concepts (cosine; cols 3-4); 4511 dimensions, $\bar{x}$: 19.2, σ^2: 54.6

sun	2266.00	java	0.84
java	1443.00	working	0.75
open	819.00	microsystems	0.72
solaris	817.00	workstation	0.72
system	565.00	product	0.72
source	529.00	work	0.71
new	459.00	lot	0.71
work	456.00	community	0.70
working	438.00	customers	0.70
people	385.00	system	0.70
company	379.00	product	0.70
customers	306.00	people	0.69
server	299.00	developer	0.69
desktop	292.00	company	0.69
blog	278.00	ibm	0.65
software	277.00	desktop	0.65
good	275.00	software	0.65
product	275.00	employees	0.65
ray	273.00	developers	0.64
microsystems	266.00	worked	0.64
lot	262.00	new	0.64
community	252.00	reason	0.64
linux	249.00	part	0.64
cluster	243.00	ray	0.63
employees	240.00	vendor	0.63
things	232.00	customer	0.63
products	225.00	cluster	0.62
business	223.00	hardware	0.61
support	219.00	new	0.61
ibm	213.00	don	0.61
workstation	213.00	companies	0.61
...	...	...	...

5.1 Information flow based query expansion

The "Financial rules" section in the policy (Appendix A) states:

There are all sorts of laws about what we can and can't talk about. Talking about revenue, future product ship dates, road maps, or our share price is apt to get you, or the company, or both, into legal trouble.

The challenge is to mimic human's ability to interpret the above standard while considering a certain blog entry.

A semantic space H_B can be constructed from the blog corpus B using the Hyperspace Analogue to Language model. The goal is to provide a semantic representation $\sigma(c)$ for concept c which will be used as a "query" to match incoming blog entries. If the match score is above a certain threshold it can be flagged for human perusal.

In our previous work, encouraging improvements in retrieval precision were produced by information flow based query expansion [2]. For the purposes of illustration, we focus on the financial area by characterizing it with the concept "share price", which is a noun phrase. Our concept combination heuristic produces a semantic representation of the compound using the individual semantic representations σ(share) and σ(price). (See [15] for more details of this heuristic). Each of the two pseudo-queries was expanded using information flow. Table 3 shows the top information flows from the concept "share price". The top 65 information flows (empirically determined) were used to expand the pseudo-query. The resulting expanded query was matched against blog entries which were ranked on decreasing order of retrieval status score. In order to facilitate matching each blog was indexed using the BM-25 term weighting score[5], with stop words removed. Both query and document vectors were normalized to unit length. Matching was realized by the dot product of the respective vectors and the top five ranked blog entries were chosen. This threshold was chosen as we assume that human judges will not want to manually peruse rankings much longer than this.

Table 3: Information flows from the concept "share price"

Flow	*Value*
price	0.77
share	0.68
sun	0.59
good	0.51
back	0.44
don	0.44
software	0.53
...	...

5.2 Experimental Results

Due to the small number of pseudo-queries it is not warranted to present a precision-recall analysis. A much larger experimental setting would be required.

Discussion will proceed based on anecdotal evidence. The following document (next column) was ranked second with respect to the pseudo-query "share price" and is the most interesting of the top five.

[5] BM-25 represents state-of-the-art in term weighting, e.g. [14]

5.3 Discussion

The retrieval of the above document demonstrates the potential of information flow query expansion. Note how the phrase "share price" does *not* appear in this blog entry, but is clearly about a strongly related concept (stock option). This example also shows how information flow based query expansion facilitates the promotion of potentially disconformant blogs in the retrieval ranking when there is little or no term overlap between the pseudo-query and blog entry. In order to place this claim in perspective, we expanded the pseudo-query "share price" with a highly respected probabilistic retrieval model - the BM25 model [14], and a query expansion technique - the Robertson's Term Selection Value (TSV) [14]. Both techniques were unable to rank the above document in the top five.

<CONTEXT ID="//blogs.sun.com/roller/page/pdiamond/20040624#stock options why not expense">
</CONTEXT>

Stock Options - Why not expense them?

Just came from the rally in Palo Alto to oppose FASB ruling that stock options should be expensed. For those who do NOT have access to stock options, the answer seems pretty simple:

> "These people are making lots of money off stock options, taking advantage of opportunities we don't have and inaccurately reflecting their expense on their companies' bottom lines. Of course they should be counted as an expense when they are granted"

I'm sure a lot of this is also reflective of the abuses which have been widely reported, of CxOs making million$ while their companies went down the tubes.

Now here's another view of reality - for those of us who have

- made some money (thank you, Netscape) and
- not made any yet (I am still optimistic, Sun),

it also seems pretty obvious.

All those options we have been granted which we do NOT exercise, because they are "underwater", e.g.:

- Netscape /AOL options at $75 when the stock price was $20,
- current Sun options at $12 (and I know many people with options well above that price) with the stock a little over $4, are irrelevant to anyone. They are no more expense to the companies which granted them than they are profit to the employees who are not exercising them.

If and when they are exercised, then let's talk about how the companies should expense the benefit received by the employees. I admit to being ignorant as to how this is handled today. This seems to be a much more relevant issue than trying to assess some current value on some theoretical future benefit, which in many cases will either not happen, or will occur at a totally unpredictable level.

June 24, 2004 04:07 PM PDT Permalink

5.4 Temporal topics of Pseudo-queries

Tracking the temporal profile of a pseudo-query over time can help visualize blog activity around a topic relevant to detecting disconformance. Figure 3 depicts the probability of the pseudo-query "share price" over time. The underlying theory combines information flow based query expansion [2] with document language models. The probabilities of queries were calculated from top ten documents retrieved by the information flow model and then smoothed using a back-off model based on collection statistics. The spikes in the figure depict localized

probabilities of the topic which can be used to localize activity around a pseudo-query. Such localities may warrant closer inspection for disconformance.

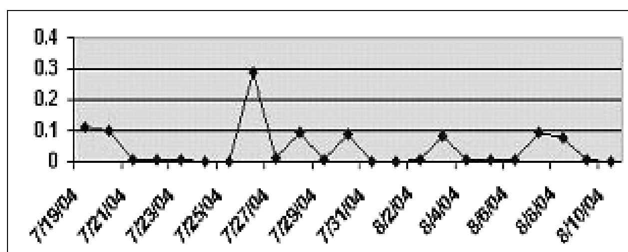


Figure 3: Temporal profile for topic "share price"

5.5 Optimal projections

The approach here is to assume that blogs disconforming to a standard S_i will cluster around a given axis, or somehow project differently into the semantic space than conforming blog entries. Dimensional reduction approaches may gain some purchase, for example independent component analysis or projection pursuit. Further investigation is required.

4. CONCLUSION

This article deals with the problem of providing automated support for the detection of disconformant blog entries with respect to a publishing policy. The problem is considered from a normative perspective. The detection of disconformant blog entries has an abductive character. Automated support for detecting disconformant blogs is realized via query expansion, the goal of which is to abduce salient terms in relation to pseudo-query representations of publishing standards. The expanded pseudo-queries are computed vie information flows through a high dimensional semantic space derived from the blog corpus.

Anecdotal evidence suggests that information flow based query expansion may be promising in regard to retrieving disconformant blog entries, which can then be manually examined for a final judgment. The case study reported in this paper suggests that the problem of furnishing (semi-) automated support for the detection of disconformat blog entries to be a challenging one requiring further investigation using non-supervised approaches.

ACKNOWLEDGMENTS

The work reported in this paper has been funded in part by the Co-operative Research Centre for Enterprise Distributed Systems Technology (DSTC) through the Australian Federal Government's CRC Programme (Department of Education, Science, and Training). The research was funded by a Discovery grant from the Australian Research Council. Thanks to the members of the Research Management Team and participants.

REFERENCES

[1] Gabbay, D. and Woods, J. (2003): Normative models of rational agency: the theoretical disutility of certain approaches. *Logic Journal of the IGPL.* 11:597-613

[2] Bruza, P.D and Song, D. (2002): Inferring query models by computing information flow. In *Proceedings of the 11th International Conference on Information and Knowledge Management (CIKM 2002)* ACM Press, pp.260-269.

[3] Nonaka, I. and Takeuchi, H. (1995): *The Knowledge-Creating Company*, OUP, New York

[4] Freyd, J. (1983). Shareability: the social psychology of epistemology. *Cognitive Science*.**7**:191-210

[5] Gärdenfors, P. (2000): *Conceptual Spaces: the Geometry of Thought.* MIT Press, London, 2000

[6] Ducheneaut, N. and Bellotti, V. (2003). 'Ceci n'est pas un objet? Talking about things in email. *Journal of Human-Computer Interaction (special issue)* **18**(1-2): 85-110.

[7] McArthur, R. and P. Bruza (2003). Dimensional Representations of Knowledge in Online Community. In *Chance Discovery.* Y.Ohsawa & P.McBurney, Springer**:** 98-112

[8] McArthur, R. and P. Bruza (2003). Discovery of Tacit Knowledge and Topical Ebbs and Flows within the Utterances of Online Community. In *Chance Discovery.* Y. Ohsawa and P. McBurney, Springer**:** 115-131.

[9] McArthur, R. and P. Bruza (2003). Discovery of Implicit and Explicit Connections between People using Email Utterance. In *Eighth European Conference on Computer-Supported Cooperative Work (ECSCW)*, Helsinki, Finland, Kluwer.

[10] Burgess, C., Livesay, K. and Lund, K. (1998): Explorations in context space: words, sentences, discourse. *Discourse Processes*, v25, pp.211-257

[11] Burgess, C. and K. Lund (1997b). Representing Abstract Words and Emotional Connotation in a High-Dimensional Memory Space. *Cognitive Science*.

[12] Lund, K., C. Burgess and R. A. Atchley (1995). Semantic and Associative Priming in High-Dimensional Semantic Space. *Cognitive Science*, Erlbaum Publishers, Hillsdale, N.J.

[13] Barwise, J. and Seligman, J. (1997) *Information Flow*. Cambridge University Press.

[14] Robertson, S., Walker, S., Jones, S., Hancock-Beaulieu, M. and Gatford, M. (1995) Okapi at TREC-3. In *Proceedings of TREC-3* 1995. Available at trec.nist.gov.

[15] Song,D. and Bruza, P. (2003) Towards context sensitive information inference. *Journal of the American Society for Information Science and Technology*, 54(3):321-334.

[16] Kumar, R., Novak, J., Raghavan, P. and Tomkins, A. (2004): Structure and evolution of blogspace. *Communications of the ACM*, 47(12) pp35-39

[17] Nardi, B., Shiano, D., Gumbrecht, M. and Swartz, L. (2004) Why we blog? *Communications of the ACM.* v47(12)

APPENDIX A: SUN'S BLOGGING POLICY

Advice By speaking directly to the world, without benefit of management approval, we are accepting higher risks in the interest of higher rewards. We don't want to micro-manage, but here is some advice.

It's a Two-Way Street The real goal isn't to get everyone at Sun blogging, it's to become part of the industry conversation. So, whether or not you're going to write, and especially if you are, look around and do some reading, so you learn where the conversation is and what people are saying.

If you start writing, remember the Web is all about links; when you see something interesting and relevant, link to it; you'll be doing your readers a service, and you'll also generate links back to you; a win-win.

Don't Tell Secrets Common sense at work here; it's perfectly OK to talk about your work and have a dialog with the community, but it's not OK to publish the recipe for one of our secret sauces. There's an official policy on protecting Sun's proprietary and confidential information, but there are still going to be judgment calls.

If the judgment call is tough—on secrets or one of the other issues discussed here—it's never a bad idea to get management sign-off before you publish.

Be Interesting Writing is hard work. There's no point doing it if people don't read it. Fortunately, if you're writing about a product that a lot of people are using, or are waiting for, and you know what you're talking about, you're probably going to be interesting. And because of the magic of hyperlinking and the Web, if you're interesting, you're going to be popular, at least among the people who understand your specialty.

Another way to be interesting is to expose your personality; almost all of the successful bloggers write about themselves, about families or movies or books or games; or they post pictures. People like to know what kind of a person is writing what they're reading. Once again, balance is called for; a blog is a public place and you should try to avoid embarrassing your readers or the company.

Write What You Know The best way to be interesting, stay out of trouble, and have fun is to write about what you know. If you have a deep understanding of some chunk of Solaris or a hot JSR, it's hard to get into too much trouble, or be boring, talking about the issues and challenges around that.

On the other hand, a Solaris architect who publishes rants on marketing strategy, or whether Java should be open-sourced, has a good chance of being embarrassed by a real expert, or of being boring.

Financial Rules There are all sorts of laws about what we can and can't say, business-wise. Talking about revenue, future product ship dates, roadmaps, or our share price is apt to get you, or the company, or both, into legal trouble.

Quality Matters Use a spell-checker. If you're not design-oriented, ask someone who is whether your blog looks decent, and take their advice on how to improve it.

You don't have to be a great or even a good writer to succeed at this, but you do have to make an effort to be clear, complete, and concise. Of course, "complete" and "concise" are to some degree in conflict; that's just the way life is. There are very few first drafts that can't be shortened, and usually improved in the process.

Think About Consequences The worst thing that can happen is that a Sun sales pro is in a meeting with a hot prospect, and someone on the customer's side pulls out a print-out of your blog and says "This person at Sun says that product sucks."

In general, "XXX sucks" is not only risky but unsubtle. Saying "Netbeans needs to have an easier learning curve for the first-time user" is fine; saying "Visual Development Environments for Java suck" is just amateurish.

Once again, it's all about judgment: using your weblog to trash or embarrass the company, our customers, or your co-workers, is not only dangerous but stupid.

Disclaimers Many bloggers put a disclaimer on their front page saying who they work for, but that they're not speaking officially. This is good practice, but don't count it to avoid trouble; it may not have much legal effect.

Expressing WS Policies in OWL

Bijan Parsia
Maryland Information and Network Dynamics Laboratory
University of Maryland
College Park, MD 20742
bparsia@isr.umd.edu

Vladimir Kolovski
Dept. of Computer Science
University of Maryland
College Park, MD 20742
kolovski@cs.umd.edu

Jim Hendler
Maryland Information and Network Dynamics Laboratory
University of Maryland
College Park, MD 20742
hendler@cs.umd.edu

ABSTRACT
In this paper, we present two translations of the Web Service Policy Framework (WS-Policy) into OWL-DL. First, we provide an introduction to WS-Policy and we argue the benefits of using OWL and RDF to express web service policies. Then, we provide two translations from WS-Policy to OWL, one of them representing policies as instances, and the second one as classes. Finally, we provide a survey of existing web policy languages and a general idea of their expressivitiy.

Categories and Subject Descriptors
I.2.4 [**Artificial Intelligence**]: Knowledge Representations and Formalisms – Semantic Web, *OWL, RDF, Web Service Policy.*

General Terms
Performance, Design, Standardization, Languages.

Keywords
Web services, Policy languages, OWL, RDF, WS-Policy

1. Introduction
Web services interact with each other by exchanging SOAP messages. To provide for a robust development and operational environment, services are described using machine-readable metadata. This metadata serves several purposes, one of them being describing the capabilities and requirements of a service — often called the service *policy.* In recent years, there have been many different web service policy language proposals, all of them describing languages with varying degrees of expressivity and complexity. However, with most current proposals it is difficult to determine their expressivity and computational properties as most lack formal semantics. One characteristic of the proposed languages is that they involve policy *assertions* and combinations of assertions. For example, a policy might assert that a particular service requires some form of reliable messaging or security, or it may require both reliable messaging and security. Several industrial proposals (e.g., WS-Policy [13] and Features and Properties [2]) appear to restrict them to a kind of *propositional* logic with policy assertions being atomic propositions and the combinations being conjunction and disjunction. By mapping the policy language constructs into a logic (e.g., some variant of first order logic) we can acquire a clear semantics for the policy languages, as well as a good sense of the computational aspects of the languages.

If we can map the policy languages into a standardized logic, then we can benefit from the tools and general expertise one expects to come with a reasonably popular standard. By mapping two policy languages into the same background formalism, we will be able to provide some measure of interoperability between policies written in distinct languages. If we are smart in our mapping, we should also be able use pre-existing reasoners for the standardized logic to do useful reasoning about policies.

Our language of choice is the Web Ontology Language, OWL [4], and the Resource Description Framework (RDF [6]). Both RDF and OWL are strict subsets of first order logic, with the subspecies OWL-DL being a very expressive yet decidable subset. OWL-DL builds on the rich tradition of *description logics* where the tradeoff between computational complexity and logical expressivity has been precisely and extensively mapped out and practical, reasonably scalable reasoning algorithms and systems have been developed.

In this paper, we have mapped one of the policy languages, WS-Policy, to OWL-DL. WS-Policy is a policy language being developed by IBM, Microsoft, BEA, and other major web services vendors and is generally considered to be the policy language with the most momentum. We have chosen two approaches: expressing policies as instances, and expressing them as classes. With the latter, we are able to use our OWL-DL reasoner, Pellet [8] as a policy engine with analysis services that go far beyond what is usually offered. In the next section we describe our mappings.

2. Mappings
Our implementation consists of two different translations, one being where the WS-Policy grammar is encoded in OWL and the other where we are trying to capture the formalism underlying the WS-Policy grammar. In the first case, individual policies are translated to OWL-DL instances, whereas in the second case they are translated into OWL-DL class expressions. This is no surprise as WS-Policy is pretty clearly intended to be a subset of propositional logic and OWL-DL is propositionally closed.

2.1. Policies as Instances

The first ontology is an attempt at designing an OWL ontology that accurately reflects the WS-Policy grammar which is originally expressed as an XML Schema. This translation essentially captures the syntax of WS-Policy, but not its semantics.

As mentioned before, WS-Policy introduces a simple grammar for expressing policy assertions. These assertions allow developers to add metadata to service description at development time or at runtime. Examples of development time policy would include a specification of which character encodings are supported, or which specifications, and which versions of those specifications are supported by the service. An example of runtime policy would include interruption in the availability of the Web service due to system maintenance.

Assertions are the building block of a Web service policy and satisfying them usually results in a behavior that satisfies the conditions for the service endpoints to communicate. A policy assertion is supported by a requestor if and only if the requestor satisfies the requirement, or accommodates the capability, corresponding to the assertion. Policy assertions usually deal with domain-specific knowledge, and they can be grouped into policy alternative. An alternative is satisfied only if the requestor of the service satisfies all of the policy assertions contained in the alternative. Note that in our ontology policy assertions and alternatives are represented with separate OWL classes related with the **containsAssertions** property. Determining whether a policy alternative is supported is done automatically using the results of the policy assertions.

A policy is supported by a requestor of a service if the requestor satisfies at least one of the alternatives in the policy. Once the policy alternatives have been evaluated, it can be automatically deduced whether a policy is supported by the requestor.

There are two operators used to express relations between policies, alternatives and assertions: All and ExactlyOne. These operators are implemented as OWL classes **OperatorAll** and **OperatorExactlyOne** in our ontology. OperatorAll requires all the assertions to hold in order for the policy alternative to be satisfied. OperatorExactlyOne specifies that exactly one of the assertions has to hold in a collection of policy alternatives for the policy assertion to be satisfied.

In order to illustrate our work, we present an OWL version of a policy requiring the web service to use X.509 certificates or Kerberos tickets as security token types.

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:wsp="http://www.mindswap.org/~kolovski/ws-policy.owl#"
          xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-
         secext-1.0.xsd">
<wsp:Policy>
  <wsp:constrainedBy>
    <wsp:OperatonExactlyOne>
      <wsp:constrainedBy>
        <wsse:SecurityToken wsp:Preference="100">
          <wsp:Usage rdf:resource="http://www.w3.org/2004/08/20-ws-pol-pos/ns#Required"/>
          <wsse:tokenType>wsse:Kerberosv5TGT</wsse:tokenType>
        </wsse:SecurityToken>
      </wsp:constrainedBy>
      <wsp:constrainedBy>
        <wsse:SecurityToken wsp:Preference="1">
          <wsp:Usage rdf:resource="http://www.w3.org/2004/08/20-ws-pol-pos/ns#Required"/>
          <wsse:tokenType>wsse:X509v3</wsse:tokenType>
        </wsse:SecurityToken>
      </wsp:constrainedBy>
    </wsp:OperatolExactlyOne>
  </wsp:constrainedBy>
</wsp:Policy>
</rdf:RDF>
```

Listing 1. RDF representation of a policy using a WS-Policy grammar expressed in OWL[1].

The approach described above gives us a clear way of expressing the syntax of WS-Policy in OWL. This approach has its advantages, as described in Section 2. However, the previous ontology does not capture the semantics of WS-Policy, so it is

[1] The ontology capturing the WS-Policy grammar is available at http://mindswap.org/dav/ontologies/ws-policy_instance.owl

difficult, for example, to determine whether two policies are consistent with each other. The following translation does a better job of capturing the semantics of WS-Policy.

2.2. Policies as Classes

Our second translation maps the WS-Policy formalism directly in OWL. We accomplish that by translating the WS-Policy constructs from a normal form policy expression into OWL constructs. A normal form policy expression is a straightforward XML Infoset representation of a policy, enumerating each of its alternatives that in turn enumerate each of its assertions. Following is a schema outline for the normal form of a policy expression:

```
<wsp: Policy…>
  <wsp:ExactlyOne>
    [ <wsp:All> [<Assertion…> … </Assertion>]* </wsp:All> ]*
  </wsp:ExactlyOne>
</wsp:Policy>
```

Listing 2. Normal form of a policy expression

Policy expressions can also be represented in more compact forms, using additional operators such as **wsp:Optional**, however as shown in [13] the policy expressions can all be expanded to normal form. Therefore we only provide a mapping of the constructs used in a normal form policy expression: **wsp:ExactlyOne** and **wsp:All**.

First, we map policy *assertions* directly into OWL-DL atomic classes (which correspond to atomic propositions). Though WS-Policy assertions often have some discernible substructure, it is not key to their logical status in WS-Policy. Or rather, that substructure is idiosyncratic to the assertion set, rather than being a feature of the background formalism. So a general WS-Policy engine must be adapted to deal with their structure, if it is to do so. The WS-Policy specification asserts:

> "Assertions indicate domain-specific (e.g., security, transactions) semantics and are expected to be defined in separate, domain-specific specifications."

It seems unfortunate that each domain-specific specification comes with its own domain specific syntax. If we are to capture the semantics of each assertion language, we must separately map each assertion language into OWL. Our default of treating each assertion as a simple atomic proposition is reasonable for general policy manipulation, since a general purpose policy engine will work roughly the same way.

Mapping wsp:All to an OWL construct is straightforward because wsp:All means that all of the policy assertions enclosed by this operator have to be satisfied in order for communication to be initiated between the endpoints. Thus, it is a logical conjunction and can be represented as an intersection of OWL classes. Each of the members of the intersection is a policy assertion, and the resulting class expression (using the operator **owl:intersectionOf**) is a custom-made policy class that expresses the same semantics as the WS-Policy one.

Handling wsp:ExactlyOne might be trickier, depending on the interpretation of the construct. There are two possible interpretations:

a) wsp:ExactlyOne means that a policy is supported by a requester if and only if the requester supports at least one of the alternatives in the policy. In the previous version of WS-Policy there was a wsp:OneOrMore construct capturing this meaning. In such case, the wsp:ExactlyOne is an inclusive OR , and can be mapped using owl:unionOf.

b) The other interpretation is that wsp:exactlyOne means that only one, not more, of the alternatives should be supported in order for the requester to support the policy. This is supported by [13], where it's stated that although policy alternatives are meant to be mutually exclusive, it cannot be decided in general whether or not more than one alternative can be supported at the same time. Our translation covers this more complicated case.

Wsp:ExactlyOne can be translated in OWL in the following way: for n different policy assertions, expressed as OWL classes themselves, wsp:ExactlyOne is the class expression consisting of the members of each separate policy class that do NOT also belong to another policy class. In OWL terms, it is the union of all of the classes with the complement of their pair-wise intersections. Because of the pair-wise intersections there is a quadratic increase in the size of the OWL construct that is used as a mapping for wsp:ExactlyOne. Following is a table summarizing both of the translations:

Table 1. Mapping WS-Policy to OWL

WS-Policy Construct	OWL Expression
Wsp:All (policies A and B)	intersectionOf (A B)
Wsp:ExactlyOne (policies A and B)	intersectionOf(complementOf (intersectionOf (A B)) unionOf (A B))

In order to illustrate how the mapping of wsp:All and wsp:ExactlyOne works, we present a sample policy ontology. The general WS-Policy Assertions are stored as OWL classes, for example there is a SecurityTokenType class with subclasses KerberosTicket, UsernameToken and X509Certificate. Other assertions are stored, too: Language, Messaging, SpecVersion and TextEncoding. Figure 1 illustrates the WS-Policy class hierarchy.

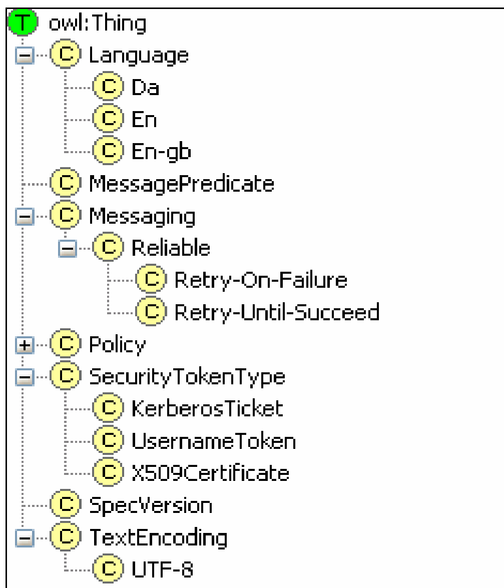


Figure 1. Sample Policy Ontology

Having stored a part of WS-PolicyAssertions [14] as OWL classes, now it's possible to develop our own custom policies. For example, say we wanted a policy such that the requestor supports Kerberos tickets and reliable messaging. Those two conditions can be represented as two assertions in a policy alternative, implying that they can be mapped to an owl:intersectionOf. The corresponding OWL expression shown in Figure 2.

```
<owl:Class rdf:about="http://www.mindswap.org/dav/ontologies/policyContainmentTest.owl#GeneralReliabilityKerberosPolicy">
  <owl:equivalentClass>
    <owl:Class>
      <owl:intersectionOf rdf:parseType="Collection">
        <owl:Class rdf:about="http://www.mindswap.org/dav/ontologies/policyContainmentTest.owl#Reliable">
        </owl:Class>
        <owl:Class rdf:about="http://www.mindswap.org/dav/ontologies/policyContainmentTest.owl#KerberosTicket">
        </owl:Class>
      </owl:intersectionOf>
    </owl:Class>
  </owl:equivalentClass>
  <rdfs:subClassOf>
    <owl:Class rdf:about="http://www.mindswap.org/dav/ontologies/policyContainmentTest.owl#Policy">
    </owl:Class>
  </rdfs:subClassOf>
</owl:Class>
```

Figure 2. Wsp:All representation in OWL

For the wsp:ExactlyOne example, we consider a policy that expects the requestor to provide either a Kerberos ticker, or an X509 certificate, but not both. In OWL, it would be represented by the class expression composed of the elements that are exclusive to KerberosTicket and X509Certificate. Figure 3 represents a serialization of the class expression in RDF/XML.

```
<owl:Class rdf:about="http://www.mindswap.org/dav/ontologies/policyContainmentTest.owl#KerberosOrX509Policy">
  <owl:equivalentClass>
    <owl:Class>
      <owl:intersectionOf rdf:parseType="Collection">
        <owl:Class>
          <owl:unionOf rdf:parseType="Collection">
            <owl:Class rdf:about="http://www.mindswap.org/dav/ontologies/policyContainmentTest.owl#KerberosTicket">
            </owl:Class>
            <owl:Class rdf:about="http://www.mindswap.org/dav/ontologies/policyContainmentTest.owl#x509Certificate">
            </owl:Class>
          </owl:unionOf>
        </owl:Class>
        <owl:Class>
          <owl:complementOf>
            <owl:Class>
              <owl:intersectionOf rdf:parseType="Collection">
                <owl:Class rdf:about="http://www.mindswap.org/dav/ontologies/policyContainmentTest.owl#X509Certificate">
                </owl:Class>
                <owl:Class rdf:about="http://www.mindswap.org/dav/ontologies/policyContainmentTest.owl#KerberosTicket">
                </owl:Class>
              </owl:intersectionOf>
            </owl:Class>
          </owl:complementOf>
        </owl:Class>
      </owl:intersectionOf>
    </owl:Class>
  </owl:equivalentClass>
  <rdfs:subClassOf>
    <owl:Class rdf:about="http://www.mindswap.org/dav/ontologies/policyContainmentTest.owl#Policy">
    </owl:Class>
  </rdfs:subClassOf>
</owl:Class>
```

Figure 3. Wsp:ExactlyOne representation in OWL

2.3. Policy processing

One of our arguments for expressing policies using OWL was the ability to reason about policy containment – whether the requirements for supporting one policy are a subset of the requirements for another. That would allow us to be more flexible in determining whether a particular requestor supports a policy, in the cases where the requestor supports a superset of the requirements established by the policy.

Policy
- GeneralReliabilityKerberosPolicy
- GeneralReliabilityUsernamePolicy
  - ExactlyOneExamplePolicy
  - Retry-On-FailureUsernamePolicy
  - Retry-Until-SucceedUsernamePolicy
- KerberosOrX509Policy

Figure 4. Example of policy containment

In the previous figure, Retry-On-FailureUsernamePolicy is an intersection of Retry-On-Failure and UsernameToken. However, since Retry-On-Failure is a subclass of Reliable, our OWL-DL reasoner [8] classifies Retry-On-FailureUsernamePolicy as a subclass of GeneralReliabilityUsernamePolicy , meaning that any user that supports the latter will also support the former.

owl:Thing
- Messaging
- Policy
  - GeneralReliabilityKerberosPolicy
  - GeneralReliabilityUsernamePolicy
    - ExactlyOneExamplePolicy
    - Retry-On-FailureUsernamePolicy
    - Retry-Until-SucceedUsernamePolicy
- SecurityTokenType
- owl:Nothing
  - IncoherentPolicy

Figure 5. Example of policy incoherence

The above figure is an example of Swoop showing an incoherent policy. IncoherentPolicy selects two policy alternatives from an wsp:ExactlyOne, which, according to our current translation is forbidden. Note that Swoop displays an explanation of the incoherence, which can aid in repairing the policy.

In general, we get the following inferences out of the box:

1) policy inclusion (if x meets policy A then it also meets policy B; a.k.a., A rdfs:subClassOf B);
2) policy equivalence (A owl:equivalentTo B);
3) policy incompatibility (if x meets policy A then it cannot meet policy B; a.k.a, A owl:disjointWith B);
4) policy incoherence (nothing can meet policy A; a.k.a., A is unsatisfiable)
5) policy conformance (x meets policy A; a.k.a, x rdf:type A)

Some care must be taken given the open world semantics of OWL. For example, an OWL reasoner does not assume that because it cannot prove that x conforms to policy A, that x *does not* conform to policy A. It is unclear what the WS-Policy authors intend, though a closed world assumption is not unlikely. However, even if there is a closed world assumption on WS-Policies, we can handle at least some of those cases by adding explicit disjoint statements at translation time.

One futher reasoning service supported by Pellet, and integrated with Swoop, is *explanations* for inconsistencies,[16] which can be used to help debug policy incompatibility, incoherence, and the like. As we add further explanation capability to our systems, this debugging power will grow.

Thus we see that with a fairly simple mapping, we can use an off the shelf OWL reasoner as a policy engine and analysis tool, and an off the shelf OWL editor as a policy development and integration environment. OWL can also used to develop domain specific assertion languages (essentially, domain Ontologies) with a uniform syntax and well specified semantics. We can also experiment with extensions to WS-Policy, by using more expressive constructs from OWL at the policy language, as well as the assertion language, level. We can play with extensions before having to write a yet another processor for them. Of course, if it turns out that we really want to restrict ourselves to a very inexpressive subset, then we may still want to build specific reasoners and processors that are tuned for that sublanguage. But there again, our tools can help us. Pellet does expressivity analysis of ontologies, so can help determine what logic we are really using and the price of extensions.

Furthermore, ontology development techniques can be useful for policy development as well. Most human generate ontology develop iteratively, with specializations added to the class tree over time. Similarly, we can build up our policies from more general ones. A general policy could be very restrictive, setting tough guidelines for all of a companies policies.

Finally, given a similar style mapping for another policy language (say, Features and Properties, described in the next section) we can do policy analysis and integration across policy languages.

3. Other Policy Languages

In this section we provide a quick overview of the state-of-the-art in web service policy specification, by looking at the policy languages presented at [12]. To the best of our knowledge, they are sorted by increasing level of expressivity, even though lack of formal semantics and analysis hampered our effort to provide a fully correct listing. The list follows:

3.1. The Platform for Privacy Preferences Project (P3P) [9] enables Web sites to express their privacy practices in a standard format that can be retrieved automatically and interpreted easily by user agents. P3P user agents allow users to be informed of site practices (in both machine- and human-readable formats) and to automate decision-making based on these practices when appropriate. According to [15], there exists a data-centric relational semantics for P3P in which a P3P policy is modeled as a relational database. This simple semantics allows us to express P3P using RDF.

3.2. The Features and Properties architecture [2] originates from SOAP 1.2, and was integrated into WSDL 2.0 in order to support the SOAP-specific concepts. Afterwards the architecture was further expanded in order to allow Features to be more abstract. Simply put, a Feature identifies a piece of functionality, identified by a URI. An example of a Feature would be encryption. Properties are the parameters of a Feature, also identifiable by a URI. For an encryption feature, property might be the algorithm used, part of message encrypted, etc. Features & Properties are similar to WS-Policy in terms of expressivity, with one exception – they lack operators for combining policy assertions. It is argued at [3] that adding a choose one/all operators (called combinators) will prove to be useful in expressing higher-level semantics combining multiple policies.

3.2. WS-Policy [13] provides a general-purpose model and syntax to describe and communicate the policies of a Web service. It specifies a base set of constructs that can be used and extended by other Web service specifications to describe a broad range of service requirements and capabilities. WS-Policy introduces a simple and extensible grammar for expressing policies and a processing model to interpret them. The policy assertions are expressing using XML and the grammar itself is specified with XML Schema.

By using OWL we increase the expressiveness of the WS-Policy representation and it will simplify the interaction between any new protocols on one hand, and WS-Policy and WSDL on the other hand. By using OWL/RDF we do not need to focus on the ways in which the policy is attached to the web service, instead we can concentrate on analyzing the policy itself.

Also, WS-Policy uses an open content model on policy assertions to provide extensibility, and the usage of OWL and RDF can provide more expressiveness by way of subclass and subproperty constructs – re-using of derived policy assertions.

[10] provides a comparison of XML and RDF in terms of expressing WS-Policies. It also provides arguments for usage of RDF to represent WS-Policy by describing how RDF meets document merging and extensibility goals described in the WS-Policy specifications. To support this, the paper presents an RDF schema for representing web service policies upon which our policies as instances mapping was built.

3.4. KaOS Policy and Domain Services [11] use ontology concepts encoded in OWL to build policies. These policies constrain allowable actions performed by actors which might be clients or agents. The KAoS Policy Service distinguishes between authorizations (i.e., constraints that permit or forbid some action) and obligations (i.e., constraints that require some action to be performed when a state- or event-based trigger occurs, or else serve to waive such a requirement). The applicability of the policy is defined by a class of situations which definition can contain components specifying required history, state and currently undertaken action. In the case of the obligation policy the obligated action can be annotated with different constraints

restricting possibilities of its fulfillment. KAoS services have been extended to work equally well with both agent-based (e.g., CoABS Grid, Cougaar, SFX, Brahms) and traditional clients on a variety of general distributed computing platforms.

3.5. WSPL

WSPL[14] is being developed at Sun Microsystems. The Web Services Policy Language (WSPL) is suitable for specifying a wide range of policies, including authorization, quality-of-service, quality-of protection, reliable messaging, privacy, and application-specific service options. WSPL is of particular interest in several respects. It supports merging two policies, resulting in a single policy that satisfies the requirements of both, assuming such a policy exists. Policies can be based on comparisons other than equality, allowing policies to depend on fine-grained attributes such as time of day, cost, or network subnet address. By using standard data types and functions for expressing policy parameters, a standard policy engine can support any policy. The syntax is a strict subset of the OASIS eXtensible Access Control Markup Language (XACML [5], discussed below) Standard. WSPL has been implemented, and is under consideration as a standard policy language for use with web services.

3.6. XACML provides a policy language which allows administrators to define the access control requirements for their application resources. The language and schema support include data types, functions, and combining logic which allow complex (or simple) rules to be defined. XACML also includes an access decision language used to represent the runtime request for a resource. When a policy is located which protects a resource, functions compare attributes in the request against attributes contained in the policy rules ultimately yielding a permit or deny decision. It is a powerful language, able to also express first order and higher order functions.

3.7. Rei [7] is a policy specification language based on a combination of OWL-Lite, logic-like variables and rules. It allows users to develop declarative policies over domain specific ontologies in RDF, DAML+OIL and OWL. Rei allows policies to be specified as constraints over allowable and obligated actions on resources in the environment. A distinguishing feature of Rei is that it includes specifications for speech acts for remote policy management and policy analysis specifications like what-if analysis and use-case management. As Rei is geared towards distributed environments, it also includes conflict resolution specifications like modality preferences or priority assignments between policies or between individual rules of a policy.

Having produced a mapping for WS-Policy to OWL, we have shown that also Features and Properties and P3P can also be mapped, since they are less expressive than WS-Policy. We plan to focus on the more expressive languages (WSPL, XACML, Rei) in the future, to determine how much of them can be mapped into OWL, or whether we must move to a more expressive language (like SWRL), or out of first order logic altogether. We believe that translation considerations for existing and used policy languages should be a factor in future extensions to OWL.

4. Conclusion

We have presented a translation of the base formalism of WS-Policy into OWL-DL and described how those translations can be used for policy analysis, processing, and development. If our translation is correct, we have provided a formal semantics for WS-Policy. At worst, we have exposed some of the assumptions and ambiguities about the current specification.

We have demonstrated that an OWL-DL reasoner provides useful services for policy analysis, including policy containment, incompatibility, conformance, and incoherence. We expect that having such services available will raise the bar for policy engines overall.

In our future work, we intend to provide a standard mapping of all the current WS-Policy assertion languages with some structural fidelity. We also plan to attempt translations of at least parts of the other policy languages we described in order to get a more precise sense of their expressivity. If they cannot be mapped into OWL, we intend to isolate the incompatible expressivity in order to determine whether there are reasonable extensions to OWL that could accommodate it.

Finally, we intend to further develop our tools *as* WS-Policy processing tools. We shall investigate the gap between general purpose tools like Pellet and Swoop and things tuned for WS-Policy. For example, our explanation facility might do better for WS-Policies if it knew the characteristic structure of their translations.

5. ACKNOWLEDGEMENTS

This work was completed with funding from Fujitsu Laboratories of America- College Park, Lockheed Martin Advanced Technology Laboratory, NTT Corp., Kevric Corp., SAIC, National Science Foundation, National Geospatial-Intelligence Agency, DARPA, US Army Research Laboratory, NIST, and other DoD sources.

6. REFERENCES

[1] Anderson, A. H. An Introduction to the Web Services Policy Language. Sun MicroSystems. http://research.sun.com/projects/xacml/Policy2004.pdf

[2] Daniels, G. Comparing Features / Properties and WS-Policy. *W3C Workshop on Constraints and Capabilities for Web Servies*. Redwood Shoes, CA, USA, Oct 12 -13, 2004.

[3] Daniels, G. Features and Properties Musings. www-ws-desc@w3.org mailing list, October 2003. http://lists.w3.org/Archives/Public/www-ws-desc/2003Oct/0144.html

[4] Dean, M. and Schreiber G. OWL Web Ontology Language. Reference W3C Recommendation, http://www.w3.org/tr/owl-ref/. Feb 2004.

[5] Godik, S., and Moses,T., eds. OASIS eXtensible Access Control Markup Language (XACML) Version 1.1. Oasis Committee Specification, http://www.oasis-open.org/committees/download.php/4103/cs-xacml-specification-1.1.doc. 24 July 2003.

[6] Lassila, O. and Swick, R. Resource Description Framework (RDF) Model and Syntax Specification. W3C recommendations, WWW Consortium. Cambrigde,MA, USA. Feb 1999.

[7] Kagal, L. et al. A policy Language for a Pervasive Computing Environment. In Collection, *IEEE 4th International Workshop on Policies for Distributed Systems and Networks*. June 2003.

[8] Pellet – OWL-DL reasoner, http://www.mindswap.org/2003/pellet.

[9] Platform for Privacy Preferences Project. http://www.w3.org/P3P/

[10] Prud'hommeaux, E. RDF for Web Service Assertions. *W3C Workshop on Constraints and Capabilities for Web Services*. Redwood Shores, CA, USA. Oct 12-13, 2004.

[11] Uszokand, A. and Bradshaw, J. KAoS Policies for Web Services. *W3C Workshop on Constraints and Capabilities for Web Servies*. Redwood Shoes, CA, USA, Oct 12 -13, 2004.

[12] W3C Workshop on Constraints and Capabilities for Web Services. Redwood Shores, CA, USA. Oct 12-13, 2004. http://www.w3.org/2004/06/ws-cc-cfp.html.

[13] Web Services Policy Framework (WS-Policy). http://www-106.ibm.com/developerworks/library/specification/ws-polfram/.

[14] Web Services Policy Assertions Language. http://www-106.ibm.com/developerworks/library/ws-polas/

[15] Yu, T., Li N., and Anton A.L. A formal semantics for P3P. *ACM Workshop on Secure Web Services, October 29 2004, Fairfax VA, USA.*

[16] *Parsia, B., Sirin. E., Kalyanpur, A.* Debugging OWL Ontologies*, In The 14th International World Wide Web Conference (WWW2005), Chiba, Japan, May 2005.*

Policy-based Access Control for Task Computing Using Rei

Ryusuke Masuoka, Mohinder Chopra, Yannis Labrou, Zhexuan Song, Wei-lun Chen
Fujitsu Laboratories of America
8400 Baltimore Avenue, Suite 302
College Park, MD 20740, USA
+1 (301) 486-0398
{ryusuke.masuoka, mohinder.chopra, yannis.labrou, zhexuan.song, sam.chen}@us.fujitsu.com

Lalana Kagal
Massachusetts Institute of Technology (MIT)
Computer Science and Artificial Intelligence Laboratory (CSAIL)
32 Vassar Street, Cambridge, MA 02139, USA
+ 1 (617) 253-2613
lkagal@csail.mit.edu

Timothy Finin
University of Maryland, Baltimore County
1000 Hilltop Circle
Baltimore, MD 21250, USA
+1 (410) 455-3522
finin@umbc.edu

ABSTRACT

In this paper, we describe a policy-based access control implementation for Task Computing using the Rei policy engine.

Task Computing lets ordinary end-users accomplish complex tasks on the fly from an open, dynamic, and distributed "universe of network-accessible *resources*" in ubiquitous computing environments as well as those on the Internet.

The Rei policy specification language is an expressive and extensible language based on Semantic Web technologies. The Rei policy engine reasons over Rei policies in OWL and domain knowledge to answer queries about the current permissions and obligations of an entity.

To provide unobtrusive and flexible access control for Task Computing, a framework was created in which several Rei policy engines were endowed with Web Services APIs to dynamically process facts from clients, the private policies of service providers, shared policies, and common shared ontologies. The framework is implemented and deployed for Fujitsu Laboratories of America (FLA), College Park office and evaluated.

Categories and Subject Descriptors

J.7 [**Computer Applications**]: Computers in Other Systems

General Terms

Management, Design, Experimentation, Security, Human Factors, Languages.

Keywords

Task Computing, Rei, Policy, Semantic Web, OWL, OWL-S

1. Introduction

As the World Wide Web evolves as a computing and network infrastructure, policy management becomes crucial to provide access control not only for information on the Internet, but resources in general, including networked devices and Web Services, in such diverse environments as ubiquitous computing and grid computing.

This paper focuses on access control for end-users to resources in ubiquitous computing environments. These resources are described abstractly in OWL as services, and are mainly realized as UPnP devices and simple Web Services. This focus of resources in ubiquitous computing poses a different set of requirements and problems than for information on the Internet. It is not that one is more difficult than the other. For example, a framework can leverage from the physical reality in ubiquitous computing environments. But the very dynamic nature of ubiquitous computing environments definitely offers new kinds of challenges.

Our contribution is to add a flexible policy-based access-control to ubiquitous computing and demonstrate its utility and effectiveness in a ubiquitous computing application. Task Computing (TC, [1][2][3][[4]]) is a user-oriented framework that lets end-users accomplish complex tasks on the fly from open, dynamic, and distributed "universe of network-accessible *resources*" in environments rich with applications, devices, and services. Task Computing provides many ways for users to interact with these ubiquitous environments and applies Semantic Web technologies, such as OWL (Web Ontology Language, http://www.w3.org/2001/sw/WebOnt/) and OWL-S (http://www.daml.org/servcies) as its core enablers. In each environment, functions (devices/OS/applications) are virtualized as services. Through discovery mechanisms such as UPnP, TC clients find those services and obtain their OWL-S files as their semantic service descriptions. With those OWL-S files, TC clients let the end-users to manipulate (compose, execute, publish, etc.) the services on the spot.

When started, the TC client dynamically finds the local services on the computer on which it runs and pervasive services in the sub-network the computer is on. UPnP is used for the service discovery on the sub-network. When a local or subnet service is discovered, the TC client downloads the appropriate OWL-S files that represents its semantic service description. Using the OWL-S descriptions, TC client such as STEER allow a user to compose and execute the services. The user can also create new semantic services dynamically by instantiating or composing other services. For example, Task Computing enables a user to display a presentation file from his mobile PDA or computer on the

Copyright is held by the author/owner(s).
WWW 2005, May 10--14, 2005, Chiba, Japan.

stationary room projector without connecting a VGA cable, even if this is the first time the user has been in the room. In another example, a user can print a presentation file from his laptop on the printer provided in the room without configuring his computer, or show a photo just taken with his digital camera on the photo frame in the same room immediately and print it on a photo printer without moving memory cards around, or, display the current weather at an address in his PIM (Personal Information Manager) on the projector with just a few operations of point and click. Task Computing enables end-users to accomplish all of the above and more through a simple graphical user interface to the Task Computing environment. You can even use your own voice to make those same things happen through a voice-based Task Computing client, VoiceSTEER.

Rei is a policy specification language for describing different kinds of policies in a wide range of application domains. The main goal of Rei is to address the issue of governing *autonomous* entities in *constantly evolving* distributed environments. Rei provides specifications for describing declarative machine-understandable policies enabling both policy enforcement and a more normative approach where autonomous entities can decide whether or not to fulfill the applicable policy.

Rei is represented in an extension of OWL-Lite ([6][8][9][10]) and can be used to describe policies over domain knowledge in different ontology languages such as RDF, DAML+OIL, and OWL. Though its classes and properties are represented in OWL, Rei also includes logic-like *variables* giving it the flexibility to specify constraints that are not directly possible in OWL e.g., the uncle relation, the same age as relation etc. Rei models deontic concepts of permission, prohibition, obligation, and dispensation and supports speech acts such as delegation, revocation, cancel, and request for dynamic policy modification.

As most entities in pervasive environments will have several overlapping policies of behavioral norms, constraints, and rules acting on them, they will be over-constrained. This means that they cannot always satisfy all of the policies, but deviating too much or too often has its consequences - loss of reputation, penalty clauses, imposition of sanctions, etc. Rei provides two mechanisms for handling these situations namely consequences and meta policies. Rei allows consequences to be modelled as 'sanctions' so that autonomous entities or providers can reason over them to decide whether or not to deviate from a certain policy. Rei also allows meta policies to be used to resolve conflicts. Rei models two main types of meta policies: (i) for defaults and (ii) for conflict resolution to handle different requirements of policies. Depending on the type of conflict resolution required, the appropriate meta policy should be selected. Some policies may want a more high level meta policy and can use default behaviors or modality precedences. However, for tighter control, priorities are more suitable but are tougher to define and maintain.

In order to support policy development, Rei provides two forms of policy analysis: use-cases (also known as test-case analysis) and what-if analysis (also known as regression testing). The policy engine includes analysis tools accessible via a Java interface that can be executed by policy engineers to check the consistency and validity of the policies and ontologies.

From the initial implementation of Task Computing Environment, it was immediately apparent that it requires some kinds of access control for the services because it makes so easy for the end-users to use the devices and services dynamically found on the same sub-network. In home network environment, it would not be so much a problem as long as the network is firewalled from the outside networks. But when Task Computing should be applied to, for example, office or hospital environments where there are many devices that should be protected from abuses by unauthorized accesses.

To provide unobtrusive and flexible access control for Task Computing, a framework is created with Rei policy engines endowed with Web Services API to process facts from the client, service's private policy, shared policies, and ontologies dynamically. The framework is implemented and deployed for Fujitsu Laboratories of America (FLA), College Park office and evaluated.

In this paper, the motivation and design goals of the work are given in Section 2. The implementation and test deployment of Task Computing access control with the Rei policy engine is described in Section 3. Then Section 4 describes how the above design goals are met. Related work is discussed in Section 5 and Section 6 concludes this paper.

2. Motivation and Design Goals

As mentioned above, the initial implementation of Task Computing immediately revealed the need for access control of services. A simple access control mechanism, which will not be described any further here, was implemented in the early stage. This mechanism leveraged the physical embodiment as devices of many services in Task Computing and this mechanism is often enough for a simple deployment of services based on devices, but it had its limitations. It was inappropriate for large deployments of dynamic services and clients, or for services *without* their physical embodiments. Simple identity or role based access control mechanisms were unable to meet the requirements of these dynamic environments. A sophisticated policy-based solution for Task Computing was necessary to cover such cases. At the core of the solution, a way to express rule based policies and an engine to process the policies were required. The Rei policy specification language and Rei policy engine came as a perfect match.

Rei is an expressive policy language based on Semantic Web technologies. As Task Computing had already embraced OWL and OWL-S as its core enabler, it made it easier to integrate many aspects of Task Computing into the policy language. Specifically Task Computing needs seamless inferences over policies, facts, and ontologies. The Rei policy engine can combine dynamically policies including delegations, OWL ontologies, and facts described using ontologies and infer the access rights for users and programs.

The dynamic nature of ubiquitous computing environments also requires the policies to be defined not in terms of ID's and roles, but rules based on properties of entities such as users, devices, and services. In the ubiquitous environment with often unforeseeable entities, the access control should shift to rule-based approaches using descriptions of entities involved.

In order to give enough flexibility, it necessitates the use of mechanisms for updating the policies on the fly. Especially delegation mechanisms, which Rei also supports, are imperative. Users do let others use devices and services on their behalf or temporarily in everyday life. If the system does not allow such

flexibility, the users would be forced to drop the mechanism totally or find a way to evade it.

We also deem it important that the system allows developers, system administrators, and even end-users to specify the policies in a natural and intuitive way. It would make the system very easy to use if, for example, the system lets the user specify policy very close to everyday languages and processes them in the way the ordinary people would expect it to be processed. While the policy language itself needs not to have everyday language aspects, a policy language with high expressivity enables such a system by allowing mapping the user's policy specifications correctly into the policy language.

Such considerations made the Rei policy specification language and policy engine a natural choice for us.

On the other hand, we wanted to get leverage from the ubiquitous computing environments as application areas of Task Computing. It can be difficult to hand out credentials signed by appropriate CA's to the users. As it turns out in the next section, the process is smoothly incorporated into the office check-in process and the credential is copied on the physical memory device for the user with the full Task Computing client on it. The credential is sent by the user through the Task Computing client to the service to be authenticated and consumed by the Rei engine.

When we design the access control for Task Computing Environment using Rei, the following items are set as its design goals.

1. Minimally obtrusive for users
2. Enable both centralized/distributed solutions
3. Allow multiple certificate authorities
4. Secure dynamic delegation

For the first point, it is always a trade-off between security and ease of use. (You can create a perfectly secure system … just let no one use it.). But with appropriate technologies and smart deployment of the system, we can shift the balance, more security with less obtrusiveness. If the access control is difficult or cumbersome to use, it would kill the Task Computing experience. It is also imperative to finish the policy calculation in a reasonable amount of time. The access control is secondary function to the main function. It is like putting the cart in front of the horse if it takes longer time than the main function.

Secondly, we wanted to have both centralized/distributed solutions possible because the access control deployments can be different from one site to another. For some site, an IT department might want to manage the policy centrally, thus requiring a centralized solution. In some other cases, the end-user might want to set some policy for a single device. It is preferable, for example, the end-user can set the policy for the device at its initial configuration. Such a distributed solution is often enough for a small office. There is another aspect of centralized/distributed solutions as to where the policy engine should run. In case of resource-limited devices, there might be no choice, but to choose the centralized solution in which the device accesses the policy engine running on a different more powerful machine.

The third point is important when you consider the applications for relatively open spaces such as shopping malls. By allowing multiple certificate authorities in the framework, it can maximize the chances that the user can use the service. Of course, the user and the service need to agree on at least one common certificate authority that they both trust, in order for authentication to happen.

The last point is crucial in order to make the access control flexible. Sometimes one wants to override the default access control to let someone else to use the service. It is necessary that the person has the enough authority to do it and that it should be done securely. But if the system does not allow such flexibility, the user would eventually find the system useless or tries to find ways to circumvent the access control.

3. Implementation and Test Deployment

We have ported the Rei policy engine to run in the Windows environment because many of the Task Computing services are provided by Windows-based systems. A Web Services API was created for the Rei engine to facilitate its use in a highly distributed environment. We incorporated the access control based on the Rei policy engine into the "*Pervasive Print"* TC service, which lets users print files remotely (without any printer setup) to create the "*Secure Print"* service. The Credential Creator software was produced to easily create a digitally signed credential in the OWL format. We also created the Delegation Manager software to let the users insert and/or remove delegation statements (in the Rei format) into/from the shared policy site securely over HTTPS.

The resulting system was deployed in the Fujitsu Laboratories of America (FLA), College Park office. The Credential Creator was installed on the desktop machine in the reception area, the *"Secure Print"* service was installed on a computer with a printer in the conference room along with the Rei policy engine. (Here we had the "distributed solution" in the sense that the policy engine is distributed to each service.)

We will explain the usage scenario first and then give the details how it is realized.

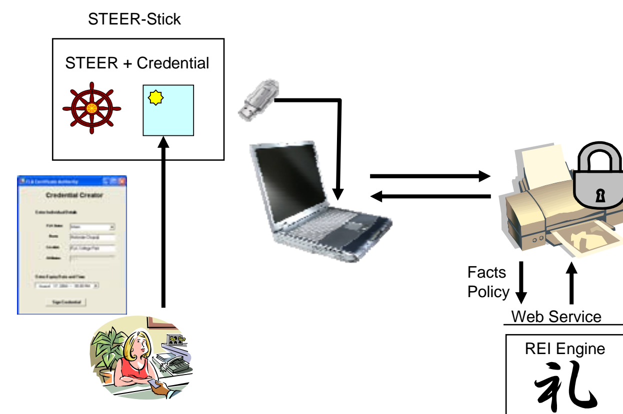


Figure 1. Deployment of Task Computing Access Control

The scenario goes like this (See Figure 1). Mohinder, a UMBC (University of Maryland, Baltimore County) student, visits FLA, College Park. Valerie, the Office Administrator of FLA, College Park, greets Mohinder in the reception area.

1. Valerie creates a STEER-Stick with credential for Mohinder.

STEER-Stick is a USB memory device with all the software necessary to run STEER, a Task Computing client including Java runtime along with the credential. The credential includes his name, affiliation, status (*''Visitor''*) and metadata of credential (its creation date, expiration date/time, etc.), and the digital signature signed with the FLA private key. The Credential Creator software saves the credential in OWL format in the credential folder of the STEER-Stick. It also saves an HTML file for the human to check the contents of the credential.

✓ Mohinder runs the STEER using the STEER-Stick in his laptop. STEER finds the "Secure Print" service dynamically and show the service with a key icon.

The *"Secure Print''* OWL-S file states that an FLA credential is required. (It can state that it requires one of multiple credentials.) When STEER finds that the service requires a credential, it shows a key icon for the service.

2. Mohinder tries to use the "*Secure Print"*, but he fails because a "*Visitor"* is not allowed to use it.

Based on the *"Secure Print"* OWL-S file, STEER looks for an FLA credential in its "credential" folder. When it finds it, it sends the credential along with service invocation parameters in the Web Service call.

"Secure Print" checks the digital signature of credential to make sure it is valid. (So that facts in the credential are not modified.) Then it uses these facts to determine if the caller has the authority to use the service by the Rei policy engine, which is called through Web Service API. The Rei policy engine determines that Mohinder can not use the service as he is just a "Visitor" and returns the result through its Web Service API. The service , in turn, returns an error for the original Web service call with the reason.

3. Mohinder asks Ryusuke to delegate the right to print.

Ryusuke uses the Delegation Manager software to assert a statement to delegate the right to Mohinder by Ryusuke to the FLA shared policy site securely.

4. Mohinder tries again to use the "Secure Print" and this time he succeeds.

There is a statement at FLA Policy Site that Senior Employee has a right to delegate the right to visitors. With the newly added statement of the delegation, it enables Mohinder to print.

5. After that, Ryusuke revokes the delegation.

The delegation assertion created in the step 3 is removed from the FLA shared policy site by using the Delegation Manager. Mohinder can not use the service any longer.

Access control is determined based on the following elements:

✓ Facts provided by the client (authenticated by the digital signature)

✓ Printer's private policy

✓ FLA shared policy (and potentially other shared policies)

✓ Ontologies

The service can use multiple shared policies depending on its configuration. Each time, these elements listed above are mixed to determine the access control.

Figure 2 shows what happens behind the scene. The number given in the figure corresponds to the numbered item in the scenario. Shared policies and ontologies are cached and they are downloaded only when they are updated.

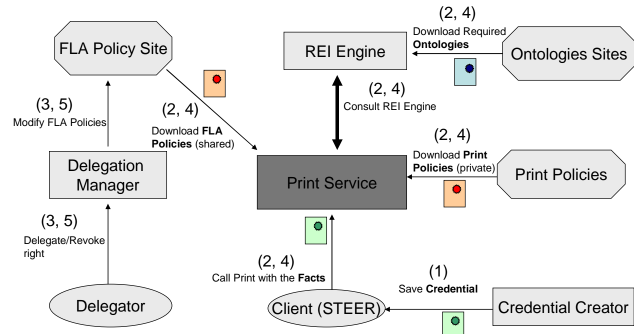


Figure 2. What is Happening behind the Scene

Figure 3 gives parts of fact, private policy for the Secure Print service, and the shared policy for FLA used in this scenario.

The scenario above centers around the value for "flaonto:Status" in the fact. All pieces of information in the fact are digitally signed and the digital signature assures its integrity. If any part of it is changed, the facts can not be authenticated.

Another thing to note is that it has the expiration time as a part of the credential's metadata. If the time has passed this expiration time, the *"Secure Print"* service will decline any request to print.

The Printer's private policy states that it can be used by a senior employee, but not by a visitor. Therefore Mohinder, who is a visitor, fails to print at first.

The FLA shared policy states that a Senior Employee has the right to delegate the right to use the *"Secure Print"* service (It is not shown in Figure 3). When Ryusuke insert his delegation statement (which is shown in Figure 3) using the Delegation Manager, this enables Mohinder to use the *"Secure Print"* because the service detects the update at the FLA Policy site and downloads the new FLA shared policy (and because of the statements that Ryusuke Masuoka is a Senior Researcher and that a Senior Researcher is a Senior Employee in the ontologies).

```
<!– Fact from Task Computing client -->
<rdf:RDF …>
  <rdfs:label lang=en>Mohinder Chorpa</rdfs:label>
  <flaonto:Name  …>Mohinder Chorpa</flaonto:Name>
  <flaonto:Expiry …>2004-08-23T23:05:28Z</flaonto:Expiry>
  <flaonto:Status …>&flaonto;FLACPVisitor</flaonto:Status>
  <flaonto:Affiliation …>UMBC</flaonto:Affiliation>
  <Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
    <SignedInfo>
      …
    </SignedInfo>
    <SignatureValue>ZrbEVA7JWWGNbpqc…Jo6dDw=</SignatureValue>
  </Signature>
</rdf:RDF>

<!– Printer Private Policy -->
…
<deontic:Permission rdf:about="&flapolicy;right_to_be_printed_on“
  policy:desc="All senior employees have the right to print">
  <deontic:actor rdf:resource="&flapolicy;var1"/>
  <deontic:action rdf:resource="&flapolicy;printing_in_conference"/>
  <deontic:constraint rdf:resource="&flapolicy;preOrSenior"/>
</deontic:Permission>
…

<!– Delegation Inserted (and Removed) in Shared Policy-->
<action:Delegation
  rdf:ID=“Delegation2004-08-23T19:32:19ZRyusukeMasuoka">
  <action:sender rdf:resource="&inst;RyusukeMasuoka"/>
  <action:receiver rdf:resource="&inst;MohinderChorpa"/>
  <action:content>
    <deontic:Permission>
      <deontic:action rdf:resource="&inst;ASeniorEmployeePrintingAction"/>
    </deontic:Permission>
  </action:content>
</action:Delegation>
```

Figure 3. Fact, Private Policy, and Shared Policy

The example given above is kept relatively simple for the sake of easy understanding. The system as it is now can fully utilize the expressivity which the Rei engine allows. For example, a scenario, such as one in which a senior employee gives to a class of users (ex. all visitors from UMBC on Jan 31st) the right to use a class of resources (ex. all devices in the conference room), is possible.

4. Evaluation and Discussions

In this section, we discuss how we met the initial design goals set forth in Section 1.

1. Minimally obtrusive for users

We tried to keep the additionally requirements for all the users involved as little as possible.

We have created software tools such as Credential Creator and Delegation Manager so that end-user needs not to write complex OWL/Rei statements, but just to give essential information.

The credential creation process is integrated into an ordinary office check-in process in which the Office Administrator types in the visitor's name and affiliation, selects the appropriate status (selections created dynamically from an ontology) and the expiration time in the Credential Creator GUI ,and hit the save button. The digitally signed credential in OWL is automatically created and saved in the appropriate folder of the STEER-Stick USB memory device,.

The STEER-Stick includes a full Task Computing client, STEER, on it and the user can run STEER from the STEER-Stick without any installation.

STEER hides the details of using the secured services and shows only essential information. Secured services are shown with key icons so that the user knows that it requires appropriate authority to execute it. When the execution fails because of the security clearance, it will notify the user the reason. All the details are handled behind the scene such as determining from the OWL-S file if the service is secured and what kinds of credential is necessary and sending appropriate credential to the secured service.

On the service side, we found that the Rei policy engine needed to be accelerated so as not to hamper the user's experience. Originally it took seven to eight seconds to finish the access control calculation based on the fact, policies, and ontologies. In general, caching answers does not help as we can not expect fact, policies, and ontologies to remain fixed (especially facts). We made various changes to the Rei policy engine to enable it to produce answers to queries in less than one second.

2. Enable both centralized/distributed solutions

From the aspect of policy management, we can have the spectrum between centralized and distributed solutions. One can put the policies that should be kept private in the private policy while policies that can or should be shared can be put in one of the shared policies at the shared policy sites. Which shared policies for the service to use is up to the service to decide.

From the aspect of policy engine, the Rei policy engine with Web Services API allows very flexible deployment as long as the Rei policy engine is accessible from the service by HTTP/HTTPS. But the privacy of private policy is compromised to some degree when the Rei policy engine is running on a different machine because the private policy needs to be sent to the Rei policy engine for the access control calculation.

3. Allow multiple certificate authorities

We allow the OWL-S file for the service to include the multiple certificate authorities that the service accepts. On the other hand, STEER looks into its credential folder for credentials from compatible certificate authorities for the service and send the credential along with the Web Services calls if found.

For example, Mohinder may carry two credentials, one from FLA and one from UMBC in the credential folder. The OWL-S file of "Secure Print" may state that it requires a credential from FLA or 8400 Baltimore Avenue Building (where FLA, College Park office is located in). STEER selects the credential from FLA in the credential folder to use "Secure Print" service.

4. Secure dynamic delegation

With the Delegation Manager software, it is possible for end-users easily to insert (and later remove) the Rei delegation assertions into the shared policy hosted at a Web server securely over HTTPS. This gives flexibility often necessary in everyday usage of the system.

In addition to the initial design goals, we would like to discuss here about our decision not to make the Rei engine Web Services discoverable dynamically as a semantic service as it is usually the case for Task Computing Web Services. While it is easy to make the Rei Web Service discoverable through, for example, UPnP

and the service automatically starts using the Rei Web Service, it can be a security hole simply doing that. The dynamically found Rei engine needs to be authenticated and there is a bootstrapping issue. It is also likely that the human service provider has a very specific idea of which policy engine to use along with each service.

5. Related work

Extensible Access Control Markup Language (XACML) [16] is a language in XML for expressing access policies. This work is similar to ours; in that it allows control over actions and supports resolution of conflicts. However, as it is based in XML, it does not benefit from the interoperability and extensibility provided by Semantic Web languages. It also does not model speech acts or handle conflict resolution across policies.

Lately there has been a significant body of standardization efforts for XML-based security, such as WS-security, -trust, and -policy at W3C, or SAML of the OASIS Security Services Technical Committee, and the Security Specifications of the Liberty Alliance Project. The standards support low-level security or policy markups that concern formats of credentials or supported character sets for encoding. They do not address semantic user- or application-specific trust tokens and their relations. These standards have been developed to support controlled B2B applications where both client and service can be mutually authenticated and recognized. These standards are not extensible to more dynamic environments in which simple authentication is not enough, but authentication on user-defined attributes needs to be considered. For this, a semantic approach like we take in this paper, is a possible solution.

KAoS is a policy language based in OWL [17][18]. This language is similar to Rei in that it can be used to develop positive and negative authorization and obligation policies over actions. KAoS policies are OWL descriptions of actions that are permitted (or not) or obligated (or not). This limits the expressive power, so that there are policies that Rei can describe that KAoS cannot. However, KAoS allows the classification of policy statements enabling conflicts to be discovered from the rules themselves. The Rei engine can only discover conflicts with respect to a particular situation and cannot pre-determine them. However, Rei includes run-time conflict resolution by supporting meta-policies.

The paper [19] presents an XML-based specification language, which incorporates content and context based requirements for documents in XML-based Web Services. It uses a role-based access control model which simplifies authorization administration by assigning permissions to users through roles. Although it relates roles to permissions, there is no way to dynamically change these roles or permissions. Using the delegation module of REI we can change the policies dynamically to adapt to the changes in roles or permissions.

6. Conclusion

It is our belief that security and access control should be natural, flexible and minimally obtrusive for the end-users as they try to accomplish everyday tasks. If not, the users will eventually find ways to evade the mechanisms rendering them useless, at best, and possibly counter-productive. It is also important to give enough flexibility in the deployment aspect of security and access control because their requirements and rules differ from one site/office to another.

To that regard, we have been successful in adapting our flexible access control framework to blend in an ordinary office environment.

Future work includes:

✓ Discovery security

By making it so that only accessible services are found for Task Computing client, it will make the whole system more secure and easy to use.

✓ Service authentication

By using the OWL-S file of the service, the service notifies its (shareable part of) policy to the client. It enables the client to better determine if the service is executable in advance.

✓ Explanation and negotiation

The user would get frustrated if the system simply rejects his/her use of certain resources without giving a reason. The system needs to give out understandable explanation for the rejection if asked. It should also be very useful if the system can provide the information on what it requires in order to gain permission.

7. REFERENCES

[1] Masuoka, R., Parsia, B., and Labrou, Y., “Task Computing – The Semantic Web meets Pervasive Computing -.” In D. Fensel et al. (Eds.), “The Semantic Web - ISWC 2003,” the Second International Semantic Web Conference (ISWC 2003), Sanibel Island, FL, USA October 2003 Proceedings, LNCS 2870, 2003, pp. 866-881.

[2] Masuoka, R., Labrou, Y., Parsia, B., and Sirin, E., “Ontology-Enabled Pervasive Computing Applications,” IEEE Intelligent Systems, vol. 18, no. 5, Sep./Oct. 2003, pp. 68-72.

[3] Song, Z., Labrou, Y., and Masuoka, R., “Dynamic Service Discovery and Management in Task Computing,” MobiQuitous 2004, August 22-26, 2004, Boston, USA, pp. 310 - 318.

[4] Task Computing Home Page, http://taskcomputing.org

[5] Lalana Kagal, Tim Finin, and Anupam Joshi, “Trust Based Security for Pervasive Computing Enviroments”, IEEE Communications, December 2001.

[6] Lalana Kagal, “Rei : A Policy Language for the Me-Centric Project”, HP Labs Technical Report, 2002.

[7] Jefferey Undercoffer, Filip Perich, Andrej Cedilnik, Lalana Kagal, Anupam Joshi, Tim Finin, “A Secure Infrastructure for Service Discovery and Management in Pervasive Computing”, The Journal of Special Issues on Mobility of Systems, Users, Data and Computing, 2003.

[8] Lalana Kagal, Tim Finin, and Anupam Joshi, “A Policy Language for Pervasive Systems”, Fourth IEEE International Workshop on Policies for Distributed Systems and Networks, 2003.

[9] Lalana Kagal, Tim Finin and Anupam Joshi, "A Policy Based Approach to Security for the Semantic Web", Second Int. Semantic Web Conference (ISWC2003), Sanibel Island FL, October 2003.

[10] Lalana Kagal, "A Policy-Based Approach to Governing Autonomous Behavior in Distributed Environments", Dissertation, September, 2004.

[11] Grit Denker, Lalana Kagal, Tim Finin, Massimo Paolucci, and Katia Sycara, "Security for DAML Web Services: Annotation and Matchmaking", Second Int. Semantic Web Conference (ISWC2003), Sanibel Island FL, October 2003,

[12] Lalana Kagal and Tim Finin, "Modeling Conversation Policies using Permissions and Obligations", AAMAS 2004 Workshop on Agent Communication (AC2004), July, 2004,

[13] Pranam Kolari, Lalana Kagal, Anupam Joshi, and Tim Finin, "Enhacing P3P Framework with Policies and Trust", UMBC Technical Report and under review, 2004.

[14] Anand Patwardhan, Vlad Korolev, Lalana Kagal, and Anupam Joshi, "Enforcing policies in Pervasive Environments", International Conference on Mobile and Ubiquitous Systems: Networking and Services, 2004.

[15] Lalana Kagal, Massimo Paolucci, Naveen Srinivasan, Grit Denker, Tim Finin, and Katia Sycara , "Authorization and Privacy in Semantic Web Services", IEEE Intelligent Systems (Special Issue on Semantic Web Services), 2004.

[16] S. Godik and T. Moses, "OASIS eXtensible Access Control Markup Language (XACML)", OASIS Committee Secification cs-xacml-specification-1.0, November 2002.

[17] A. Uszok, J. Bradshaw, P. Hayes, R. Jeffers, M. Johnson, S. Kulkarni, M. Breedy, J. Lott, and L. Bunch, "DAML reality check: A case study of KAoS domain and policy services", International Semantic Web Conference (ISWC 03), Sanibel Island, Florida, 2003.

[18] A. Uszok, J. M. Bradshaw, R. Jeffers, M. Johnson, A. Tate, J. Dalton, and S. Aitken, "Policy and Contract Management for Semantic Web Services", AAAI Spring Symposium, First International Semantic Web Services Symposium, 2004.

[19] Arif Ghafoor, James B. D. Joshi, Rafae Bhatti, and Elisa Bertino, "XML-Based Specification for Web Services Document Security", IEEE Computer Society Press , 2004.

Describing the P3P base data schema using OWL

Giles Hogben,
European Commission,
Joint Research Centre,
Via Enrico Fermi 1,
21020 VA, Ispra, Italy
+39 0332789187
giles.hogben@jrc.it

ABSTRACT

This paper describes use cases and requirements for a privacy policy data schema. It describes problems with existing schemas in relation to these requirements (P3P 1.0, P3P 1.1 and RDFS schema for P3P). It proposes and motivates the use of an OWL schema to describe the same semantics, which fulfils all the requirements and may be used in a semantic web based privacy and identity management context. It describes the advantages which this gives to a policy evaluation engine based on such a schema and describes some of the reasoning use cases addressed in modelling the schema.

Modelling the schema using OWL appears simple at first sight, because the entire schema can be constructed with OWL-Lite predicates or using one custom predicate. However, the fact that modal logical statements must be made about data types in the schema (e.g. Organization x May Collect Data of type Y) makes reasoning over the typing schema challenging. The paper also looks at syntactic and semantic validation using the schema as well as extensions and modifications to the vocabulary items supported.

Categories and Subject Descriptors

D.3.3 [**Programming Languages**]: Language Constructs and Features – *abstract data types.*

E.1 **[Data Structures]:** *Distributed data structures, graphs and networks.*

General Terms

Standardization, Languages, Theory

Keywords

Policy engineering, P3P and policies, Semantic Web Groundings

This work was supported by the IST PRIME project; it represents the view of the authors only.

1. Introduction

P3P [1] is a policy framework for describing web site privacy practices using XML. The main body of a P3P policy is made up of a set of statements about data collection practices. Each statement refers to the practices claimed for a certain type of data, described by "data elements" which are typed and validated according to a special P3P data schema. The vast majority of existing policies use the P3P Base Data Schema [2], the base typing schema provided by P3P for this purpose. The exact specification of this schema is outlined in [3] .

The P3P 1.0 base data schema is intended to provide a base set of data types to cover the most common categories of personal data about which P3P privacy policy statements might be made. The schema also provides extensibility mechanisms for expressing custom types. The fact that it is one of the only mechanisms to offer this functionality for such a broad range of data types has meant that the P3P data schema has also been adopted for several other use-cases which were unforeseen by the P3P working group.

This paper shows how an OWL [4] based semantics can be used in these use cases to fulfil many of the requirements that are problematic for the P3P base data schema. The schema is designed to fit into the policy architecture framework proposed in "P3P Using the Semantic Web (OWL Ontology, RDF Policy and RDQL Rules)" [5]

One important problem resolved by this paper is that P3P makes statements about data types in the schema which use modal logic (e.g. Organization x *May* Collect Data of type Y). This makes reasoning over the typing schema challenging. The paper presents a solution for achieving this using available OWL reasoning tools.

2. Use cases

Our motivation for creating an OWL data schema for privacy policy languages was broader than the usage scenarios envisaged for P3P and our schema is designed to cover scenarios envisaged for both P3P and nascent enterprise privacy management standards such as EPAL [6] and the technologies being developed by the PRIME project [7], as well as to satisfy identity management requirements such as those for automated form filling and pseudonym management. In practice, the P3P data schema has already been used beyond its design remit in many projects [7],[8],[9] and it is therefore an urgent need to provide a schema which can satisfy these broader requirements.

The schema we propose should allow the description of data types in the following policy contexts:

a. Requesting data or credentials (the auto-form filling/Xforms [10] scenario). The data typing schema is used to describe the type of data to be inserted into a form field.

b. Describing data or credentials (metadata). The data typing schema is used to describe data or credential instances.

c. Describing data practices (P3P type scenario) according to data types. The data typing schema is used to describe types of data to which certain data handling practices may be applied.

d. Application of access control rules. The schema is used to describe types of data to which groups of access control rules should be applied. For example it should be able to describe the type used in the natural language rule: "Do not give user emaill addresses to third parties".

3. Requirements on a privacy and Identity Management data schema

An analysis of the above use cases has led to the following specific requirements:

1. Data types must describe data (i.e. the object is the information), not properties of individuals. This is needed to allow for types of data which are personal but do not necessarily apply to individuals. It is also correct semantically as data handling policies for example, make statements about data and not about individuals and their properties.

   This implies that data types must be modelled as classes rather than properties. So for example "email" means "data of type email" rather than "the email property of user x". This allows the model to be centred around statements about data collection practices rather than statements about individuals and their properties. It is more difficult to use OWL to provide meta-information about properties than it is about classes. The semantics of properties also breaks down when it comes to data types such as "user". If user is a property, what does it refer to? [11] breaks the schema down into classes and instances so that "user" is a class, while "prefix" is the value of a property, but this seems unnecessarily complex as all the types in the P3P schema can be described as classes of data.

2. The schema should distinguish between abstract (cannot be instantiated) and concrete types. This gives the possibility to use the schema for data and credential requests such as automated form filling. It is not possible to use the P3P base data schemas for automatic form requests because it does not satisfy this requirement. But if types are designated abstract and concrete status, then an application can ask for say "user, online data" and a reasoning engine can drill down the schema to dig out the concrete types "home page, email address etc…

3. It should be easily possible within the semantics to apply meta-data both to instances of data types and to the types themselves. This requirement is derived from the need both to describe data literals, and to make statements about classes of data when describing data handling practices. This is another strong reason to model data as classes and not as properties, because it is much more natural to apply metadata to classes rather than to properties.

4. The schema should be able to describe both literals (data submissions) and classes of data.

5. The semantics of the OWL base data schema should not conflict with any semantics which can be inferred from the P3P Base Data Schema unless this can be shown to be inconsistent with other requirements. The vocabulary used in the P3P Base Data Schema semantics is based on a standards process and thereby represents a consensus on the actual data types required for describing most data. Although the syntax and semantics is poorly expressed, the actual taxonomy represented has considerable value.

6. The number of classes defined should be minimized. As with any data model, redundancy is to be avoided and the description of classes should be as normalized as possible.

7. The schema should provide validation functionality for allowed data types and for the syntax of instances of a designated type. If the schema is to be used for typing instances, it is natural to provide syntactic validation functionality.

8. The schema should use standardized, well-defined syntax. In order to foster adoption.

9. The schema should have a well-defined semantics. This makes it easy to apply the schema to new use cases.

4. Existing data schemas in relation to requirements

4.1 P3P1.0 base data schema

Some literature exists outlining problems with the Base Data Schema [11],[12]. [12] cites the over complexity of the syntax and proposes an XML schema version of the syntax which has now been incorporated into the P3P1.1 working draft [13].

In relation to the above requirements, the P3P1.0 data schema has the following specific problems:

1. (Requirement 2) It does not distinguish between abstract and instantiatable types.

2. (Requirement 7.) There is no provision for validation of instance data.

3. (Requirement 8) The schema uses a highly complex and obscure custom syntax which:

   a. Does not use mechanisms available in XML syntax, which are commonly used to model semantics. For example it does not use nesting to indicate subclass or other class relationships, but rather a convoluted custom syntax involving string matching.

   b. Is not well defined – the syntax used for defining the relations between allowed data types can only be deduced by examining the base data schema and examples. It does not follow directly from the specification document. To take one example out of several:

   Data Structures are abstract types (for example "POSTAL") which appear in the schema, but are never actually allowed as types in data elements. They serve to group concrete elements together. Nowhere in the specification document is it stated that in a data schema, data structures refer to their child elements by parsing the data element name, splitting it by "." delimiters and then taking the first token!

Another example is that, according to [1], the categories of the data schema (broad classes of data types) follow a "bubble-up rule". The meaning of this phrase is not precisely explained in the P3P specification, but by examining the base data schema, one can deduce that it means data types which can be expanded into further structures must inherit any categories which are valid for those structures. In fact,

however, not all the categories quoted in the P3P base data schema do follow a "bubble-up rule". For example, the postal.name data structure is not (according to the official specification [2]) assigned to the category demographic of its child data structure, personname prefix.

Many of these problems were not picked up because the syntax is so obscure.

4. (Requirement 9) The semantics is also not well defined. There is a confusion between classes of data and properties of individuals. For example, "user.employer" :"Name of User's Employer" seems to model an object (the user) and its properties. But "dynamic.cookies" "Use of HTTP Cookies" models an abstract class of data (dynamic.cookies) and not the "cookies" property of a "dynamic" object. Furthermore the specification does not define whether syntax such as "user.email" is meant to represent a set of user's email addresses – or the intersection of the class of user data with the class of email data. This has important implications when trying to describe instance data.

   Furthermore the semantics of the dot relationship between the data types is not made clear. The specification says that elements "include" other elements, implying that the relation is equivalent to "subclass" but elements are also included by several disjoint classes, making this incoherent. It is one of the aims of this paper to make clear the exact semantics of the base data schema in order to model it using OWL.

4.2 The P3P 1.1 Data Schema

The P3P 1.1 Data Schema (still in draft at the time of writing) [13] addresses some of the problems outlined in 4.1

a. The P3P 1.1 Data Schema prescribes a standardized XML syntax for describing the relationships between data elements. Abstract "data structures" are abandoned, and relationships are described simply by nesting tags within each other. Custom schemas can be created by referencing another XML schema.

b. A more precise semantics for the elements can also be derived from the specification of this document. That " for an element `<B>` to be defined as an allowed child of element `<A>` means if the policy states that it may collect data of type `<A>`, *then it can also be taken to state that it may also collect data of type* `<B>`*.* "

The use of XML rather than description logic syntax is however fundamentally limited because

a. XML semantics is only informal and is based on a questionable interpretation of its syntax.

b. In practical terms, semantics expressed using a custom interpretation of XML syntax such as in the P3P 1.1 Data Schema cannot be interfaced with reasoning engines in the way that RDF + OWL can. Much of the utility of the data schema is lost because reasoning is proceduralized in program code which then cannot be reused.

c. Since the structure of the schema is not well suited to representation as a tree (as opposed to a directed graph), a custom syntax has to be used to represent the structure.

4.3 The RDFS Schema for P3P

[14] is a previous attempt at producing a P3P data schema using Description Logic syntax (RDFS). The RDF Schema for P3P models data types as properties and describes a different class for every possible combination of basic data types. While it does provide a well-defined "p3p:extends" relation between data types, it also describes all possible properties created by this extension relation. This is highly redundant as the extension relation is then contained in the syntax of the class names. It also has over 350 classes of data instead of less than 80 classes which are used to compose these.

Furthermore, the definition of the extends relation as "Extends another dataElementComponent" suggests a parallel with object oriented design, which is not consistent with the semantics. (Does a user's email extend the properties of a user?).

Finally, the RDFS schema's use of properties rather than classes does not fulfil requirement 1.

5. Modelling Class Relationships in OWL

OWL provides a syntax which fulfils all the above requirements. In using OWL, we implement the base data schema semantics in the context of a semantic web enabled privacy architecture as described in [5]. We chose OWL instead of other object oriented modelling languages because it gives a standard XML based syntax which provides the functionality required by the semantic web based architecture in which the schema is used.

5.1 Reasoning use cases

We begin by describing a reasoning use case and then go on to show how this can be implemented using an OWL-based semantics which accurately reflects the intended semantics of the P3P1.0 base data schema.

Identity management and access control systems typically need to reason over policies or requests for broad data types which correspond to specific data types in a store. Some important reasoning use cases are as follows:

5.1.1. A typical statement of collection practices specifies that the service may collect any data which is in both User and Name classes (i.e. specializing Name as a User, not a Business, name)

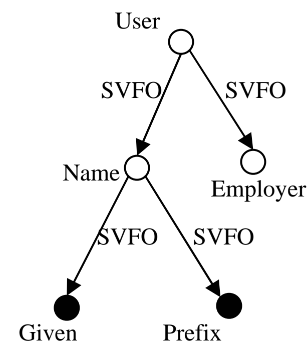


Figure 1. Classes related by SVFO

The diagram represents this scenario. The reasoner is required to deduce that this implies that the service may also collect the data

classes Given and Prefix (concrete types are filled in black, inferences dotted lines).

5.1.2. A policy states that a company collects any data of type User, whereas a preference rule refers to protecting Online data. The reasoner needs to infer that if a service might collect User data, it might also collect Online data.

5.1.3. A policy gives sensitivity ratings to data types which determine their release by an identity management policy. The reasoner selects the type with the maximum or minimum rating in a given context.

Formally speaking, 5.1.1 and 5.1.2 require a system of modal logic since it is describing possibilities. However, we show below that it is possible to produce the required entailments using an ordinary propositional logic system such as prolog.

5.2 Modelling the entailments using the structure of the P3P 1.0 Data Schema

The P3P1.0 specification states: "*P3P1.0 Data elements are organized into a hierarchy based on the data element name as specified by the data schema. A data element automatically "includes" all of the data elements below it in the hierarchy. For example, the data element representing "the user's name" includes the data elements representing "the user's given name", "the user's family name", and so on. Thus the data elements user.name.given, user.name.family, and user.name.nickname are all children of the data element user.name, which is in turn a child of the data element user.*"

It is important to note that the exact meaning of "includes" here is not specified. It appears to mean "subclasses" but, if one examines the structure and semantics of the schema, this cannot be the case because data elements such as personname are used as part of disjoint classes such as User and Business.

Data schemas often need to reuse a common group of data elements. P3P 1.0 data schemas support this through named data structures. A data structure is a named, abstract definition of a group of data elements. The name of the data structure itself (e.g. postal) is never actually used in a data element. We quote the P3P 1.0 Specification's example:

```
<DATA-STRUCT name="date.ymd.year"
   short-description="Year" />
<DATA-STRUCT name="date.ymd.month"
   short-description="Month"/>
<DATA-STRUCT name="date.ymd.day"
   short-description="Day"/>
```

The structure of the P3P base data schema is, as [11] correctly points out, not a forest, but a semi-lattice, as elements are used repeatedly in different contexts. Figure 2 below is a Venn diagram showing a fragment of the schema classes, which illustrates the relation that holds between the data elements. The figure shows the Classes User and ThirdParty, which both include some (>1) values from Cert, Personname, Bdate and Gender.

All data elements in the P3P base data schema which are "included" are in fact related as shown. That is if A "includes" (B and C) then A contains some values from B and some values from C and no other values unless otherwise stated (note that in fig 2, User is shown outside of Cert, Personname etc… because it also "includes" other data elements.)

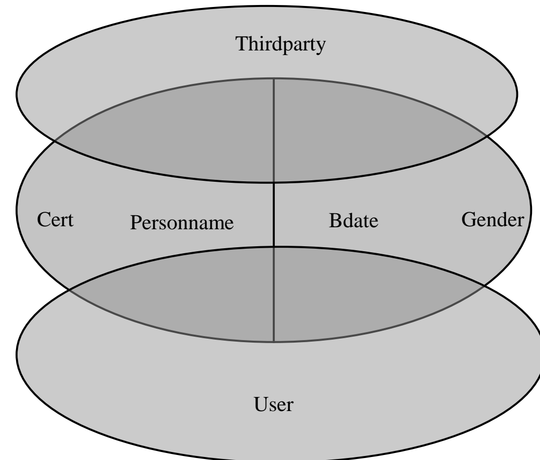


Figure 2: Fragment of schema classes as Venn Diagram

5.3 OWL semantics

If we model all data elements as classes of data (as shown in figure 2), then a single relationship, "SomeValuesFromOnly" can be used to define the entire P3P base data schema using OWL.

In formal set theoretic notation, then, we wish to express a relation R between three classes A, B and L (as shown in figure 2), where L is an RDF collection of classes:

If A <R> L then,

$$\forall li \in L, A = \bigcup (A \cap li)$$

and

$$\forall li \in L, A \cap li \neq \{\};$$

(A is the union of the intersection of A with each member, li, of L, and no intersection is null). Or alternatively, in terms of class members,

$$\forall l \in L, \exists i (i \in (A \cap l))$$

and

$$\neg \exists a, a \in A, \forall li \in L, a \notin l$$

Informally, this means that if A <R> L, where L is a list of classes, then A is made up of some values from every class which is a member of L and no other values. For example suppose L is made up of Login, Name, Bdate and Gender. Then suppose we state that (User <R> L), then User is made up of Login data, Name data, Bdate data and Gender data. Note that the Venn diagram does not show all the classes in User and therefore User has some values not in Login, Name, Bdate or Gender. Note however, that these classes are *not* subclasses of User data as they also share members with other classes which are disjoint from User.

We found that the relation <R> can in fact be expressed in OWL-Lite using the following syntax:

```
<owl:Class rdf:ID="A">
<owl:equivalentClass rdf:parseType="Collection">
<owl:Restriction>
<owl:onProperty rdf:resource="&rdf;type" />
<owl:someValuesFrom rdf:resource="#B" />
</owl:Restriction>
<owl:Restriction>
<owl:onProperty rdf:resource="&rdf;type" />
<owl:someValuesFrom rdf:resource="#C" />
</owl:Restriction>
</owl:equivalentClass>
</owl:Class>
```

This uses a restriction on the property "type" to say that some the class A is made up of some values of type B and some values of type B. The equivalent class predicate ensures that there are no other values included. Using this syntax, in combination with rules defining typical inferences to be made over the class graph for various policy predicates, we found that all the necessary deductions can be made. In order to increase reasoning efficiency, we decided to abbreviate the above syntax to the equivalent syntax:

```
<owl:Class rdf:ID="A">
  <customNS:SVFO rdf:parseType="Collection">
        <B/>
        <C/>
  </customNS:SVFO>
</owl:Class>
```

(SVFO stands for "Some Values from Only", which is an abbreviation of the relations expressed in OWL-Lite above applying to a list of objects)

These two syntaxes are equivalent and the second is only a performance enhancement. We do not therefore specify the use of one syntax preferably to the other if performance considerations are addressed in some other way (e.g. introducing custom procedural code into reasoning engines).

Furthermore, as in P3P1.1, we have also removed the "data structure" names such as "postal" which are never referred to and therefore complicate the schema unnecessarily. Structural information can be included in labels if required for readability.

The whole schema hierarchy is then modelled using relations such as:

```
<owl:Class rdf:ID="User">
  <customNS:SVFO rdf:parseType="Collection">
        <Personname/>
        <Cert/>
        ……
  </customNS:SVFO>
</owl:Class>
<owl:Class rdf:ID="Personname">
  <customNS:SVFO rdf:parseType="Collection">
        <Given/>
        <Prefix/>
        ……
  </customNS:SVFO>
</owl:Class>
```

Note that the categories of the base data schema can also be modelled using this syntax, since they are just another class to which some of the other data types stand in relation SVFO. The syntax for integrating categories is more succinct and readable than other syntaxes because it is only necessary to list the categories and their allowed SVFO relations. The categories then stand as an orthogonal system to the main hierarchy of types.

For example,

```
<owl:Class rdf:about="#Political-data-category">
   <customNS:SVFO rdf:resource="#Cookies"/>
   <customNS:SVFO rdf:resource="#Miscdata"/>
   <rdfs:subClassOf rdf:resource="#Categories"/>
</owl:Class>
```

6. Concrete and abstract types

Many applications need to know whether a data type can be instantiated or not. For example if an application requests "User data", this cannot be instantiated and the application must first derive the concrete types inferred from the request. For this reason, all concrete classes are designated as type Instantiatable. If a type is not designated as instantiatable, then it is assumed to be abstract.

7. Shortcut classes

In order to abbreviate the syntax of typing instance data, we provide a set of shortcut classes for all possible instantiatable classes. For example for data of type User, Name and Given, the RDF syntax for typing an instance would be very verbose, so we define the class

```
<owl:Class rdf:ID="User.Name.Given">
 <rdf:type rdf:resource="#Instantiateable"/>
 <owl:intersectionOf rdf:parseType="Collection">
 <owl:Class rdf:about="#User"/>
 <owl:Class rdf:about="#Name"/>
 <owl:Class rdf:about="#Given"/>
 </owl:intersectionOf>
</owl:Class>
```

These classes do not add anything to the semantics of the ontology, but make it quicker and easier to type instance data and to reason over the type ontology.

8. Referencing the schema from privacy policies

There are 3 main use cases for referring to types from the schema expressed using this syntax

1. Requesting a type – in a privacy negotiation between an access control system and requester, the access control system may require information or credentials. It therefore needs to send hints as to the credentials required. For example a web service may require a certain certificate in order to allow access to a client. In this case, the service must be able to provide hints to the client as to what is needed to get authorization to use the service.

This can be expressed using the following syntax

1. Requesting typed data (Entity requests the data specifying the user's name).

```
<Entity>
<requests-data-types>
<rdfs:Class>
        <rdfs:subClassOf rdf:resource="User"/>
        <rdfs:subClassOf rdf:resource="Name"/>
</rdfs:Class>
</requests-data-types>
</Entity>
```

```
(Or shorcut class syntax can also be used – see
Sec 7.)
```

2. Typing an instance (Entity Submits data of type User's Given Name). This is expressed using the following syntax:

```
<Entity>
<hasData>
  <User.Name.Pseudonym>
       <rdf:value>Pseud1</rdf:value>
  </User.Name.Pseudonym>
</hasData>
</Entity>
```

3. Describing a practice carried out on a data type (Entity collects any values which are of type User and Name i.e. the class which is the intersection of both these classes)

```
<Entity>
<collectsAny rdf:parseType="Collection">
<rdfs:Class>
       <rdfs:subClassOf rdf:resource="User"/>
       <rdfs:subClassOf rdf:resource="Name"/>
</rdfs:Class>
</ collectsAny>
</ Entity>
```

The following points are worth noting in relation to this syntax:

- Each of these descriptions uses a different semantic to describe the operation on the data, but the data types are always referred to in terms of classes from the ontology. That is using rdf:type or rdfs:subClassOf.
- In order to express a specific type, it is often necessary to use multiple type declarations. For instance a name may be a User name or a Business name so in the data request description, it is declared as being both of type User and Name to make clear this specialization.
- As discussed in section 7, the predicates "collectsAny" and "requests-data-type" are in fact modal predicates and effectively convert DataClassX into a prototypical class representing all possible classes satisfying the subClass properties. This somewhat contradicts the formal semantics of OWL, however it will be shown that the correct deductions can still be derived using prolog style rules to extend the OWL semantics.

9. Inferencing over the schema

There are many possibilities for customized reasoning over such a schema, as discussed in section 5.1. We discuss below how the reasoning use case 5.1.1 (and implicitly also 5.1.2) may be implemented. These cases are key to each of the policy use cases described in section 2. This solution has been implemented and tested using the Jena API [15].

9.1 Deriving types of data collected or requested from broad types.

The relation "SVFO" (someValuesFromOnly) defined above specifies a directed graph which has a tree structure. Figure 3 represents a typical statement about collection practices as described in 5.1.1 (expanding the possible types of Name data).

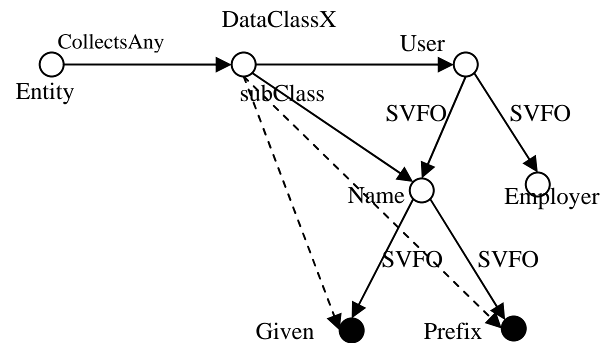


Figure 3: Data Handling Directed Graph

The policy states that the service may collect any data which is in both User and Name classes (i.e. specializing Name as a User, not a Business, name) – we give the elements of this class the temporary name DataClassX. The reasoner then is required to deduce that "Service may collect DataClassX" implies that the service may also collect the data classes Given and Prefix (concrete types are filled in black, inferences dotted lines).

Generally speaking, if X mayCollect DataClassX where DataClassX is some subClass of A (This is a typical statement from a P3P policy), then if A <SVFO> B and B <SVFO> C then it is also true that:

X mayCollect DataClassX where DataClassX subClass of C

The property mayCollect is a modal predicate and effectively converts DataClassX into a prototypical class representing all possible classes satisfying the subClass properties. However the correct deductions can still be derived using prolog style rules.

There are 3 important problems that a reasoner must address in this context:

1. The reasoner must return that DataClassX (the class of data which may be collected) is a subClass only of the lowest node in the SVFO hierarchy which DataClassX is already a subclass of. In the example then, the reasoner must not deduce that the service may collect Employer data. Therefore it cannot simply return that DataClassX is a subclass of – (all classes to which User and Name stand in relation SVFO to). I.e. the reasoner must only expand the graph below Name but not below User.

Using a typical OWL reasoner, this is impossible because it involves a form of negation (or "Unsaid within the context"). In order to determine to expand the SVFO child nodes of Name and not User in the above example, it must determine that Name does not have any SVFO children of which DataX is already a subclass. This can only be done using a forAll(not…) type rule functor.

2. The conclusion (DataClassX subClass Given) invalidates any rule premises searching for Unsaid((Name SVFO ?x), (DataX subClass ?x)). This means that the conclusions invalidate the premises, which is nonmonotonic reasoning. This is solved using marker triples, which tell the reasoner to ignore conclusion triples. This is a workaround which forces the reasoning to be monotonic.

3. Finally, transitivity does NOT hold for SVFO. It is not always true that if X has some values from Y and Y has some values from Z, then X has some values from Z. For example if (User has some values from Only Name and Employer) and (Name has

some values from only Given and Prefix), we cannot deduce that (User has some values from Prefix), because the some values that User has from Name may not be any of those that Name has from Prefix.

What we need from such a property however is the following:

If and only if a service may collect any data of classes Given and X, and Given is in the relation SVFO to class X, where X is relation SVFO to class Y then the reasoner should also return that the service might collect any data in class Y. (a kind of conditional transitivity for SVFO).

All these requirements can be met within the limits of acceptable performance using the proposed OWL ontology in combination with prolog style rules. We used the Jena inference libraries to derive these inferences on a sample policy. The following two Jena rules correctly expand the types based as described above (we have abbreviated the name space declaration for brevity). The question mark syntax indicates universal quantification and all triples are ANDed in the premises and conclusions:

Rule 1. The following complex rule ensures that the reasoner deduces that A is a subclass of SVFO child nodes of any class, X such that N requests-data-types A and A subClassOf X where there is no class Y such that X <SVFO> Y (problems 1 and 2.)

```
[(?N ns:mayCollect ?A),
(?A rdfs:subClassOf ?X),
unSaidSpecial(?A,ns:someValuesFrom,rdf:type,?X)
->
[r3:(?A rdfs:subClassOf ?E)
(?X rdf:type ns:marker)
<- (?X ns:someValuesFrom ?E)]]
```

Rule 2. The following rule ensures that all SVFO children of a class are returned as being of the same as the policy node, as long as they have been previously marked using the second rule (problem3).

```
[(?A rdfs:subClassOf ?D) <-
(?A rdfs:subClassOf ?B)
(?B rdf:type ns:marker)
(?B ns:someValuesFrom ?C)
(?C ns:someValuesFrom ?D)
]
```

The Rule builtin, unSaidSpecial provides the required negation functor described above (problem 1) and is defined as follows:

```
unSaidSpecial(A,P,Q,X)
```

True iff for all(Y), (X,P,Y) there is no triple st (A,Q,Y)

Note that using the shortcut classes (see sec 7.), this reasoning step can be performed much more simply for the case of finding instantiatable types, however for the case of matching preferences without the benefit of shortcut classes, this reasoning is still necessary.

10. Validation using the OWL format

As well as reasoning functionality, most applications require some validation functionality. This is of two kinds:

10.1 Synactic validation

This is available for concrete types such as "email". For example the schema can specify that the concrete type email must contain an @ sign – this can then be used to validate form entries for example. This is achieved simply by specifying the rdfs:range of instantiatable types described by the schema as being over an xsd datatype.

```
e.g.
<rdfs:range rdf:resource="&xsd;dateTime"/>
```

This example shows a builtin data type. OWL does not specify a mechanism for referencing user-defined xsd data types, but it does not prohibit their usage. The OWL specification has this to say about the question of user defined XML schema datatypes:

"*Because there is no standard way to go from a URI reference to an XML Schema datatype in an XML Schema, there is no standard way to use user-defined XML Schema datatypes in OWL.* "

If we specify a mechanism for referring to custom data types in a resource, we are therefore able to define a namespace containing syntactic validation constraints on the concrete types for the OWL data schema.

For example the following could be used to validate an email address:

```
<rdfs:range rdf:resource="&PII-DS-
XML;emailAddress"/>
```

Can be specified to refer to the simpletype in the schema as follows:

```
<simpleType name='emailAddress'>
     <restriction base='duration'>
      <pattern value='\w*@\w*\.\w*((\.\w*)*)?'/>
     </restriction>
</simpleType>
```

10.2 Semantic validation

A data type assignation breaks semantic validation rules if it refers to a type of data which cannot exist. OWL is not a language which is well adapted to making negation based statements of this kind. However, we have added disjointness relations for classes which should not be assigned simultaneously to data types (i.e. they have no common values). For example if a policy describes a class which is a subclass of both User and Business this should be flagged as invalid. More sophisticated semantic validation constraints may be added later. for example, a user's login can have only one value. This may also involve the use of custom rules within the reasoner module.

11. Changes to the P3P data schema vocabulary

Based on input from other researchers, we have also altered the available classes of the P3P data schema. For example, the following alterations have been made.

1. Name is a single class rather than dividing it into user name and business name. It is then specialized using business and name classes.
2. We have added classes corresponding to fields in electronic credentials,for example electronic identity card, drivers' licence and passport fields.

3. We have taken into account recommendations on identity document fields given in the recent ICPP study on identity management systems [16].
4. The techniques used to model credential metadata have also added other classes and predicates, which are out of the scope of this paper. For example we have added classes for describing proof methods for assertions made by credentials which fit into the typing schema.

12. Conclusion

OWL can be used to satisfy the requirements on data schemas for privacy and identity management policies within and beyond the use case scenarios of P3P. Some modification of the rulebase for reasoning over OWL is needed to deal with the modal "may collect values from" and "requests values from" predicates required by these scenarios, but this is possible using standard semantic web libraries. OWL data schemas can also provide required type validation functionality.

13. REFERENCES

[1] Platform for Privacy Preferences Specification, Cranor et al. ,Platform for Privacy Preferences, W3C Recommendation, http://www.w3.org/tr/p3p

[2] *P3P Base data schema*, part of W3C Recommendation on P3P, http://www.w3.org/TR/P3P/base

[3] Cranor et al. *P3P Base Data Schema specification*, http://www.w3.org/TR/P3P/#Data_Schemas

[4] *Web Ontology Language*, W3C recommendation, see http://www.w3.org/TR/owl-semantics/

[5] Giles Hogben, *P3P Using the Semantic Web (OWL Ontology, RDF Policy and RDQL Rules)*, W3C Working Group Note 3 September 2000, http://www.w3.org/P3P/2004/040920_p3p-sw.html

[6] Powers, C., Schunter, M., Enterprise Privacy Authorization Language (EPAL 1.2), W3C Member Submission 10 November 2003, http://www.w3.org/Submission/EPAL/ (for references to P3P data schema, see http://www.w3.org/2003/p3p-ws/pp/ibm2.html)

[7] Privacy and Identity Management in Europe, European Research Project, see http://www.prime-project.eu.org

[8] Claus,S. and Doring,S. *P3P Based Negotiation of Personal Data*, Technische Universitat Dresden, Fakultat Informatik, D-01062 Dresden, Germany

[9] Electronic Commerce Modeling Language (ECML), http://www.faqs.org/ftp/rfc/pdf/rfc3505.txt.pdf

[10] Eds Dubinko et al, *XForms 1.0*, W3C Recommendation 14 October 2003, http://www.w3.org/TR/xforms/

[11] E. Damiani, S. De Capitani di Vimercati, C. Fugazza, and P. Samarati: *Semantics-aware Privacy and Access Control:Motivation and Preliminary Results*, Proceedings of 1st Italian Semantic Web Workshop, 10th December 2004

[12] Giles Hogben , A technical analysis of problems with P3P v1.0 and possible solutions, Position paper for "Future of P3P" workshop, Dulles, Virginia, USA, 12-13 November 2002, http://www.w3.org/2002/p3p-ws/pp/jrc.html

[13] Cranor, Dobbs, Egelman, Hogben et al., *The Platform for Privacy Preferences 1.1 (P3P1.1) Specification W3C Working Draft* 4 January 2005 http://www.w3.org/TR/2005/WD-P3P11-20050104/

[14] McBride, B., Wenning, R., Cranor, L., *An An RDF Schema for P3P*, W3C Note 25 January 2002, http://www.w3.org/TR/p3p-rdfschema

[15] Jena open source semantic web libraries, http://jena.sourceforge.net/

[16] Independent Centre for Privacy Protection (ICPP) / Unabhängiges Landeszentrum für Datenschutz (ULD), Schleswig-Holstein and Studio Notarile Genghini (SNG), *Identity Management Systems (IMS): Identification and Comparison Study*.

Predicates for Boolean web service policy languages

Anne H. Anderson
Sun Microsystems Laboratories
Burlington, MA
Anne.Anderson@sun.com

ABSTRACT
Four of the web service policy languages that have been proposed as the basis for a new standard are based on Boolean combinations of predicates. This paper discusses why these types of policy languages are of interest to industry, proposes an abstract layering for them, and compares the predicate forms used by two of these languages.

General Terms
Standardization, Languages.

Keywords
web services, policy.

1. INTRODUCTION
At the W3C Workshop on Constraints and Capabilities for Web Services [1], various proposals for a standard language for use in expressing policies for web services were presented. Four of the languages presented were variations on Boolean combinations of predicates: the *Web Services Policy Framework (WS-Policy)* [2], the *Web Services Description Language (WSDL)* [3] with the addition of *compositors* [4], the *XACML profile for web services (WSPL)* [5], and a language outline from IONA Technologies [6]. These languages differ in the predicates that are used. In WS-Policy, the predicates are *Assertions* that return a Boolean result, but are not otherwise defined in the policy framework itself; *Assertion* definitions are to be provided as part of each domain-specific document that defines items to be controlled by a policy. In WSDL *compositors,* the predicates are WSDL Boolean *Features, Properties,* or nested *compositor* (Boolean operator) expressions; *Features* and *Properties* are not further defined, although some semantic guidance is provided. In XACML WSPL, the predicates are XACML [7] functions that return a Boolean result and operate on *Attributes* and literal values, where an *Attribute* may be a name/type/value triple or a node in an XML document identified by an XPath [8] expression. In the IONA outline, the predicates are simple XML elements, with at most a *Yes/No* parameter; the process of defining the elements to be used is not elaborated in the outline proposal.

All these languages must rely on some mechanism for associating policies with services or service elements. WS-Policy relies on *Web Services Policy Attachment (WS-PolicyAttachment)* [9]. WSDL relies on attachment points defined in WSDL itself. WSPL relies on a specified convention for the use of the XACML *Target* element. The IONA outline does not describe its mechanism.

This paper will discuss why these languages are of interest to industry, will propose an abstraction for the layering of functionality involved in such languages, along with the functions of each layer, and will compare the forms of two types of predicates, discussing their advantages and disadvantages.

Copyright © 2005 Sun Microsystems, Inc. All rights reserved.
WWW 2005, May 10--14, 2005, Chiba, Japan.

2. WEB SERVICE POLICIES
This section describes, from this author's industry point of view, how “web service policy” has come to be defined by industry and why these Boolean combination policy languages have been of interest to industry.

The proponents of these Boolean policy languages view “web service policy” as being focused primarily on those aspects of a service required to establish a connection and a session such that message exchanges can be initiated. This focus arises from the fact that these aspects of policy are almost universal among web services – they all need to establish mutually agreeable security and reliable messaging parameters, for example, and standards for such parameters already exist. The proponents recognize that more complex languages may be required for some application-specific policy negotiations, but before such negotiations can occur, communication must usually be established. It may also be necessary to identify candidate service providers from a large pool, and thus highly efficient policy matching is a primary goal. Access control (web and OS) and security parameter (IPSec) policies with these same constraints have been in production use for years, so the design of “web service policy” languages has tended to grow out of those models.

A standard language for addressing basic web service communication is urgently needed, so industry is looking for a solution that can be standardized quickly. The W3C Workshop's call for position papers included a basic test case that position papers were supposed to address. Several of the Boolean combination policy language proposals included concrete solutions for this test case. Rightly or wrongly, the fact that none of the semantic web language proposals addressed the specific use case did not lessen some industry skepticism about whether semantic web languages are ready for production use in this area.

3. POLICY USES AND PROCESSORS
In order to develop appropriate web service policy languages, it is important to understand how web service policies will be used and which components of a web services architecture will use them.

One important use is simply for a service provider to publish its policies. A service consumer can query the policies of a provider instance and dynamically configure itself to those policies. The policy processor in this case is the consumer service application itself. In addition to processing the policy expression, the consumer must implement any functionality necessary for conforming to the policy. The consumer must understand the semantics of the items controlled by the policy in order to implement this functionality.

A second important use is for a service to verify that communications and messages it receives conform to its own policy. A service may have an internal policy that is more complex or more complete than the policy it publishes publicly,

but any communication would usually need to satisfy at least the published policy. Verifying that a communication or message conforms to a given policy does not require that the verifier understand the semantics of the items controlled by the policy, but only that the verifier know how to match communication or message information against the policy.

A third important use is for determining a mutually agreeable policy between a service consumer and a service provider. This operation might be performed by a service broker that accepts service registrations and client requests for services, and matches consumers with providers where there is a mutually acceptable policy. The entity that determines the mutually compatible policy need not understand the semantics of the policy items, but only that it can determine the intersection between two policies.

Policies can be used in other ways not directly involving interactions between service providers and service consumers. An example would be the use of a policy for describing the values to be used in a particular deployment of a service. Such a policy might specify which of various options supported by the service are to be enabled, and with which values, in this particular deployment. In this case, the service application is the policy processor, and must understand the semantics of the policy items.

4. POLICY LAYERS

Several functional layers can be identified for such policy languages. The languages all require some underlying "vocabulary" that defines the items to be controlled by a policy, some mechanism for expressing predicates related to that vocabulary, a mechanism for expressing Boolean combinations of predicates, and a mechanism for associating the policy with a service or service element. The following diagram illustrates this layering, along with examples of where such layers are specified:

Table 1. Policy layers

Layer	Specification examples		
Vocabulary	WS-Security, WS-Reliability		
Predicate	WS-Security Policy, WS-Reliability Policy	XACML functions	undefined
Boolean combination	WS-Policy	XACML Boolean operators	WSDL compositors
Association	WS-Policy Attachment	XACML target	WSDL

Additional functions can logically be assigned to these layers. 1) An "`or`" of two predicates means that either predicate is acceptable, but at the time communication is established, one of the options must be selected. This suggests there should be a mechanism for specifying preferences among "`or`"d predicates, which would have to be specified at the Boolean combination layer. 2) Likewise, a single predicate may indicate that a range or set of values is acceptable for some item (e.g. "key length must be at least 1024 bits"), yet one value must be selected at the time communication is established. Preferences for these must be specified at the predicate layer. 3) A policy consumer needs to know the universe of items controlled by the policy and the defaults for items not included in the policy: Must there be a predicate for each item? Are unmentioned items prohibited or unrestricted? This functionality belongs at the Boolean combination layer. 4) Depending on how the defaults are specified, the predicate layer may need to provide predicates to indicate that a particular item is prohibited or is unrestricted. 5) In order to match policies, there must be a way to tell which predicates refer to the same underlying vocabulary item. 6) In order to determine if two policies are consistent, there needs to be a way to determine the set of values, if any, that satisfies each of two different predicates over the same vocabulary item.

The major difference between these Boolean combination policy languages is in the way the predicates themselves are defined. The other layers are functionally equivalent, although the syntax differences could affect the ease with which web service specifications can be associated with policies. Since neither the WSDL nor the IONA proposals describe their predicate layers in detail, the remainder of this paper will focus on WS-Policy and WSPL.

5. WS-POLICY

5.1 WS-Policy Overview

WS-Policy is a proprietary specification developed by a group of companies that includes IBM, Microsoft, and BEA. As of the writing of this paper, it has not been submitted to any standards body.

WS-Policy defines two Boolean operators - `<All>` (Boolean "and") and `<ExactlyOne>` (exclusive-or) - that may be applied to sequences of *Assertion* predicates. These operators may be nested. Previous versions of WS-Policy included a mechanism for providing hints about the policy writer's preferences among various alternatives, but this mechanism was omitted from the most recent version.

5.2 WS-Policy Predicate Layer

In WS-Policy, each web service specification must define a set of policy *Assertions* to be used in expressing policy predicates related to the vocabulary defined in the specification. For example, if the underlying vocabulary specification defines an XML schema element `<v:A>` that is to be controlled by web service policies, then there must one or more additional elements defined for use in expressing the policy predicates relating to `<v:A>`.

WS-SecurityPolicy [10], which defines the *Assertion* predicates to be used with the WS-Security [11] vocabulary, is the example used in the WS-Policy specification. Each new domain's vocabulary will require its own set of *Assertion* predicates, although the WS-Policy authors suggest that in the future, such *Assertions* will be defined as part of the underlying vocabulary specification – WS-Security and WS-Reliability are examples of legacy specifications for which external *Assertions* must be defined.

In comparing policies, conceptually each policy is first converted to Disjunctive Normal Form, such that the policies become sequences of acceptable alternative sets of *Assertions*. The intersection of two policies includes the "compatible policy alternatives (if any) included in both requester and provider policies. Intersection is a commutative, associative function that takes two policies and returns a policy." If the intersection is empty, the two policies are incompatible. A set of *Assertions* in one policy is compatible with a set of *Assertions* in another policy if each instance of an *Assertion* type in one policy is compatible with each instance of that *Assertion* type in the other policy. If an instance of a given *Assertion* occurs in only one set, then "the behavior associated with that *Assertion* type is

prohibited in the intersection of those policies", although this interpretation does not seem semantically consistent: if one policy requires encryption, and the other says nothing about encryption, then prohibiting encryption is not compatible with the first policy.

5.3 WS-Policy Predicate Processing

The specification that defines *Assertion* `<vp:A ...>` must Whether two instances of a given *Assertion* type are compatible is determined by the semantics defined in the domain-specific *Assertion* specification. The WS-Policy authors intend to provide guidance to *Assertion* developers on how to write *Assertions* that can be compared easily [12].

An *Assertion* may be a complex XML type. For example:

```
<vp:A attrB="..." attrC="...">
     <vp:D>example1</vp:D>
     <vp:E>25</vp:E>
     <vp:F attrG="..." />
</vp:/A>
```

The specification that defines *Assertion* `<vp:A ...>` must define all possible variations of this element that a service consumer might request, what the intersection of any two instances of this *Assertion* is, which combinations are not allowed, and how the various forms of the *Assertion* relate to acceptable instances of the underlying domain-specific vocabulary that is the subject of the policy. Any policy processor that must verify a message against or compare instances of `<vp:A ...>` must incorporate a code module that implements the semantics specified for `<vp:A ...>`.

6. WSPL

6.1 WSPL Overview

The syntax of WSPL is a strict subset of the OASIS eXtensible Access Control Markup Language (XACML) Standard. Additional semantics have been specified in the WSPL specification. A WSPL prototype has been implemented.

A WSPL policy is a sequence of one or more rules, where each rule represents an acceptable alternative. A rule is a sequence of predicates, all of which must be satisfied in order for the rule to be satisfied. Rules are listed in order of preference, with the most preferred choice listed first. A WSPL policy is in Disjunctive Normal Form, where the rules are logically connected with "OR" and the predicates within each rule are connected with "AND".

A more complete description of WSPL is contained in [13].

6.2 WSPL Predicate Layer

WSPL defines a standard language for use in specifying predicates that constrain domain-specified vocabulary items. WSPL predicates are XACML functions that return Boolean values. The parameters to the functions are XACML *Attributes* and literal values. An *Attribute* corresponds to a domain-defined vocabulary item. *Attributes* are referenced in two ways, depending on how the domain defines them. An *AttributeDesignator* references a vocabulary item using a domain-defined URI and a standard data type. An *AttributeSelector* specifies a vocabulary item using an XPath expression that selects the vocabulary item from a domain-defined XML document. This document is usually an instance of the schema that defines the domain vocabulary.

Each WSPL predicate places a constraint on the value of an *Attribute*. The constraint operators are: equals, greater than, greater than or equal to, less than, less than or equal to, set-equals, and subset. All the comparison operators are strongly typed and must agree with the data types specified for the function parameters. WSPL supports the rich set of data types used in XACML: string, integer, floating point number (double), date, time, Boolean, URI, hexBinary, base64Binary, dayTimeDuration, yearMonthDuration, x500Name, and rfc822Name. These data types are all taken from the XML Schema [14], with the exception of the two duration types taken from XQuery Operators [15], and the two name types taken from XACML.

6.3 WSPL Predicate Processing

In order to find the intersection of two WSPL policies, several steps are performed. First, the *targets* of the two policies must match (*Targets* are described more completely in [13]). If the *targets* do not match, then the two policies are not compatible. Second, a new policy is created in which there is one rule for each pair of rules from the original policies, where the new rule contains all the predicates from the two original rules. For any given set of vocabulary item values, this new policy will return "true" if and only if both original policies would return true, since the new policy retains all the constraints from the two original policies. WSPL rules are listed in order of preference in a policy: if one rule precedes another, then the policy owner prefers the combination of vocabulary item values specified by the first rule to the combination specified by the second rule. By default the entity that performs a policy intersection preserves the preferences of one policy completely, and the preferences of the second policy to the extent that those are consistent with the preferences of the first. More complex preference combining algorithms could be used, but there is always the possibility of preference conflicts, and the combining algorithm must have some mechanism for resolving these.

The next step is to merge the predicates in each of these new rules such that, for each vocabulary item referenced in the new rule, there is a single predicate (or two predicates in the case of a range of vocabulary item values bounded at each end) that will be true if and only if all predicates in the rule that reference that vocabulary item are true. WSPL specifies the computation of such predicates, based on the laws of arithmetic and logic, for every function operator and data type. For example, the two predicates "`Attribute A > Value B`" and "`Attribute A = Value C`" are both true if and only if "`Value B > Value C`" and "`Attribute A = Value C`". If "`Value B`" is not greater than "`Value C`", then the two predicates are incompatible, and thus the new rule can never be true and is eliminated from the new policy. After this step, each remaining rule is internally consistent: there are no conflicting predicates over the same vocabulary item. The two original policies are incompatible if and only if this resulting set of rules is empty.

The intersection of any two policies specified using the WSPL predicate language can be computed. Computing this intersection requires no knowledge of the semantics of the referenced domain-specific vocabulary items, but depends only on the semantics of the set of standard functions and data types. The resulting policy is in a form such that a policy user can select any rule, select values for each vocabulary item consistent with the predicates in that rule, and that resulting set of values will be acceptable to both original policies.

7. COMPARISON OF PREDICATE FORMS

Both these styles of predicate specification have their advantages and disadvantages.

A single WS-Policy predicate can control multiple related items in the underlying vocabulary; each WSPL predicate applies to only one item. We have designed an extension to WSPL, however, that allows predicates pertaining to related items to be grouped.

A WS-Policy predicate can be abstract. For example, one *Assertion* can state that a digital signature is required, without specifying any details about the syntax of that signature. This same *Assertion* could be used with multiple digital signature syntaxes. A WSPL predicate on the other hand, if it uses XPath expressions to reference actual nodes in an instance of the the underlying vocabulary schema, must depend on an actual node value that will be present in particular schema instances. This can make policies complex if there are multiple ways a particular requirement could be met in a schema instance (for example, there are multiple ways to reference an object to be signed in a message when using the XML Digital Signature standard). XACML name/type/value *Attributes* can be defined, however, to accomplish the same abstraction functions as WS-Policy *Assertions.*

The WS-Policy *Assertions* that need to be compared between two policies can be easily determined, because the *Assertions* will usually have the same name; there might be cases where two different *Assertions* might need to be compared, however, as when a consumer asserts a “MaximumBuyingPrice” *Assertion*, while a provider asserts a “MinimumSellingPrice” *Assertion.* Comparable WSPL *AttributeDesignators* can always be matched, because they must have the same name; similar “maximum” and “minimum” semantics are captured in the function operator rather than in the *Attribute* itself. If *AttributeSelectors* using XPath expressions are used, however, there may be multiple expressions that point to the same node in a schema instance. We are trying to define a subset of XPath that uniquely identifies each node to deal with this problem.

A WS-Policy *Assertion* can specify requirements on document creation, such as the requirement that information describing each document processing step be prepended to previous step information, thus allowing the steps to be “undone” in order by the message receiver. An XACML *Attribute* could be defined to express such semantics, but it can not be done with XPath expressions, since there is nothing in the document that indicates the order in which nodes were added. Note that this type of predicate can not be verified against a given message; it must simply be asserted as a requirement on a document processor.

In order to use a WS-Policy *Assertion* for message verification, the verification engine must include special code that knows how to relate that *Assertion* to a particular type of message. A WSPL predicate that uses XPath expressions can be used directly to verify that the predicate is satisfied in a message.

WS-Policy *Assertions* may be defined in proprietary specifications. Even if the specification is eventually standardized, there can be a long period during which the specification is under development and is not available to all implementers of policy processors. Particularly for policies related to application-specific vocabularies, there may be limited incentive to rush the policy specification to standardization. WSPL predicates, however, can refer directly to the underlying vocabulary specification, and the semantics of those predicates are standard and do not depend on the underlying specification. Alternatively, an XSLT can be used to translate information from an instance of a proprietary schema into a non-proprietary format such as XACML *Attributes* for use in specifying policies.

The Boolean operators defined in WS-Policy can be nested, resulting in a compact policy format; in order to process a policy, it must be at least nominally converted to Disjunctive Normal Form. In WSPL, the policies are always in Disjunctive Normal Form. This, along with the fact that functions are used to specify semantics, rather than having the semantics be implicit in the predicate itself, means that a given policy expressed in WSPL will almost always require more bytes for its expression than a corresponding WS-Policy policy.

From this author's industry perspective, the most significant difference between WS-Policy *Assertions* and WSPL predicates is that each *Assertion* has unique domain-defined semantics that must be captured in a code module incorporated into any entity that must process the *Assertion,* either to compare it or to verify it. Each new domain-defined set of *Assertions* requires that policy processors be updated to support those; any change to existing *Assertions* likewise requires processor updates. Any processor that has not been updated will not be able to process new or modified *Assertions,* making it less likely that policies will be interoperable between different platforms. As more and more *Assertions* are defined, the footprint and maintenance complexity of each policy processor increases. WSPL predicates, on the other hand, use a finite, standard set of functions that do not depend on domain-defined semantics. Any WSPL processor can process any WSPL policy, new or old, and regardless of whether the underlying vocabulary is defined in a proprietary specification or not.

As a proof-of-concept, this author has translated all the *Assertions* defined in WS-SecurityPolicy into WSPL. This exercise was successful in demonstrating that WSPL can handle the policy semantics of a real-life domain.

8. SUMMARY

The web service policy languages that use Boolean combinations of predicates differ primarily in the forms those predicates take. In WS-Policy, predicates are XML elements whose syntax and semantics are domain-specific, with each policy item or group of items having its own set of predicates. In WSPL, predicates are standard XACML functions over a reference to a policy vocabulary item and a literal value. Both forms have advantages and disadvantages. The primary advantage of the WS-Policy form is that predicates tend to be compact and easy to read. The primary disadvantage is that policy processors must be configured to support the syntax and semantics of each predicate type that will be used by any policy. The primary advantage of the WSPL form is that a standard policy processor is able both to compute the intersection of any two policies and to verify any message against a policy. The primary disadvantage is that predicates that directly reference nodes in a domain schema instance may be overly specific, although WSPL also supports the creation of new vocabulary items to express more abstract requirements. WS-Policy currently has no preference mechanism, and the semantics of missing predicates appears to be incorrect; WSPL allows policy alternatives to be ordered by preference. WSPL needs an XPath subset that can be used to uniquely identify a policy item.

9. REFERENCES

[1] W3C, *W3C Workshop on Constraints and Capabilities for Web Services*, http://www.w3.org/2004/09/ws-cc-program., 12-13 October 2004.

[2] J. Schlimmer, ed., *Web Services Policy Framework (WS-Policy)*, http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnglobspec/html/ws-policy.asp, September 2004.

[3] W3C, *Web Services Description Language (WSDL) 1.1*, W3C Note, http://www.w3.org/TR/wsdl, 15 March 2001.

[4] U. Yalcinalp, *Proposal for adding Compositors to WSDL 2.0,* http://lists.w3.org/Archives/Public/www-ws-desc/2004Jan/0153.html, 26 January 2004.

[5] T. Moses, ed., *XACML profile for Web-services*, http://www.oasis-open.org/committees/download.php/3661/draft-xacml-wspl-04.pdf, Working draft 04, 29 Sept 2003 (also known as "*Web Services Policy Language (WSPL)*").

[7] T. Moses, eds., *OASIS eXtensible Access Control Markup Language (XACML)*, OASIS Standard 2.0, http://www.oasis-open.org/committees/xacml, 1 February 2005.

[8] W3C, *XML Path Language (XPath), Version 1.0*, W3C Recommendation, http://www.w3.org/TR/xpath, 16 November 1999.

[9] C. Sharp, ed., *Web Services Policy Attachment (WS-PolicyAttachment)*, http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnglobspec/html/ws-policy.asp, September 2004.

[10] A. Nadalin, ed., *Web Services Security Policy Language (WS-SecurityPolicy)*, Version 1.0, http://msdn.microsoft.com/webservices/default.aspx?pull=/library/en-us/dnglobspec/html/ws-securitypolicy.asp, 18 December 2002.

[11] A. Nadalin, et al, eds., *WS-Security*, OASIS Standard 1.0, http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wss, 6 April 2004.

[12] J. Schlimmer, personal communication, 12 October 2004.

[13] A. Anderson, *An Introduction to the Web Services Policy Language (WSPL),* Proceedings of the Fifth IEEE International Workshop on Policies for Distributed Systems and Networks, Yorktown Heights, New York, 7-9 June 2004, pp. 189-192.

[14] W3C, *XML Schema Part 2: Datatypes*, W3C Recommendation, http://www.w3.org/TR/xmlschema-2/, 2 May 2001.

[15] W3C, *XQuery 1.0 and XPath 2.0 Functions and Operators*, W3C Working Draft 2002, http://www.w3.org/TR/2002/WD-xquery-operators-20020816, 16 August 2002.

Policy Management and Web Services

Tim Gleason
Oracle Corporation
224 Strawbridge
Moorestown, NJ 08054
tim.gleason@oracle.com

Kevin Minder
Oracle Corporation
224 Strawbridge
Moorestown, NJ 08054
kevin.minder@oracle.com

Greg Pavlik
Oracle Corporation
224 Strawbridge
Moorestown, NJ 08054
greg.pavlik@oracle.com

ABSTRACT
We maintain that the representation syntax of specific Web services policies is secondary to the general problem of policy management in the Web services space. We outline a broad view of the policy space in middleware systems, discuss emerging solutions for the Web services environment, and explain critical aspects of policy management that are required for taking Service Oriented Architectures (SOAs) to the next level.

Categories and Subject Descriptors
D.2.12 [**Software Engineering**]: Interoperability

General Terms
Management, Standardization, Languages.

Keywords
Web services, Policy Framework, Policy Management, Policy Enforcement.

1. Background
The term 'Policy' in distributed systems typically refers to an externally consumable statement of system constraints, capabilities or requirements that effect the interaction between a consumer and a service. In some cases, the policy may simply impact the decision to make use of a service; in other cases, the policy may place constraints on the interaction itself. An example of the former is a privacy policy, which, if deemed unacceptable, will cause the consumer to forgo use of a service altogether. An example of the latter is a policy that dictates that the service be used in the context of a transaction. In this case, interactions with the service must somehow be scoped as part of a larger unit of work.

Systems that are designed primarily with human users as principal actors in the consumer role tend to advertise policies that revolve around the decision to use a service. The archetypical example of such a system is the Web. Policies for the Web tend to fall into several classes

Policies designed to encourage use

Users may consider it desirable for a Web site to maintain strict rules about how information about site users is managed. For example, users are more likely to use a Web site if they have confidence the site owner will not distribute personal information and will guarantee an adequate level of protection for credit card data.

Though formal syntax is not always used to express policies of this nature, the Platform for Privacy Preferences (P3P) specification [1] describes a policy language for expressing the privacy rules adhered to by an organization in machine readable and human interpretable form. These policies generally assume a level of trust; in the Web environment, this is typically gained through a combination of certification by an independent authority and perhaps more commonly by reputation.

Policies designed to constrain access

Rules surrounding the access rights for a Web server are an example of this kind of policy. Typically, authentication and authorization procedures are integrated with the Web site's human user interface; in this case, the communication mechanism is relatively ad hoc and presented via HTML or similar markup languages.

Policies about availability

These are policies that declare under what terms a service is available. This information is typically communicated in quality of service agreements, as maintenance notices, or general information about a Web site. Examples of this kind of policy are notices of administrative practices requiring downtime for maintenance or payment requirements for use. These policy statements are important mechanisms for managing user expectations; in some cases, users may decide not to use a site based on conflicting availability requirements. Availability may apply not only to network presence of the service, but also to secondary business functions. For example, a Web site may be available on a 24X7 basis but may not have order processing available on weekends. Policies dealing with availability are also typically expressed through markup and interpreted by users.

These policy categories are not mutually exclusive. For example, a Web site may have policies that are intended to encourage the use of a site by a restricted class of users. The salient feature of Web policies is that they tend to be heavily oriented toward direct consumption by human users, assuming that users will find the policies and interpret them satisfactorily. In many cases, the policies are expressed in written statements on Web sites. Policies for Web sites tend to apply to the broad aspects of the site, rather than individual resources. For example, a certain portion of a Web site may require payment for use. More specialized services that provide access to copyrighted digital assets often place constraints on classes of resources (for example, you must pay .99 USD to download a song from Apple's popular iTunes Web site).

Distributed systems that focus on machine-to-machine interoperability have traditionally provided policies reflecting low-level constructs familiar to programmers that build such systems. Taking CORBA [2] as a representative example, policies are typically based on local configuration that is in turn tied to specific object references exported into the user environment. Policies for system level functions like security or transactions are exposed as properties of the distributed object reference (CORBA IOR). This allows programs to analyze remote services dynamically to assure that appropriate quality of service semantics are maintained when the service is invoked.

These policies are in general different from the typical Web policies in that:

1) Middleware policies are intended to be interpreted and used by software systems rather than human users.
2) For the most part, middleware policies deal with defining the semantics of interactions with a service. These policies are very different from the kinds of policies that are defined for Web resources.
3) These policies are very tightly bound to specific service implementations. In the CORBA example, policies are expressed to clients of the service within each individual object reference. Typical CORBA programs are based on the object oriented design paradigm, which may encourage the use of very fine-grained policies.

Web services policies combine elements found in both traditional middleware for machine-to-machine interoperability and policies associated with Web resources.

2. Web Services Policy

A general breakdown of the Web services policy space today includes:

Policies that focus on enabling and exposing traditional middleware system services like message delivery guarantees, transaction semantics, and security requirements. The WS-Policy Framework [3] specification proposed by Microsoft and IBM is oriented heavily toward expressing this kind of policy. Its emphasis on selection and logical operators – which we believe is of limited utility in practice even for the case of system services – make it a poor choice for other kinds of policies. As a general rule, these policies will affect the message payload by the addition of SOAP [4] headers specific to the policy selection that has been made for a message exchange. For example, the use of a WS-Reliability [5] functionality in a message exchange will include SOAP headers that look something like the following:

```
<wsrm:Request
xmlns:wsrm="http://www.oasisopen.org/committees/ws
rm/schema/1.1/SOAP1.1"
xmlns:SOAP="http://schemas/xmlsoap.org/soap/envelo
pe/"
SOAP:mustUnderstand="1">
<wsrm:MessageIdgroupId="20041221-160154-
022.9@nobody.oracle.com"/>
<wsrm:ExpiryTime>2005-04-
16T09:48:34</wsrm:ExpiryTime>
<wsrm:ReplyPattern>
<wsrm:Value>Poll</wsrm:Value>
</wsrm:ReplyPattern>
<wsrm:AckRequested/>
<wsrm:DuplicateElimination/>
</wsrm:Request>
```

Information policies: in many cases these will be formalizations of the kind of Web polices outlined above. Web services will require structured mechanisms to express informational policies, but complex policies will continue to be provided in forms targeted for direct human consumption in the near term. We believe that higher-level protocols will need to be developed to allow clients to express their expectations about specific informational policies. Informational policies typically impact the decision to use a service rather than the specific content of a message exchange. For example, a P3P document may express policies about the maintenance of personal information that are unacceptable to some users.

Service level agreements guaranteeing some combination of commitments around the quality of the service itself and the underlying business processes it represents. These policies are often tailored to specific users or classes of users and may depend on complex business rules. These policies are often applied by leveraging specific information associated with the established identity of the message sender.

Aside from the classes of policies we identify above, we assume the following requirements for Web services policies:

1) More than one policy may be associated with a service. We believe that multiple policies, often representing very different kinds of policy domains, will be in effect for a single service. For example, a single service may include policies for security, privacy, and business agreements.
2) A single policy may be associated with more than one service. Large organizations expect to set global policies and assure normal constraints and rules for sets of services. End users seeking to create a SOA are looking for mechanisms to support policy normalization.
3) Policies associated with a service may change over the lifetime of a service. For example, new polices may be introduced after a service has been deployed or existing policies may evolve over time
4) Policies need to vary independent of WSDL: new policies should be managed and provisioned independently of the basic business function and message exchanges offered by a service implementation.

At the current time, the Web services policy space is murky and evolving. There are proprietary proposals that emphasize different aspects of policy requirements, but tend to support one class of policy types better than others. In addition, there is the general problem of business rules and semantics. So called Semantic Web services have garnered great interest in academic circles but have not made in-roads in practice in the software industry.

The first step for providing a policy management solution is to achieve a standardized policy framework capable of meeting the requirements we have outlined. Regrettably, the industry has not yet been able to reach this critical milestone; in fact, no widely accepted standard effort exists in this space at the time of this writing. As a result, policies are often created in ad hoc ways and communicated through mechanisms that are out of band with respect to the Web services architecture and model. For example, we know of organizations maintaining Word documents that are passed via email describing how their Web services should be used. We believe the following design goals should be accommodated in a viable policy framework standard.

First, a policy framework should be able to support for different domains and styles of policy expression. Services will be bounded by a range of policy types, each critical in its own regard. A framework for supporting policies for security, reliability and transactions is necessary but insufficient. On the other hand, these

kinds of policies should be able to be expressed in a simple and easy to process set of assertions. We believe that a useful policy framework should provide containers for domain expressions that may utilize their own syntax and express their semantic requirements in a domain specific manner. The outline of a framework that provides domain containers is described in [6]. Much of the either/or discussions about policies that utilize Semantic Web capabilities versus assertion-based model may miss the point: domains should be free to utilize the technologies that appear best suited for the specific problem space

Second, informational policies are processed by service consumers to determine if a service may be used. Since policies may evolve independent of service interfaces, consumers should be able to express their expectations about informational policies that are believed to apply to a service. A SOAP header with a mustUnderstand=”1” attribute could be used to convey expectations about specific informational policies; services that are not observing the policy expectation should return a fault rather than process the SOAP message carrying unsatisfied expectations.

Third, a policy document will be associated with a Web service. The standard should ensure that policies are not required to be included within WSDL documents or constructs so that the two may evolve freely. To support this model, we advocate extensions to WSDL indicating that a policy is enforced and how it may be obtained.

3. Policy Management

The classes of polices and general requirements for policies in the Web services environment, taken together, directly help to define the scope of a Web services policy management solution. Specifically, a Web services policy management solution needs to manage:

1) Policy Lifecycle

This includes the definition, maintenance and application of policies. The management of policies throughout their lifecycle combines problems of metadata management and organization as well as content management versioning and control facilities. Policies may be ad hoc or informal and should also be supported within the system: another motivator for dividing policy expressions into independent domains Many Web services management products support a policy repository capability that supplies some or all of these features and some protocol to provision policies to enforcement points. At the present point in time, these functions are achieved by non-standard and proprietary mechanisms.

2) Policy Discovery/Access

End users need to have access to policies to make decision about whether to use and how to use a service. Regardless of how policy lifecycles are controlled, a policy management solution must allow for metadata retrieval and policy organization. Most solutions will provide an association of policies and services, generally organized with some logical structure, perhaps based on taxonomies. The UDDI specification [7] provides interoperable rules for service registries, which can also expose policies and associated resources. In some cases, the Web services platform on which a service is hosted will directly supply the policy in response to a specific query using the HTTP protocol or a specialized Web services protocol for metadata retrieval. The WS-MetadataExchange specification [8] is an example of the latter.

3) Enforcement of policies for individual and groups of services.

One mechanism that is emerging in practice to handle policy enforcement is gateway services that act as active intermediaries in the SOAP processing model. The gateways process SOAP messages and enforce policy constraints or resolve system-level instructions before the message is provided to the service implementation for processing. For example, a gateway service may manage authentication and authorization based on policies defining the access control rules for a service or group of services (policy normalization). We believe that Web services intermediaries will prove to be fundamental to Service Oriented Architecture (SOA) deployments; we discuss this area in more detail below.

A policy management solution is foundational to a SOA: it provides a global model for an organization to understand and control the services within an organization. While application servers provide hosting platforms for individual services, a policy management solution provides visibility and control over a SOA topology and its characteristics. From this perspective, policies for organizations may be most effectively managed in centralized repositories that allow for businesses to set global policies and store information about how a service may be used. Individual service deployments can extend and specialize policies based on their specific requirements; this implies that well-defined rules must be in place for how policy domain expressions may be combined. Again, we believe this is largely a domain specific problem. Managing and storing metadata about services is largely a data management problem and amenable to storage in metadata containers built on standard relational database solutions.

Somewhat more problematic is the enforcement of managed policies, since services typically rest on a heterogeneous set of application server technologies. We believe that the following methods of policy enforcement are viable solutions for the Web services environment: local agents and gateways.

Agents that reside at service endpoints.

Agents allow processing logic to be inserted directly at service endpoints. This can occur via interception of the carrier protocol stream or within application server specific extensibility points specific to the Web services environment, such as JAX-RPC [8] Handlers. In either case, agents need to receive current policy definitions from the management repository.

Gateway-type active intermediaries.

Active intermediaries in the SOAP processing model can often be used to spread the processing logic of ultimate message recipients across multiple servers. A gateway can be configured to transparently enforce policies that are expressed as properties of the Web service. While the archetypical use case for Web services gateways is enforcement of security policies, almost any policy can be enforced or observed via a gateway architecture by organizing a pipeline of policy enforcement steps required for the service. Since these intermediaries may combine global and service specific policies, composition rules should be well-specified and isolated to overlapping domains.

Both enforcement mechanisms can be used to provide data about

policy enforcement to systems management consoles. This combination of a well-factored policy framework, policy provisioning, access, and enforcement mechanisms, and monitoring capabilities provide a compelling solution for the Web services environment.

One area that requires special care is the provisioning of policies between centralized repositories and enforcement points: it is important that policies are applied consistently, particularly in replica-based cluster environments. This can be a significant challenge in agent-based systems and is an area that is rife for interoperability research proposals and ultimately standardization.

4. Conclusion

A complete Policy framework needs to accommodate the requirements for different classes of policies and the solution architecture that is emerging for the management of policies. We do not believe that current proposals meet the full range of requirements that exist for a complete Web services policy solution. In particular, current proposals are not tailored to the emerging requirements, organization and deployment topologies of Web services networks and policy management solutions that are required for a coherent SOA deployment.

5. Acknowledgements

Special thanks to Ashok Malhotra and Jon Maron for their insightful comments. Thanks also to the Oblix CoreSV product team for sharpening our understanding of Web services management in commercial practice.

6. References

[1] Cranor, Lorrie et al. The Platform for Privacy Preferences 1.0 Specification. (April 2002) http://www.w3.org/TR/P3P/

[2] Common Object Request Broker Architecture: Core Specification. (March 2004) http://www.omg.org/docs/formal/04-03-01.pdf

[3] Bajaj, Siddharth et al. Web Services Policy Framework. (September 2004) ftp://www6.software.ibm.com/software/developer/library/ws-policy.pdf

[4] Gugdin, Martin et al. SOAP Version 1.2 Part 1: Messaging Framework. (June 2003) http://www.w3.org/TR/soap12-part1

[5] Iwasa, Kazunori. WS-Reliability 1.1. (August 2004) http://docs.oasis-open.org/wsrm/2004/06/WS-Reliability-CD1.086.pdf

[6] Ashok Malhotra and Umit Yalcinalp. Position Paper for W3C Constraints and Capabilities Workshop. (August 2004) http://www.w3.org/2004/08/ws-cc/amuy-20040903

[7] Ballinger, Keith et al. Web Services Metadata Exchange (September 2004) ftp://www6.software.ibm.com/software/developer/library/WS-MetadataExchange.pdf

[8] Chinnici, Roberto and Hadley, Marc. Java API for XML based RPC (JAX-RPC) 2.0. (June 2004) http://jcp.org/aboutJava/communityprocess/edr/jsr224

Representing Security Policies in Web Information Systems

Félix J. García Clemente
Departamento de Ingeniería de la Información y las Comunicaciones
Campus de Espinardo, s/n
30.071 Murcia, Spain
+34 968 367645
fgarcia@dif.um.es

Gregorio Martínez Pérez
Departamento de Ingeniería de la Información y las Comunicaciones
Campus de Espinardo, s/n
30.071 Murcia, Spain
+34 968 367646
gregorio@dif.um.es

Juan A. Botía Blaya
Departamento de Ingeniería de la Información y las Comunicaciones
Campus de Espinardo, s/n
30.071 Murcia, Spain
+34 968 367317
juanbot@um.es

Antonio F. Gómez Skarmeta
Departamento de Ingeniería de la Información y las Comunicaciones
Campus de Espinardo, s/n
30.071 Murcia, Spain
+34 968 364607
skarmeta@dif.um.es

ABSTRACT
Policies, which usually govern the behaviour of networking services (e.g., security, QoS, mobility, etc.), are becoming an increasingly popular approach for the dynamic regulation of web information systems. The adoption of a policy-based approach for controlling a system requires an appropriate policy representation regarding both syntax and semantics, and the design and development of a policy management framework. In the context of the Web, the use of languages enriched with semantics (i.e. semantic languages) has been limited primarily to represent Web content and services. However the capabilities of these languages, coupled with the availability of tools to manipulate them, make them well suited for many other kinds of application, as policy representation and management. This paper provides the current trends of policy-based management enriched by semantics applied to the protection of web information systems. It also presents an approach for using DMTF Common Information Model (CIM) ontology with semantic languages.

Categories and Subject Descriptors
K.6.5 [**Management of Computing and Information System**]: Security and Protection.

General Terms
Management, Security, Languages

Keywords
Semantic Languages, Security Policy, CIM Ontology

1. INTRODUCTION
One of the main goals of policy-based management is to enable network, service and application control and management at a high abstraction layer. Using a policy language, the administrator specifies rules that describe domain-wide policies which are independent of the implementation of the particular network node, service and/or application. It is, then, the policy management architecture that provides support to transform and distribute the policies to each node and thus enforce a consistent configuration in all the elements involved. This is a prerequisite for achieving a mean to dynamically constrain and regulate the behaviour of a system without the human cooperation.

Copyright is held by the author/owner(s).
WWW 2005, May 10--14, 2005, Chiba, Japan.

In the web information systems security field, a policy (i.e., security policy) can be defined as a set of rules and practices describing how an organization manages, protects and distributes sensitive information at several levels. Security policies can be defined to perform a wide variety of actions, from IPsec/IKE management (example of network security policy) to access control over a web server (example of application-level policy).

Researchers have proposed multiple approaches for policy specification. They range from formal policy languages that a computer can directly process, to rule-based policy notation using an *if-then-else* format, or to the representation of policies based on Deontic logic for obligation and permissibility rules.

To cover this wide range of security policies languages, this paper aims to examine the current state of policy engines and policy languages, focusing on the approaches enriched with semantics (i.e. semantic languages) using RDF [11] and OWL [2] as standards for policy specification. We intend to show the strengths and limitations of such languages by comparing three approaches: KAoS, Rei and SWRL.

The major benefit of specifying security policy rules in this way is that an organization can utilize a common ontology that can be shared amongst services and service clients. In this sense, DMTF presents the Common Information Model (CIM) standard [4] to provide a common definition of management-related information. This paper also presents an approach for using CIM ontology with semantic languages. It permits an administrator to formally describe the security policies of an administrative domain using the DMTF methodology.

This document is structured as follows. Section 2 presents the requirements of policy frameworks, focusing on policy languages and policy architectures. Then, section 3 presents a comparative analysis between “traditional” non-semantic and semantic policy frameworks to emphasize the advantages of semantic approaches. Section 4 describes and compares the three semantic approaches aforementioned. Then, section 5 presents the extension of the semantic policy language SWRL with the CIM ontology and shows an example for an authorization policy. Finally, we conclude the paper with our remarks and some future directions derived from this work.

2. REQUIREMENTS FOR A POLICY FRAMEWORK

The policy administrator needs to use a policy language that assures that the representation of policies guarantee the following requirements:

Well-defined. A policy language can be considered as well-defined if the syntax and structure is clear and no-ambiguous, and the meaning of a policy written in this language is independent of its particular implementation.

Flexibility and extensibility. A policy language has to be flexible enough to allow new policy information to be expressed, and extensible enough to allow new types of policy to be added in future versions of this language.

Interoperability with other languages. There are usually several languages that can be used in different domains to express similar policies, and interoperability is a must to allow different services or applications from these different domains to communicate with each other according to the behaviour stated in these policies.

Once the policy has been defined for a given administrative domain, a management architecture is required to transfer, store and enforce this policy in that domain. The main requirements for such policy management architecture are:

Well-defined interface. Policy architectures need to have a well-defined interface independent of the particular implementation in use. In it, the interfaces between the components need to be clear and no-ambiguous.

Flexibility and definition of abstractions to manage a wide variety of device types. The system architecture should be flexible enough to allow addition of new types of devices with minimal updates and recoding of existing management components.

Interoperability with other architectures (inter-domain). The system should be able to interoperate with other architectures that may exist in other administrative domains.

Conflict Detection. It has to be able to check that a given policy does not conflict with any other existing policy.

Scalability. It should maintain quality performance under an increased system load.

The policy framework has to support all these requirements to guarantee the correct system operation.

3. ADVANTAGES OF SEMANTIC SECURITY POLICY FRAMEWORKS

There are some non-semantic security policy frameworks such as Ponder [3] and XACML [7] that we describe briefly as follows:

Ponder, is a declarative, object-oriented language developed for specifying management and security policies. Ponder permits to express authorizations, obligations, information filtering, refrain policies, and delegation policies. Ponder can describe any rule to constrain the behaviour of components, in a simple and declarative way.

The eXtensible Access Control Markup Language (XACML) describes both an access control policy language and a request/response language. The policy language provides a common means to express subject-target-action-condition access control policies and the request/response language expresses queries about whether a particular access should be allowed and describes answers to those queries.

However, they do not take care of the description of the content of the policy (e.g., description of the specified components, the system, etc). The adoption of a semantic web language can overcome this limitation since it uses an ontology to describe the content of the policies.

In general, table 1 shows a comparative between semantic and non-semantic policy languages based on [9] and complemented with our own analysis [6].

Table 1. Comparative analysis between semantic and non-semantic policy languages

	Semantic Languages	Non-Semantic Languages
Abstraction	Multiple levels	Medium and low level
Extensibility	Easy and at runtime	Complex and at compile-time
Representability	Complex environments	Specific environments
Readability	Specialized tools	Direct
Interoperation	By common ontology	By interfaces
Enforcement	Complex	Easy

Semantic approaches using RDF/OWL (see Section 4) as standards for policy representation enable runtime extensibility and adaptability of the system, as well as the ability to analyse policies relating to entities described at different levels of abstraction. The representation facilitates careful reasoning about policy disclosure, conflict detection, and harmonization about domain structure and concepts. However, it is required complex policy automation mechanisms for enforcement.

4. SEMANTIC SECURITY POLICY LANGUAGES

As stated before, security policies can be specified at different levels of abstraction. The process starts with the definition of a business security policy. This can be the case of the next authorization security policy, which is defined in natural language: *"Permit the access to the e-payment service, if the user is in the group of customers registered for this service"*.

Next, the security policy is usually expressed by a policy administrator as a set of IF-THEN policy rules, for example: *IF ((<Requester> is member of Payment Customers) AND (<Server> is member of Payment Servers)) THEN (<Requester> granted access to <Server>)*

The policy languages we will be analyzing in this section are able to specify several types of security policies and will be used to provide policy examples related to this case study.

Although many semantic policy specifications exist, we have selected three of them as they are considered nowadays as promising options: KAoS, Rei and SWRL.

4.1 KAoS

KAoS [10] is a collection of services and tools that allow for the specification, management, conflict resolution, and enforcement

of deontic-logic-based policies within domains describing organizations of human, agent, and other computational actors.

KAoS uses ontology concepts encoded in OWL to build policies. The KAoS Policy Service distinguishes between authorization policies and obligation policies. The applicability of the policy is defined by a set of conditions or situations whose definition can contain components specifying required history, state and currently undertaken action. In the case of the obligation policy the obligated action can be annotated with different constraints restricting possibilities of its fulfilment.

The current version of the KAoS Policy Ontologies (KPO) defines basic ontologies for actions, conditions, actors, various entities related to actions, and policies. It is expected that for a given application, the ontologies will be further extended with additional classes, individuals, and rules.

Figure 1 shows an example of the type of policy that administrators can specify using KAoS. It is related with the case study described earlier.

```
<owl:Class rdf:ID="PaymentAuthAction">
<owl:intersectionOf rdf:parseType="owl:collection">
  <owl:Class rdf:about="&action;AccessAction"/>
  <owl:Restriction>
  <owl:onProperty rdf:resource="&action;#performedBy"/>
  <owl:toClass
        rdf:resource="&domains;MembersOfPayCustomer"/>
  </owl:Restriction>
  <owl:Restriction>
  <owl:onProperty rdf:resource="&action;#performedOn"/>
  <owl:toClass
       rdf:resource="&domains;MembersOfPayServer"/>
  </owl:Restriction>
</owl:intersectionOf>
</owl:Class>
<policy:PosAuthorizationPolicy rdf:ID="PaymentAuthPolicy1">
  <policy:controls rdf:ID="PaymentAuthAction"/>
  <policy:hasSiteOfEnforcement rdf:resource="#TargetSite"/>
  <policy:hasPriority>1</policy:hasPriority>
</policy:PosAuthorizationPolicy>
```

Figure 1. Example of policy representation in KAoS

KAoS defines a Policy Framework that includes the following functionality:

Creating/editing of policies using KAoS Policy Administration Tool (KPAT). KPAT implements a graphical user interface to policy and domain management functionality.

Storing, de-conflicting and querying policies using KaoS Directory Service.

Distribution of policies to Guard, which acts as a policy decision point.

Policy enforcement/disclosure mechanism, i.e. finding out which policies apply to a given situation.

Every agent in the system is associated with a Guard. When an action is requested, the Guard is automatically queried to check whether the action is authorized based on the current policies and, if not, the action is prevented by various enforcement mechanisms. Policy enforcement requires the ability to monitor and intercept actions, and allow or disallow them based on a given set of policies. While the rest of the KAoS architecture is generic across different platforms, enforcement mechanisms are necessarily specific to the way the platform works.

4.2 Rei

Rei [5] is a policy framework that integrates support for policy specification, analysis and reasoning. Its deontic-logic-based policy language allows users to express and represent the concepts of rights, prohibitions, obligations, and dispensations. In addition, Rei permits users to specify policies that are defined as rules associating an entity of a managed domain with its set of rights, prohibitions, obligations, and dispensations.

Rei provides a policy specification language in OWL-Lite that allows users to develop declarative policies over domain specific ontologies in RDF, DAML+OIL and OWL.

A policy primarily includes a list of granting and a context used to define the policy domain. A granting associates a set of constraints with a deontic object to form a policy rule. This allows reuse of deontic objects in different policies with different constraints and actors. A deontic object represents permissions, prohibitions, obligations and dispensations over entities in the policy domain. It includes constructs for describing what action (or set of actions) the deontic is described over, who the potential actor (or set of actors) of the action is and under what conditions is the deontic object applicable.

An action is one of the most important in the Rei specifications as policies are described over possible actions in the domain. The domain actions describe application or domain specific actions, whereas the speech acts are primarily used for dynamic and remote policy management.

There are six subclasses of SpeechAct: Delegate, Revoke, Request, Cancel, Command, and Promise. A valid delegation leads to a new permission. Similarly, a revocation speech act nullifies an existing permission (whether policy based or delegation based) by causing a prohibition. An entity can request another entity for a permission, which if accepted causes a delegation, or to perform an action on its behalf, which if accepted causes an obligation. An entity can also cancel any previously made request, which leads to a revocation and/or a dispensation. A command causes an obligation on the recipient and the promise causes an obligation on the sender.

To enable dynamic conflict resolution, Rei also includes meta-policy specifications, namely setting the modality preference (negative over positive or vice versa) or stating the priority between rules within a policy or between policies themselves.

Figure 2 shows an example to illustrate the policy representation in Rei. It is related with the case study described earlier.

```
<constraint:SimpleConstraint rdf:ID="IsPayCustomer"
   constraint:subject="#RequesterVar"
   constraint:predicate="&example;memberOf"
   constraint:object="&example;payCustomer"/>
<constraint:SimpleConstraint rdf:ID="IsPayServer"
   constraint:subject="#PayServerVar"
   constraint:predicate="&example;memberOf"
   constraint:object="&example;payServer"/>
```

```
<constraint:And rdf:ID=”ArePayCustomerAndPayServer”
   constraint:first=”#IsPayCustomer”
   constraint:second=”#IsPayServer”/>
<deontic:Permission rdf:ID=”PayServerPermission”>
   <deontic:actor rdf:resource=”#RequesterVar”/>
   <deontic:action rdf:resource=”&example;access”/>
   <deontic:constraint
          rdf:resource=”#ArePayCustomerAndPayServer”/>
</deontic:Permission>
<policy:Policy rdf:ID=”PaymentAuthPolicy1”>
   <policy:grants rdf:resource=”#PayServerPermission”/>
 </policy:Policy>
```

Figure 2. Example of policy representation in Rei

The Rei framework provides a policy engine that reasons about the policy specifications. The engine accepts policy specification in both the Rei language and in RDF-S [1], consistent with the Rei ontology. Specifically, the engine automatically translates the RDF specification into triplets of the form (subject, predicate, object). The engine also accepts additional domain-dependent information in any semantic language that can then be converted into this recognizable form of triplet. The engine allows queries according to the Prolog language about any policies, meta-policies, and domain dependent knowledge that have been loaded in its knowledge base.
The Rei framework does not provide an enforcement model. In fact, the policy engine has not been designed to enforce the policies but only to reason about them and reply to queries.

4.3 SWRL

Semantic Web Rule Language (SWRL) [8] is based on a combination of the OWL DL and OWL Lite sublanguages of the OWL with the Unary/Binary Datalog RuleML sublanguages. SWRL extends the OWL abstract syntax to include a high-level abstract syntax for Horn-like rules. A model-theoretic semantics is given to provide the formal meaning for OWL ontologies including rules written in this abstract syntax.

We distinguish between the following facts/rules for policy representation:

Structural/organizational facts and rules. These rules are used to encode domain specific ontologies.

Service definition facts and rules, provided with links to the structural rules and facts.

Task-specific rules and facts, provided by the service clients.

SWRL is defined by an XML syntax based on RuleML and the OWL XML Presentation Syntax. The rule syntax is illustrated with the following example related with the case study described earlier.

```
<ruleml:imp>
  <ruleml:_head>
   <swrlx:individualPropertyAtom
                       swrlx:property="GrantedAccess">
          <ruleml:var>requester</ruleml:var>
          <ruleml:var>server</ruleml:var>
   </swrlx:individualPropertyAtom>
  </ruleml:_head>
  <ruleml:_body>
   <swrlx:classAtom>
          <owlx:Class owlx:name="User" />
          <ruleml:var>requester</ruleml:var>
   </swrlx:classAtom>
   <swrlx:classAtom>
          <owlx:Class owlx:name="Server" />
          <ruleml:var>server</ruleml:var>
   </swrlx:classAtom>
   <swrlx:individualPropertyAtom swrlx:property="Member">
          <ruleml:var>requester</ruleml:var>
          <owlx:Individual owlx:name="#PayCustomer" />
   </swrlx:individualPropertyAtom>
   <swrlx:individualPropertyAtom swrlx:property="Member">
          <ruleml:var>server</ruleml:var>
          <owlx:Individual owlx:name="#PayServer" />
   </swrlx:individualPropertyAtom>
  </ruleml:_body>
</ruleml:imp>
```

Figure 3. Example of policy representation in SWRL

A useful restriction in the form of the rules is to limit antecedent and consequent classAtoms to be named classes, where the classes are defined purely in OWL. Adhering to this format makes it easier to translate rules to or from existing or future rule systems, including Prolog.

4.4 Comparative Analysis

Table 2 shows a comparison of the aforementioned security policy languages. Many aspects can be identified as part of this comparison, although the most relevant are:

Approach. Two types of approaches have been identified: rule-based and deontic logic-based.

Specification language. It can be XML, RDF-S or OWL.

Tools for policy specification.

Reasoning engine for policy analysis and verification.

Enforcement support to the policy deployment.

Table 2. Comparative analysis between KAoS, SWRL and Rei

	KAoS	Rei	SWRL
Approach	Deontic Logic	Deontic Logic + Rules	Rules
Specification language	DAML/OWL	Prolog-like syntax + RDF-S	Prolog-like syntax + OWL
Tools for specification	KPAT	No	No
Reasoning	KAoS engine	Prolog engine	Prolog engine
Enforcement	Supported	External Functionality	External Functionality

OWL has a limited way of defining restrictions using the tag owl:Restriction. This limitation also appears in KAoS, but SWRL overcomes it by the extending the set of OWL axioms including horn-like rules. On the other hand, SWRL is not limited to deontic policies as it happens in Rei and KAoS.

5. USING CIM ONTOLOGY WITH SEMANTIC LANGUAGES

The Common Information Model (CIM) is an approach from the DMTF that applies the basic structuring and conceptualization techniques of the object-oriented paradigm to provide a common

definition of management-related information for systems, networks, users, and services.

The CIM model is independent of any implementation or specification. However, for an information model to be useful, it must be mapped into some implementation. As Figure 4 showed, CIM can be mapped to several structured specifications.

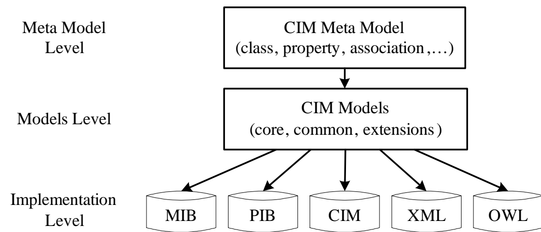


Figure 4. CIM modelling levels

An advantage of CIM is that the model can be mapped to structured specifications such as OWL, which can then be used to define management resources for Web Information System (WIS). Also note that the mapping of CIM to a valid representation for WIS is beneficial, since it permits to model WIS components using the DMTF methodology and hence obtain a standard and interoperable representation of it.

According to our approach, regarding the mapping of CIM into OWL, the main principles identified as part of this process are:

- Every CIM class generates a new OWL class using the tag <owl:Class>.
- Every CIM generation (inheritance) is expressed using the tag <rdfs:subClassOf>.
- Every CIM class attribute is specified using the tag <owl:DatatypeProperty> for literal values or <owl:ObjectProperty> as references to class instances.
- Every CIM association is expressed as an OWL class with two <owl:ObjectProperty> where their identifiers (i.e., <rdf:ID>) are the names of the properties of the CIM association; this is the most suitable general-purpose mechanism currently available.

An example of these transformations for the CIM classes related to the user authorization is now presented and explained. CIM defines the classes depicted in Figure 5 to represent the management concepts that are related to an authorization privilege. Privilege is the base class for all types of activities, which are granted or denied to a subject by a target. Authorized-Privilege is the specific subclass for the authorization activity.

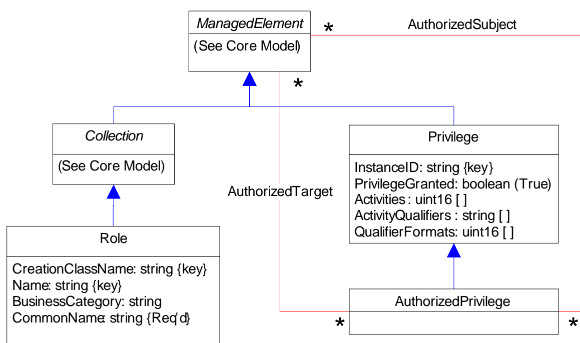


Figure 5. UML diagram of User-Authentication classes

Whether an individual Privilege is granted or denied is defined using the PrivilegeGranted boolean. The association of subjects to AuhorizedPrivileges is accomplished explicitly via the association AuthorizedSubject. The entities that are protected (targets) can be similarly defined via the association AuthorizedTarget. Note that AuthorizedPrivilege and its AuthorizedSubject/Target associations provide a static mechanism to represent authorization policies.

An example of the mapping of these CIM classes to OWL is illustrated in the Figure 6. This example shows a fragment of the mapping of CIM class Privilege and CIM association AuthorizedSubject.

```
<owl:Class rdf:ID="CIM_Privilege">
        <rdfs:subClassOf
                rdf:resource="CIM_ManagedElement"/>
</owl:Class>
<owl:Class rdf:ID="CIM_AuthorizedSubject">
        <rdfs:subClassOf rdf:resource="LogicalEntity"/>
</owl:Class>
<rdf:DatatypeProperty rdf:ID="InstanceID">
        <rdfs:domain rdf:resource="CIM_Privilege"/>
        <rdfs:range rdf:resource="String"/>
</rdf:DatatypeProperty>
<rdf:DatatypeProperty rdf:ID="PrivilegeGranted">
        <rdfs:domain rdf:resource="CIM_Privilege"/>
        <rdfs:range rdf:resource="Boolean"/>
</rdf:DatatypeProperty>
<rdf:DatatypeProperty rdf:ID="Activities">
        <rdfs:domain rdf:resource="CIM_Privilege"/>
        <rdfs:range rdf:resource="Uint16"/>
</rdf:DatatypeProperty>
<rdf:DatatypeProperty rdf:ID="ActivityQualifers">
        <rdfs:domain rdf:resource="CIM_Privilege"/>
        <rdfs:range rdf:resource="String"/>
</rdf:DatatypeProperty>
<rdf:DatatypeProperty rdf:ID="QualiferFormats">
        <rdfs:domain rdf:resource="CIM_Privilege"/>
        <rdfs:range rdf:resource="Uint16"/>
</rdf:DatatypeProperty>
<rdf:ObjectProperty rdf:ID="Privilege">
        <rdfs:domain rdf:resource="CIM_AuthorizedSubject"/>
        <rdfs:range rdf:resource="CIM_ManagedElement"/>
</rdf:ObjectProperty>
<rdf:ObjectProperty rdf:ID="PrivilegedElement">
        <rdfs:domain rdf:resource="CIM_AuthorizedSubject"/>
        <rdfs:range rdf:resource="CIM_ManagedElement"/>
</rdf:ObjectProperty>
```

Figure 6. A fragment of the mapping of Privilege and AuthorizedSubject into OWL

Note that the ontological representation of CIM (i.e., OWL representation) permits to represent a CIM ontology that can be used in semantic policy languages (e.g., SWRL).

SWRL uses ontology concepts encoded in OWL to build rules. It can be extended with the OWL CIM ontology. For example, rule syntax is illustrated in the Figure 7 related with the case study described earlier.

```
<ruleml:imp>
  <ruleml:_body>
    <swrlx:classAtom>
        <owlx:Class owlx:name="CIM_Role"/>
        <ruleml:var>server</ruleml:var>
```

```
  </swrlx:classAtom>
  <swrlx:classAtom>
        <owlx:Class owlx:name="CIM_Role" />
        <ruleml:var>requester</ruleml:var>
  </swrlx:classAtom>
  <swrlx:classAtom>
        <owlx:Class owlx:name="CIM_AuthorizedPrivilege" />
        <ruleml:var>privilege</ruleml:var>
  </swrlx:classAtom>
  <swrlx:individualPropertyAtom swrlx:property="Name">
        <ruleml:var>server</ruleml:var>
        <owlx:Individual owlx:name="#PayServer" />
  </swrlx:individualPropertyAtom>
  <swrlx:individualPropertyAtom swrlx:property="Name">
        <ruleml:var>requester</ruleml:var>
        <owlx:Individual owlx:name="#PayCustomer" />
  </swrlx:individualPropertyAtom>
  <swrlx:individualPropertyAtom swrlx:property="Name">
        <ruleml:var>privilege</ruleml:var>
        <owlx:Individual owlx:name="#GrantedAccess" />
  </swrlx:individualPropertyAtom>
 </ruleml:_body>
 <ruleml:_head>
  <swrlx:classAtom>
        <owlx:Class owlx:name="CIM_AuthorizedTarget" />
        <ruleml:var>authtarget</ruleml:var>
  </swrlx:classAtom>
  <swrlx:classAtom>
        <owlx:Class owlx:name="CIM_AuthorizedSubject" />
        <ruleml:var>authsubject</ruleml:var>
  </swrlx:classAtom>
  <swrlx:individualPropertyAtom swrlx:property="Privilege">
        <ruleml:var>authtarget</ruleml:var>
        <ruleml:var>privilege</ruleml:var>
  </swrlx:individualPropertyAtom>
  <swrlx:individualPropertyAtom
        swrlx:property="TargetElement">
        <ruleml:var>authtarget</ruleml:var>
        <ruleml:var>server</ruleml:var>
  </swrlx:individualPropertyAtom>
  <swrlx:individualPropertyAtom swrlx:property="Privilege">
        <ruleml:var>authsubject</ruleml:var>
        <ruleml:var>privilege</ruleml:var>
  </swrlx:individualPropertyAtom>
  <swrlx:individualPropertyAtom
        swrlx:property="PrivilegedElement">
        <ruleml:var>authsubject</ruleml:var>
        <ruleml:var>requester</ruleml:var>
  </swrlx:individualPropertyAtom>
 </ruleml:_head>
</ruleml:imp>
```

Figure 7. Example of policy representation in SWRL using the CIM ontology

6. CONCLUSIONS

This paper has provided some discussions of the most relevant security-aware semantic specification languages and information models. Our perspective on the main issues and problems of each of them has also been presented, based on different criteria such as their approach or the specification technique they use. It has also presented an approach for using CIM ontology with the semantic languages.

Our future work is being planned to investigate how the CIM information model can be used as ontology for other semantic security policy languages. In this sense the current research work undertaken in the POSITIF EU IST project [12] is gathering requirements of security management in web and information systems and defining, based on the work presented in this paper, a semantic security policy language able to formally define the desired security policy.

7. ACKNOWLEDGMENTS

This work has been partially funded by the EU POSITIF (Policy-based Security Tools and Framework) IST project (IST-2002-002314).

8. REFERENCES

[1] Brickley, D., and Guha, R. V. (2004, January). Rdf vocabulary description language 1.0: Rdf schema. Technical report, W3C Working Draft.

[2] Connolly, D., Dean, M., Harmelen, F., Hendler, J., Horrocks, I., McGuinness, D. L., Patel-Scneider, P. F., and Stein, L. A. (2003, February). Web ontology language (owl) reference version 1.0. Technical report, W3C Working Draft.

[3] Damianou, N., Dulay, N., et al. (2001). The Ponder Policy Specification Language. Policy 2001: Workshop on Policies for Distributed Systems and Networks. Springer-Verlag.

[4] Distributed Management Task Force, inc. (2005). Common Information Model (CIM) Standards, version 2.9.0.

[5] Kagal, L., Finin, T., and Johshi, A. (2003). A Policy Language for Pervasive Computing Environment. Policy 2003: Workshop on Policies for Distributed Systems and Networks. Springer-Verlag.

[6] Martinez Perez, G., Garcia Clemente, F.J., Gomez Skarmeta, A.F. (2005), Policy-Based Management of Web and Information Systems Security: an Emerging Technology, Idea Group Inc., in press.

[7] OASIS (2004, December). Extensible Access Control Markup Language (XACML), version 2.0, http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xacml

[8] The Rule Markup Initiative (2004, May). SWRL: A Semantic Web Rule Language Combining OWL and RuleML, version 0.6.

[9] Tonti, G., Bradshaw, J. M., Jeffers, R., Montanari, R., Suri, N., and Uszok, A. (2003). Semantic Web languages for policy representation and reasoning: A comparison of KAoS, Rei, and Ponder. The Semantic Web—ISWC 2003. Proceedings of the Second International Semantic Web Conference. Springer-Verlag.

[10] Uszok, A., Bradshaw, J., Jeffers, R., Suri, N., et al. (2003). KAoS Policy and Domain Services: Toward a Description-Logic Approach to Policy Representation, Deconfliction, and Enforcement. Policy 2003: Workshop on Policies for Distributed Systems and Networks. Springer-Verlag.

[11] W3C. (1999, February). Resource description framework (rdf), data model and syntax. W3C Recommendation.

[12] EU IST POSITIF (Policy-based Security Tools and Framework) Project, http://www.positif.org/

RDF Query for Policy Management

Eric Prud'hommeaux
W3C
eric@w3.org

ABSTRACT

Queries informing policy management and enforcement must address trust issues. The RDF query language SPARQL provides access to provenance information and a reasonably rich set of constraints. This document describes how a policy management system can use SPARQL to reliably investigate and enforce policies.

Keywords

RDF Query, SPARQL, Policy Management

Introduction

RDF was designed as a description language for web resources. As such, it is useful for describing policies associated with resources. The RDF Data Access Working Group is standardizing the SPARQL Query Language for RDF [SPARQL]. The SPARQL language, used to access simple triple stores or inferred triples, is useful for expressing/testing many practical policies.

It is essential that any agent enforcing policies trust its information. In a heterogeneous trust environment such as the semantic web, the chain of custody of policy data must be rigorously examined. Many RDF stores maintain the provenance of RDF data and SPARQL provides access to that information. Queries may interrogate the provenance of query solutions or specify that the solutions come from particular sources. This capability meets the reasonable requirements of semantic web policy agents.

Some policy languages, such as KAoS [KAoS], or XML Advanced Electronic Signatures (XAdES) [XAdES], include expiries or durations. SPARQL expresses numeric, string pattern, and datetime value constraints, which can be used to determine whether a given policy is applicable, or select solutions from only the relevant policies.

SPARQL also provides a set of logical expressions, including disjunction and optionally bound patterns. Used in conjunction with an operator to test whether a variable was not bound in a pattern, SPARQL provides a limited form of negation as failure (NAF). This feature is useful for practical reasons, limiting the amount of unwanted data the client consumes, but also to enable the client to avoid receiving information that would violate some policy.

SPARQL is not the only RDF query language, nor is it the most expressive. It is, however, the product of standardization; developers may count on reasonable conformance and vendor independence. It is beyond the scope of this document to compare the RDF query languages.

Provenance Constraints

A simple example of a policy is an access control list that associates a group of principals with a set of operations. The W3C site uses a simple ontology for expressing different people's right to perform HTTP operations on resources. For example, the ACLs for this document are expressed as:

```
@prefix rdf : <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix : <http://www.w3.org/2001/02/acls/ns#>.
[ a :resourceAccessRule;
  :access :racl, :head, :get, :options, :trace;
  :accessor <http://www.w3.org/Systems/db/webId?all=all>;
  :hasAccessTo <http://www.w3.org/2005/02/14-PMQuery/>
] .
[ a :resourceAccessRule;
  :access :chacl, :racl, :head, :get, :put, :delete, :connect, :options, :trace;
  :accessor <http://www.w3.org/Systems/db/webId?group=w3t_passwords>;
  :hasAccessTo <http://www.w3.org/2005/02/14-PMQuery/>
] .
```

This first ResourceAccessRule grants the group `http://www.w3.org/Systems/db/webId?all=all` the privileges to perform the HTTP operations HEAD, GET, OPTIONS, TRACE on the resource `http://www.w3.org/2005/02/14-PMQuery`. (The "racl" privilege is not an HTTP operation, but instead the meta-operation of reading the ACLs for that resource. The second ResourceAccessRule grants some additional HTTP operations, PUT and DELETE, to the group `http://www.w3.org/Systems/db/webId?group=w3t_passwords`. This group may chacl, change the ACLs for the resource.

Elsewhere the membership of the group `http://www.w3.org/Systems/db/webId?group=w3t_passwords` is enumerated, along with various credentials.

```
<http://www.w3.org/Systems/db/webId?group=w3t_passwords>
    :includes <http://www.w3.org/Systems/db/webId?user=eric> .
<http://www.w3.org/Systems/db/webId?user=eric>
    a :user ;
    :publicKey "30 82 01 0a 02 82 01 01 00..." .
```

Thus, the principal `http://www.w3.org/Systems/db/webId?user=eric` is a member of a group that has the ability to PUT this document, which is fortunate because PUT is the HTTP operation for updating a resource and `...eric` is the author of this document.

The author has appropriate credentials to prove that he is the `...eric` in the above list, and thus, has permission to change the documents. The group `http://www.w3.org/Systems/db/webId?all=all` is a special group understood by the W3C web servers to mean everybody, regardless of credentials or lack thereof.

When the user `eric` attempts to perform a `PUT` operation on this document on the W3C site, the site machinery verifies the credentials, verifies that the request action is within those allowed for this user for this resource, and grants access. So far, we haven't gone beyond what ordinary HTTP and DAV servers do every day. We have, however, made it expressible in a language that transcends server implementations and sites.

If a proxy site were to cache some or all of the W3C web site with the agreement that they would enforce the appropriate ACLs policies, they could use publicly available W3C ACLs policy information. If they were to query a public RDF aggregator, a semantic search engine, they would need to query for the provenance information associated with the policies:

```
PREFIX s: <http://www.w3.org/2001/02/acls/ns#>
ASK
 WHERE { GRAPH <http://www.w3.org/2005/02/14-PMQuery/,access?w3c_display=13>
         { ?policy s:access s:put .
           ?policy s:accessor ?group .
           ?policy s:hasAccessTo <http://www.w3.org/2005/02/14-PMQuery/> .
           ?group s:includes ?user .
           ?user s:publicKey "30 82 01 0a 02 82 01 01 00..." } }
```

This query simply specifies that everything in the access recipe must come from a source known to be authoritative for that resource. To make the scenario much more interesting, we can abstract the query, introducing multiple trust domains:

```
PREFIX s: <http://www.w3.org/2001/02/acls/ns#>
PREFIX meta: <http://www.w3.org/2002/xx#>
ASK
 WHERE {{ ?resource meta:keywords ?keywords .
          ?resource meta:abstract ?abstract .
         FILTER regex(?keywords, "SPARQL") &&
                regex(?abstract, "policy management") } .
  GRAPH <http://policies.example/knownSites.rdf>
        { ?resource s:policyAuthority ?policyAuth } .
  GRAPH ?policyAuth
        { ?policy s:access s:put .
          ?policy s:accessor ?group .
          ?policy s:hasAccessTo ?resource .
          ?group s:includes ?user .
          ?user s:publicKey "30 82 01 0a 02 82 01 01 00..." }}
```

Here we have asked the web for a document with a keyword "SPARQL" and the phrase "policy management" in the abstract. (This document has meta tags for the keywords and abstract.) Next we asked a trusted resource `knownSites.rdf` for the corresponding policy authority. Finally, we checked that authority to see what access privileges are extended to the user holding a particular public key. This query takes the appropriate conservative approach of failing to provide any access if the chain of trust cannot being established.

Expressing our policy in RDF allows us to develop arbitrarily complex trust chains. Evolving the authenticating software is as easy as mirroring new models in the SPARQL query, minimizing vulnerability to implementation errors and reducing deployment time and costs.

Mixing SPARQL and Policy Rules

Some policy languages exceed the expressivity of SPARQL, or are impractical to enumerate in SPARQL. Since SPARQL may operate over a graph created by inference, it can be used to access the inferences of any policy language that produces triples. A policy protocol using SPARQL may rely on it simply for a standard query interface, or for matching some or all of the rule conditions. The policy protocol may trade off between expressing the policy conditions in SPARQL vs. a rule language. For example, a REI [REI] policy expression can predicate a `Permission` on some conditions:

```
...
<action:Delegation rdf:ID="TimToCSMembers">
  <action:sender rdf:resource="&inst;TimFinin"/>
  <action:receiver rdf:resource="#PersonVar"/>
  <action:content>
    <deontic:Permission>
      <deontic:actor rdf:resource="#PersonVar"/>
      <deontic:action rdf:resource="#ObjectVar"/>
    </deontic:Permission>
  </action:content>
  <action:condition>
    <constraint:And>
      <constraint:first rdf:resource="#IsMemberOfCS"/>
      <constraint:second rdf:resource="#IsFacultyPrinting"/>
    </constraint:And>
  </action:condition>
</action:Delegation>
<constraint:SimpleConstraint rdf:ID="IsMemberOfCS">
  <constraint:subject rdf:resource="#PersonVar"/>
  <constraint:predicate rdf:resource="&univ;affiliation"/>
  <constraint:object rdf:resource="&univ;CSDept"/>
</constraint:SimpleConstraint>
<constraint:SimpleConstraint rdf:ID="IsFacultyPrinting">
  <constraint:subject rdf:resource="#ObjectVar"/>
  <constraint:predicate rdf:resource="&rdf;type"/>
  <constraint:object rdf:resource="#FacultyPrinting"/>
</constraint:SimpleConstraint>
...
```

SPARQL's terse syntax provides a very short expression of the above policy:

```
...
COLLECT ?sender ?receiver
 WHERE { ?permit rei:sender ?sender .
         ?permit rei:receiver ?receiver .
         ?permit rei:actor ?person .
         ?permit rei:action ?object .
         ?person univ:affiliation univ:CSDept.
         ?object rdf:type p:FacultyPrinting }
```

Languages with rule heads that are chained to further rules will not be well-represented in SPARQL as SPARQL is not a rules language. In such cases, it would only be useful to express as queries the questions that the application ultimately needs resolved, such as, "is the client in a class that has access to a given resource?"

Value Constraints

Many policy languages express a duration of validity. The following excerpt from KAoS states a policy update time stamp:

```
<policy:PosAuthorizationPolicy>
  <policy:controls rdf:resource="#GET" />
  <policy:hasSiteOfEnforcement rdf:resource="#w3site" />
  <policy:hasPriority>10</policy:hasPriority>
  <policy:hasUpdateTimeStamp>2006-01-01T00:00:00Z</policy:hasUpdateTimeStamp>
</policy:NegAuthorizationPolicy>
```

A SPARQL query looking for a current policy would rely on built-in dateTime comparison functions:

```
...
 WHERE { ?pol policy:hasUpdateTimeStamp ?update .
       FILTER ?update > 2005-04-17T13:34:52Z }
```

SPARQL also provides numeric comparison and string regular expression operators.

Negation

The PRIME project concerns itself with privacy and identity management in Europe. The design of SPARQL is informed by the needs of PRIME. In particular, a participant in the project, Thomas Roessler, submitted this use case [PRIME] to the RDF Data Access Working Group:

> A mobile phone provider offers location and contact information to third parties which, in turn, offer location-based advertising by mobile phone short message. An airline operates an airport restaurant as a subsidiary, which wants to advertise a special gourmet meal based on pork to members of the airline's frequent flier program who are nearby the restaurant, unless these have indicated halal, kosher, or vegetarian meal preferences.
>
> (Note that meal preferences give hints about religious convictions and health conditions, and should as such not be processed by a restaurant's advertising department.)

This use case demonstrates not a strict policy established by the traveler, but instead a sensitivity on the part of the airline toward the traveler's implicit policy. The SPARQL expressivity that this query leverages is the ability to filter solutions that do ***not*** include a specified statement (specifically, whether the person has expressed a preference for any of a set of special meals).

```
PREFIX flt: <http://someAirline.org/ns#>
SELECT ?smsAddr
 WHERE {<http://someAirline.org/flt217/20050215#> flt:traveler ?traveler .
        ?traveler flt:smsAddr ?smsAddr .
       OPTIONAL {?traveler flt:mealReq ?mealReq .
       FILTER !BOUND(?mealReq) }
```

The expressivity for NAF may seem awkward, but it is effective, and specifically addressed in the specification. By restricting the results to not include any solutions where the `traveler` had a `flt:mealReq`, the restaurant avoided moving more data than necessary and avoided having to local filtering of the data. These are obvious motivations for including NAF in the language. More importantly, the restaurant was able to avoid learning travelers' religious convictions by filtering out results which implied a religious conviction.

Designing for a query language that can express negation as failure allows policy ontologists to annotate entities with types of confidentiality. Given appropriate terms, people can indicate that some material is not intended for certain audiences, allowing school library web browsers to perform content selection. These terms can advertise increased privacy preferences; sympathetic agents can voluntarily comply with these preferences.

Decisions based on NAF are non-monotonic and must be regarded as uncertain. Agents working with partial knowledge or potentially incomplete inference can make incorrect decisions. In the above example, the restaurant may advertise a pork meal to a traveler who keeps kosher and who has not filled out his or her meal preference, or a library browser may let a user see content that would have hidden, had the browser had access to more information.

In many cases, policies have a conservative response to incomplete knowledge. For instance, a principal will be denied access to a resource if it is either not known that the principal should have access or it is known that the principal should not have access. For this reason, SPARQL's use of default negation is a practical alternative to using OWL's `complementOf` for classical negation.

Additional Expressivity: Disjunction and Optional

Disjunction and Optional graph patterns in SPARQL enable complex policies to be tested efficiently. Any query involving disjunction (called `Union` in SPARQL) or optional patterns could be expressed as a set queries with purely conjunctive graph patterns. This seemingly redundant expressivity allows many complex policies to be expressed as a single query, providing an intuitive interface and an efficient protocol.

SPARQL also has extensible value restrictions. The expressivity of the restrictions could be extended to calculations such as geographical radius, repeated temporal intervals or arbitrary mathematical functions of any set of parameters derived from the graph. These extension functions will not be available across all SPARQL implementations, but they can be used to express more complex policy tests within the SPARQL syntax.

Conclusions

This document has described two use cases where SPARQL's provenance queries support a rigorous enforcement of policies. The provenance information in the ACLs query allowed the suspicious agent to establish a chain of trust between a principal requesting access and a policy for that resource. The KAoS example showed how value constraints usefully enforce policies with expiries, and the PRIME example demonstrated how query support for negation allows policy data to be added to increase privacy.

Designers of policy protocols will need to provide a query mechanism of some sort to allow applications to act on

policies. Using SPARQL provides the advantages of using a standard query language, as well as the opportunity to leverage SPARQL implementations to provide some or all of the calculations. While every language makes trade-offs between complexity and expressivity, it is the author's opinion that deployment of SPARQL agents will provide a strong foundation for a policy aware Web.

REFERENCES

[SPARQL] Andy Seaborne, Eric Prud'hommeaux, SPARQL Query Language, , http://www.w3.org/TR/rdf-sparql-query/

[KAoS] M. Johnson, et. al., KAoS Semantic Policy and Domain Services: An Application of DAML to Web Services-Based Grid Architectures, , http://www.agentus.com/WSABE2003/program/johnson.pdf

[REI] Lalana Kagal, Rei Ontology Specifications, Ver 2.0, , http://www.cs.umbc.edu/~lkagal1/rei/

[XAdES] Cruellas et. al., XML Advanced Electronic Signatures (XAdES), , http://www.w3.org/TR/XAdES/

[PRIME] Thomas Roessler, UNSAID and OR use cases, from PRIME, , http://lists.w3.org/Archives/Public/public-rdf-dawg-comments/2004Nov/0016

Application Report: An extensible policy editing API for privacy and identity management policies

Giles Hogben,
European Commission,
Joint Research Centre,
Via Enrico Fermi 1,
21020 VA, Ispra, Italy,
+39 0332789187
giles.hogben@jrc.it

ABSTRACT

This paper describes an open source policy editing API, which has been developed for use with privacy policies including P3P1.1 policies, semantic web privacy policies and enterprise privacy policies. The API has been designed to be extensible to a wide range of policy editors for access privacy and identity management. It is also designed to support the use of ontologies to specify validated and updateable human readable translations of policy elements. It provides libraries for editing any kind of policy which is associated to URI resources and which describes behaviour in terms of discrete statements. The paper gives a brief overview of new features of the API which have allowed us to generalize its application.

Categories and Subject Descriptors

D.2.6 [**Software**]: Programming Environments, Graphical Environments

General Terms

Management, Design Security, Human Factors, Languages

Keywords

Policy Authoring, Applications, Semantic Web Groundings

This work was supported by the IST PRIME project; it represents the view of the authors only.

1. INTRODUCTION

This paper is a short application report on a policy editing framework produced by the Joint Research Centre as part of the PRIME project. Many policy editors already exist in the context of P3P 1.0 [1], so we concentrate in this paper on the innovations we have introduced in order to extend the policy editing API from a P3P editor to other types of policy editing such as enterprise access control (privacy layer). We also discuss the introduction of a legal hints mechanism.

2. Editor usage scenarios

The editor has been designed for the following use cases:

2.1 P3P 1.0 Policies

The editor is designed to be able to edit P3P 1.0 policies and to output Policy Reference Files specifying which P3P 1.0 policies apply to which sets of web resources. It is also designed to be able to validate policies, and to give legal hints to policy writers about points of interest in their jurisdiction.

2.2 P3P 1.1 Policies

The editor is designed to integrate the enhancements provided by P3P1.1.[2] These are mainly in the area of the human readable strings corresponding to policy concepts, but also include a new data schema format.

2.3 Semantic web P3P style policies

The editor API is also designed to be able to produce policies using P3P semantics translated into OWL (as described in [3]).

2.4 Enterprise access control (XACML style) policies

This is the most challenging adaptation of the editor. We decided that there are sufficient similarities in the model of P3P and experimental privacy enhanced access control policy languages such as EPAL [4], and [5] to be able to justify an adaptation of the API to support editing of this type of policy.

The specification we are using is the working specification for the Prime [6] project access control module. Although this is not currently available publicly, it is however close to the specification described in the publicly available document [4] in terms of policy editing requirements. The working specification of [6] conforms to the requirements stated in [7].

In general terms, the policy framework comprises Access Control Policies, Data Release Policies and Data Handling Policies. All these operate over RDF data stores and use prolog type semantics encapsulated in XML syntax for creating inferences over access-control rules.

Throughout this document, we refer to this type of policy as "XACML style" (XACML:Oasis standard – vide http://www.oasis-open.org) as this is the closest existing standard (apart from the W3C member submission, EPAL [4]). It is important to note that the API requires access control policies of this type to operate over RDF data with data typing via RDF/OWL ontologies or P3P data schema syntax.

3. Policy editing interface API Components

3.1 Common features

Any API design always plays off simplicity against general applicability. It is clearly not possible to build an API suitable for building any conceivable type of policy. However, we have managed to abstract the features of privacy and IDM policies, including enterprise access control policy languages for privacy in

order to maximize reuse. The following features are common to all types of policy and therefore represent the building blocks of the policy editor API. The API uses the MVC (Model View Controller) paradigm, which divides the management of the user interface and storage objects into Business (Model), Interface (View) and Events (Control). Before reading the following sections, the reader may wish to refer to the end-to-end walkthrough in section 4.

3.2 Resource-policy binding

A common feature of all the above policy types is the need to associate rules or practice statements with groups of resources. This defines which policy should be applied to which resources in the data space. P3P policies, for example, use Policy reference files to associate XML P3P policies with parts of web sites which are resources groups. We found however that this model can also be extended to semantic web and XACML style policies as defined in section 2.4

XACML style policies are of 2 kinds:

a. Access control policies, which associate subject, condition, action rules with *abstract* data types drawn from an ontology describing data types and credentials. A set of policies applies in a given context defined by the administrator. Such policies contain rules of the form:

   For data or credentials of type "prime:e-Healthcard", if the accessing subject is a doctor who is employed in hospital x, allow access with the following obligations….

b. Access control policies which represent user preferences on data collected. These apply rules and obligations to specific data instances.

   Such policies apply rules and obligations to specific data instances. For example

   Delete data item X, after 10 years

The API applies the same model to all of the use case policies. In each case, the editor is required to apply rules or statements to *groups of resources*. In the case of the XACML style policies, the groups of resources are either OWL concepts (defined by a URI – scenario a. above) or RDF triples in a datastore (defined by reification ids, or an RDF query – scenario b. above). We have therefore abstracted the policy-resource association function in the API as follows.

Every editor has 3 sub-windows (see figure 1 and 2) which are managed by a set of extensible classes according to the MVC model.

1. The resource grouping window (top left):

Shows a list of resource groups organized by namespace or site domain. The underlying business object is the same for all types of policy (an XML object stores the resource groups as named patterns according to namespace), but these business objects can then be transformed into customized mapping objects. In other words, the business object abstracts Policy Reference Files for P3P1.0 and allows it to be mapped into other format (e.g. XACML targets).

The user interfaces used for capturing patterns may differ from the default implementations but can customize API implementationsn by extending the PatternInterface class, which captures the specification of the content groups from the user. Each resource group group defines a space of resources which can be either web URI's (P3P and Semantic Web P3P), Ontology concepts (XACML style a.) or RDF triple sets (XACML style b.). In P3P, this corresponds to an area of a web domain or set of domains. In semantic web based access control, this corresponds to a space of resources.

2. The policies window:

Shows the policies available. This is just a list of policy names associated to their logical identifiers (file system paths), which can be dragged onto resource groups and can be double clicked for editing the content of the policy, using a class conforming to the policyeditor interface. This interface is completely independent of the format and content of the policy and it is therefore not foreseen that this would need to be extended.

3. The mappings window:

By dragging a policy onto a resource group, the user can associate policies to resources. This association is then automatically displayed in a third window, the mappings window. The storage format for mappings is abstracted from the particular format it will eventually be output in. For example in the case of P3P, this abstraction will be mapped to a Policy Reference File. In the case of semantic web based access control, it may for example be mapped to a target statement within a policy. The API implements this abstraction using the "publish" method of the mappings tree, which currently only implements the transformation to a P3P Policy Reference File, but can be overridden to provide other transformations for example using XSLT to provide target statements within XACML style policies.

Figure 1. P3P scenario

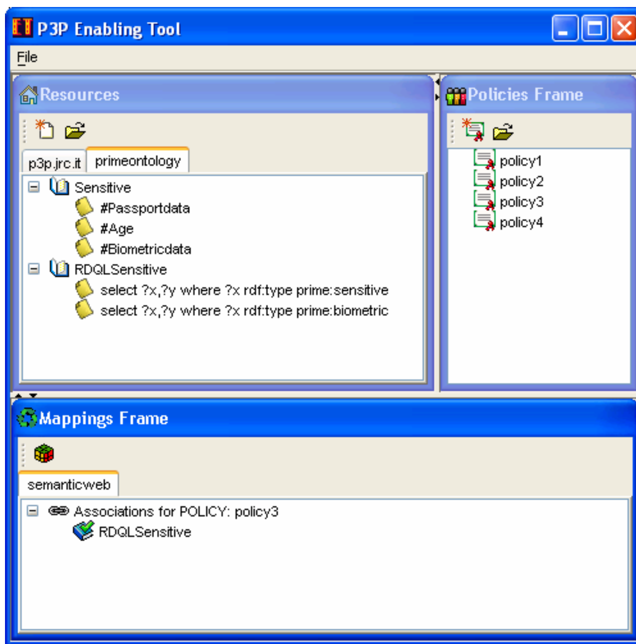


Figure 2. Semantic Web Scenario

3.3 Statement handler

Upon opening a policy for editing, the user is presented with a list of statements. Statements are derived directly from the XML policy document in memory defining the policy being edited. So in terms of the MVC architecture model, the XML document is the model (Business object) and there is no further abstraction.

The API provides a statement management package, which includes a class which abstracts the visualization of statements. The class StatementType defines how human readable strings are extracted from the XML document by means of a query string. It also defines not only the content of the strings, but also how they will be organized for display to the user.

In order to achieve this, the StatementType class defines the list of attributes into which the statement is broken down. These may, but are not required to correspond to XML attribute or tag names. For example P3P StatementType definitions define how to extract CONSEQUENCE, DATA, PURPOSE, RECIPIENT and RETENTION attributes of the statement by means of XPATH queries. This is done as follows:

Each Attribute object specifies XML or RDF queries and/or procedural code which define its relationship to the user interface. This allows the editor builder to define new types of statements and attributes and their display to the user simply by defining their attributes and queries which extract the display text.

Each attribute in a StatementViewer's AttributeList Array has a getHRQueryString method, which returns the results an RDF or XML query over the policy document (and may transform this using Java code for display). This method returns the text to display to the user to summarize the value of that attribute.

For example for P3P statements, getHRQueryString() for each attribute returns a conversion to string of the node names returned by the XPath queries:

".//*[local-name()='CONSEQUENCE']/*",

".//*[local-name()='DATA']/*"

".//*[local-name()='PURPOSE']/*"

Etc…

Separate XML and RDF flavours of these classes have been defined in order that the query language is flexible.

Once the StatementViewer object for the policy editor is defined, the API automatically creates a table displaying all non-hidden attributes. It is assumed that statements are logically independent objects i.e. that no inter-statement data (e.g. OR and AND) needs to be displayed. These kind of booleans may be included in a language but statements should be defined on a level whereby the booleans are contained within each statement but do not connect statements. StatementTypes can also be created dynamically if the number of attributes is variable.

Attributes can be assigned visible or non visible status. For example a P3P editor would not want to display the consequence attribute of a statement in the statement summary table, so this would be assigned hidden status.

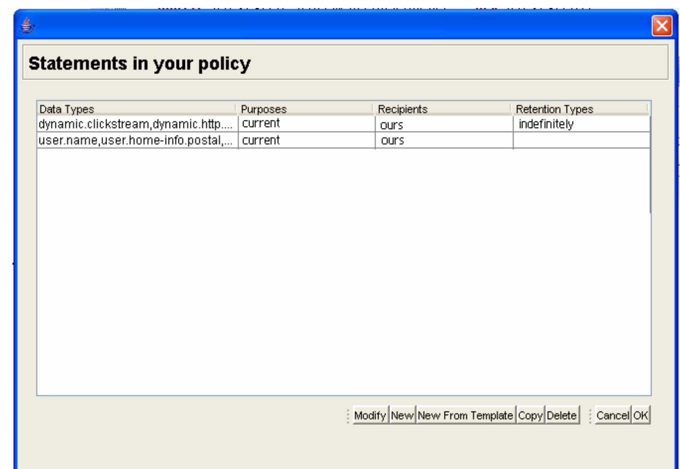


Figure 3. StatementViewer

3.4 Statement wizard

StatementType attribute arrays (see 3.3) also define the stages of the Statement wizard. The statement wizard proceeds through a series of windows on a per attribute basis. The attribute array of the StatementType therefore defines the stages of the statement wizard. Statement wizards can also be defined using a swing card layout to increase efficeincy.

Each attribute has a getViewer method, which uses classes extending the abstract class AttributeEditorWindow to specify the editing window to be displayed for that attribute.

The API provides 3 implementations of AttributeEditorWndow.

1. Typically a statement attribute editing window is a flat set of possible values displayed as a set of strings with checkboxes next to them. This uses a ConstrainedValueWindow The human readable strings for this type of attribute editing window may be defined according to an XML document or OWL ontology (See section 4.5)

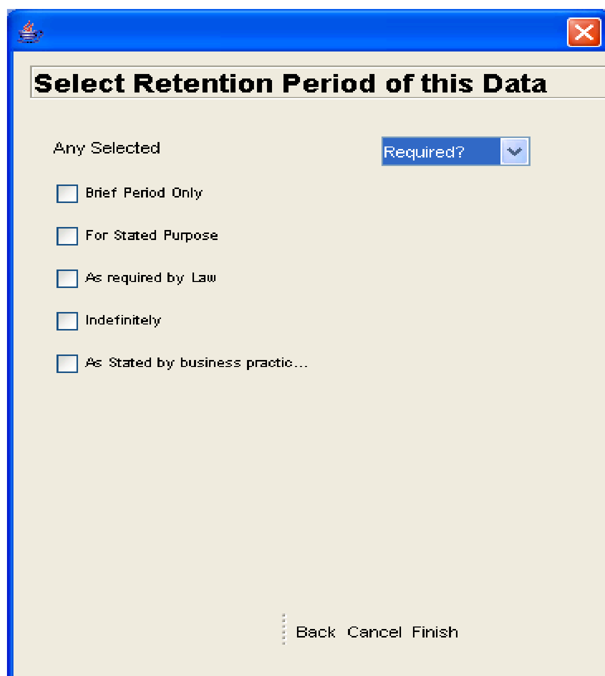


Figure 4. ConstrainedValueWindow

2. A datatype editing window (See 3.5)

3. A plaintext editing window (e.g. for P3P consequence). The type of window required is specified. Special editing windows can be created to replace the default implementations.

The above default implementations can be used to define attribute viewers.

3.5 Data typing schemes

Privacy and access control policies typically have to present the user with an ontology hierarchy of increasingly detailed data types to select from (an XML document is also understood here as an informal ontology). The editor API abstracts this process so that different data schemas can be used within the same view window as long as they have a structure representable by a JTree.

The data type editing window displays the data type tree on the left hand side and a list of selected types on the right hand side. The user simply moves types from the tree into the list on the right hand side. The elements in the list of types selected combine to make a custom type. The list element objects store the tree path as well as the leaf node selected so that they can be edited later.

The user can dynamically select different source files for the tree representation.

The API provides the abstract DataSchemaTreeViewer class which has the abstract LoadTree() method. This defines how the data typing schema is mapped onto the JTree. We will provide 3 implementations of this method - for P3P 1.0 [1], 1.1 [2] and OWL [8] versions of the P3P data schema. Once this mapping has been made, the chosen types are be inserted directly into the policy without further reference to the schema. New schemas of the given type can be loaded dynamically.

Future work would include an editor for creating custom data schemas. Figure 6 shows the datatype editing window with the P3P base data schema loaded in the left pane and the types selected in the right pane. Above the schema tree is a button for loading a new schema tree.

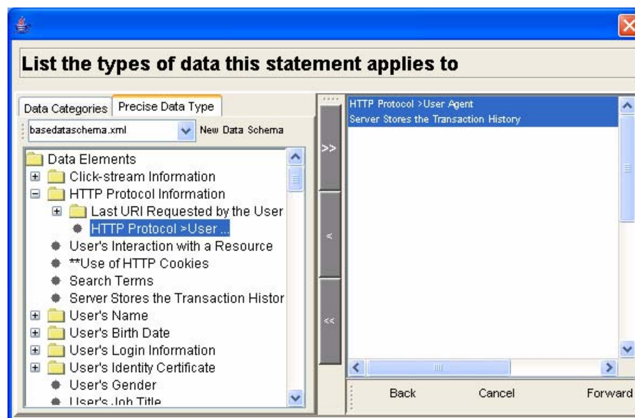


Figure 6. Datatype editing window

3.6 Linking of option handling and human readable strings to ontologies.

Because of the importance of displaying human readable translations of attributes in a consistent way [see 9], label strings for ConstrainedWindow [See Section 3.4.1] implementations are taken from an XML specification document which may be either an RDF ontology, or an XML document.

In the case of P3P, the latter is just a translation into XML of the Human readable translations in the draft P3P 1.1 specification [9]. The checkboxes and their labels are created dynamically from this document by the getAlternatives method of the Attribute object.

The exact method of associating human readable strings to checkboxes depends on whether an XML specification document is used, or an RDF ontology.

1. For an XML specification: each Alternative in the Attribute's alternative array is an object which can return an XML fragment from its getXML() method. This is the XML fragment to be inserted into the statement being edited if the choice is selected. It may also be derived from a query over an XML schema in order to minimize programming work in case of changes to the specification schema.

   Each alternative also has a getHumanReadable() method which performs a query over a human readable equivalences document in well-defined format, in order to return the human readable string for that alternative. In our implementation, the equivalences are stored as fragments of the document with sibling CDATA text nodes containing the human readable equivalent.

   For example the PURPOSE translations are stored as follows:

```
<equivalence>
    <node><ours/></node>
    <hrstring>Only parties related to this site</hrstring>
<equivalence>
```

   The user's choices are then automatically saved to the Statement's base document when the user click's OK by inserting the node associated to the alternative into the statement.

2. For an OWL ontology (parsed by the Jena [10] API): The procedure for extracting and displaying the alternatives is

the same as 1. except that the query extracting the equivalence will be an RDF query rather than an XPATH query.

3.7 Use of XSLT transforms for policy views.

The base window of the policy editor shows a set of views of the policy being edited. These views are produced by XSLT transforms which define views such as for example Human readable, statement summary and To Do (a list of incomplete parts of a policy). Another important view is the legal hints view (See next section).

The policy views can also be produced using prolog style rules running over RDF (e.g. using Jena rules). This then outputs a set of statements inferred from the policy, with a transformation to natural language. (See also 3.8).

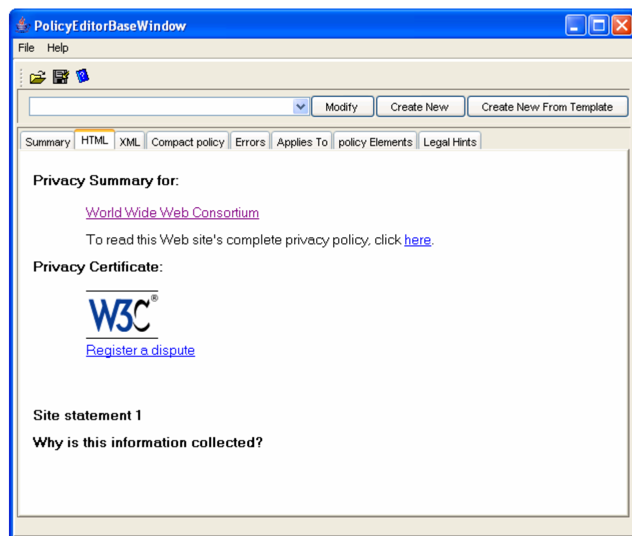


Figure 7. Policy Transformation View (Mirrors view in MS Internet Explorer Privacy Report)

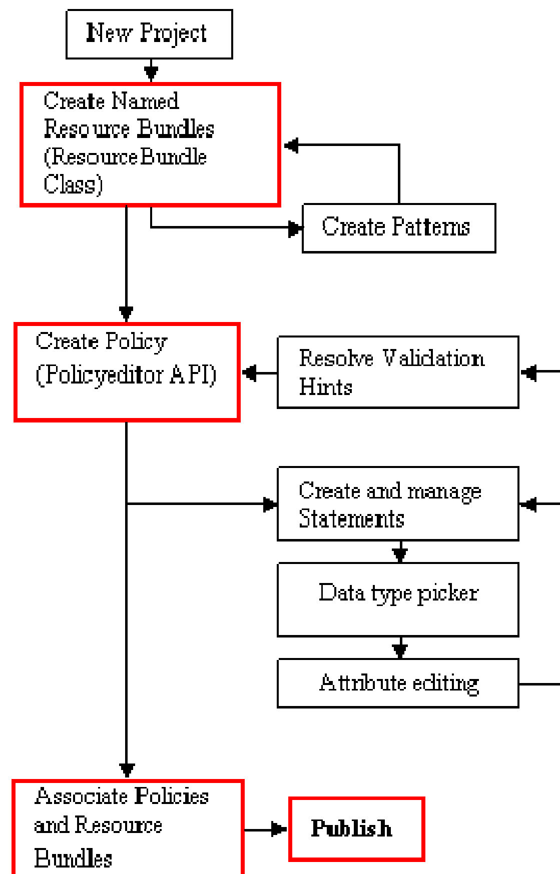
3.8 Legal hints mechanism

In Europe particularly, regulatory bodies have been concerned about the possibility of privacy languages which enable policy authors to write policies which specify data processing practices which are illegal in the author's jurisdiction.

One important policy view provided by the API is the legal hints view. This is based on XSLT transformation rules or RDF based rules which provide users with comments on the policy they have created based on legal knowledge encoded in the rules. It is envisaged that XSLT transforms or other inference rules will be imported based on jurisdiction.

For example if a user creates a P3P policy which says that they will use email data to contact the user without an opt-out (which would be illegal in Europe according to [11]), the legal hints can inform the user that this is an illegal practice in Europe. It is possible in future versions that these rules could also offer a set of corrections to the user.

4. Process walkthrough



5. Conclusion

The API described above provides a useful tool for policy authoring in many scenarios in the field of policies for the web. It provides an extensible framework for policy-resource association, statement management and statement composition. It also provides a framework for providing legal hints and different policy views.

6. REFERENCES

[1] Platform for Privacy Preferences Specification, Cranor et al. ,Platform for Privacy Preferences, W3C Recommendation, http://www.w3.org/tr/p3p

[2] Cranor, Dobbs, Egelman, Hogben et al., The Platform for Privacy Preferences 1.1 (P3P1.1) Specification W3C Working Draft 4 January 2005 http://www.w3.org/TR/2005/WD-P3P11-20050104/

[3] Hogben, G. *P3P Using the Semantic Web (OWL Ontology, RDF Policy and RDQL Rules)*, W3C Working Group Note 3 September 2000, http://www.w3.org/P3P/2004/040920_p3p-sw.html

[4] Powers, C., Schunter, M., Enterprise Privacy Authorization Language (EPAL 1.2), W3C Member Submission 10 November 2003, http://www.w3.org/Submission/EPAL/

[5] Piero A. Bonatti Ernesto Damiani Sabrina De Capitani di Vimercati Pierangela Samarati, A Component-based Architecture for Secure Data Publication http://www.acsac.org/2001/papers/114.pdf

[6] Privacy and Identity Management in Europe, European Research Project, see http://www.prime-project.eu.org

[7] Wilikens, M. et al., PRIME Requirements - Part 3: Application requirements, http://www.prime-project.eu.org/public/prime_products/deliverables/pub_del_D01.1.a.part3_ec_wp03.1_V5_final.pdf

[8] Hogben, G., Describing the P3P base data schema using OWL, Proceedings of PM4W, WWW2005 workshop.

[9] See Section on User Agent Guidelines, P3P 1.1 Draft Specification, http://www.w3.org/TR/2005/WD-P3P11-20050104/#ua

[10] Jena semantic web API, HP Labs, see http://jena.sourceforge.net

[11] EU Directive 2002/58/EC on Privacy and Electronic Communications

Policy based access control for an RDF store

Pavan Reddivari
University of Maryland,
Baltimore County
Baltimore MD USA
pavan2@csee.umbc.edu

Tim Finin
University of Maryland,
Baltimore County
Baltimore MD USA
finin@csee.umbc.edu

Anupam Joshi
University of Maryland,
Baltimore County
Baltimore MD USA
joshi@csee.umbc.edu

ABSTRACT

Resource Description Format (RDF) stores have formed an essential part of many semantic web applications. Current RDF store systems have primarily focused on efficiently storing and querying large numbers of triples. Little attention has been given to how triples would be updated and maintained or how access to store can be controlled. In this paper we describe the motivation for an RDF store with complete maintenance capabilities and access control. We propose a policy based access control model providing control over the various actions possible on an RDF store. Finally, we discuss on how the Hypertext Transport Protocol (HTTP) and its extensions can be used to provide communication with the store.

General Terms
Management, Experimentation, Security.

Keywords
RDF Store, Access control, Policies, HTTP

1. INTRODUCTION

The Semantic Web is leading us to a world of information sharing, by enabling distributed knowledge aggregation and creation. Thus many semantic web applications require management of large amounts of semantic data and there have been ample number of RDF store implementations, which are capable of storing large number of RDF triples. We believe that for RDF store to be more functional and widely deployed in applications they ought to provide a mechanism to specify restrictions on creation, modification and browsing of the knowledge. Current implementations of RDF stores such as Redland and Kowari are mostly focused on the aspect of scalability and very rarely address the issue of security and access control.

In this paper we will map out a set of actions which are required to completely manage a store, and describe a model of access control to permit or prohibit these actions. In this model, agents make requests to perform actions against the RDF store and the decision whether or not to carry out the requested action is governed by an explicit policy

Policies are defined by a collection of policy rules governing whether the action is permitted or prohibited. Examples of actions include inserting a set of triples into the store, deleting a triple, and querying whether or not a triple is in the store. The conditions on a policy rule are a Boolean combination of constraints on the agent requesting the action, the type of action requested, the history of previous actions, the contents of the store, and the possible effect on the store and its model.

Informal examples illustrating the range of policy rules we would like to support include the following.

- *Only agents assigned to an editor role are allowed to insert or delete triples.*
- *An agent can only delete triples it previously inserted.*
- *An agent is only allowed to 'add properties' to classes it introduced.*
- *No agent may see any values of a 'social security number' property.*
- *No agent may insert a triple that allows any agent to infer a patient's 'HIV status'.*
- *An agent may modify any data about itself.*
- *An agent may not add an instance of a foaf:Person without providing a foaf:name property and either a fof:mbox or foaf:mbox_sha1sum property.*

In the remainder of this paper we describe our preliminary design for RAP, a simple RDF access policy framework. An initial prototype, implemented using Jena [11], is under construction at the time of this writing.

2. RDF Graph

In this section we review the RDF model [8,9,10] and identify a set of primitive actions that can be performed on a RDF graph. An RDF graph is composed of three types of node, a RDF URI references node (N), a Blank node (B) and a RDF literal Node (L). The edges (E) in the graph are directional and each edge also is associated with a URI [1]. The triple in a RDF graph can be described as (subject, predicate, object) $\in (N \cup B) \times E \times (N \cup B \cup L)$.

The basic primitive manipulations on this graph can be performed by one of the following ways:

1. Add a triple (subject, predicate, object) to graph such that both subject and object node did not previously exist in the graph prior to this addition. This leads to addition of two new nodes and an edge to the graph.
2. Add a triple (subject, predicate, object) to graph such that either subject or object node did not exist in the graph prior to this addition. This leads addition of one new node and an edge to the graph.
3. Add a triple (subject, predicate, object) to graph such that both subject and object node exist in the graph prior to this addition. This leads addition of an edge to the graph.

4. Delete a triple (subject, predicate, object) from the graph. This will lead to the predicate edge being removed from the graph and the subject and object nodes may be removed or not, depending on whether they are part of any other triple or not.

In addition, we will introduce and make use of several compound actions and indirect actions. Compound actions include the action of updating or replacing one triple with another, the action of inserting a set of triples, and the action of deleting a set of triples. Indirect actions cover the introduction or removal of a triple in the model through the addition or deletion of separate tripe into the explicit store.

3. RDF store Actions

We need to identify the set of actions which are needed to maintain an RDF store. The access control policies will control permission and prohibition to these actions. Maintaining RDF store involves four basics actions: Adding, Deleting, Updating and Searching for triples.

3.1 Additions to the store

These actions allow agents to add new information to the RDF stores.

- **insert(A, T)**: Agent A directly inserts triple T into the graph. This action is used by the Agent to add minimal information into the store, such as '*foaf:Person* is a subclass of *foaf:Mammal.*
- **insertModel(A, T)**: Agent A insertModels triple T If Agent A performed **Insert(A, T1)** and the inserting of T1 enables the store to infer that triple T is in the model. This action leads to indirect addition of knowledge by the user, such as after adding the triple *foaf:Person* is a subclass of *foaf:Mammal,* addition of triple X Instance of *foaf:Person* leads to indirect addition of X *rdf:type foaf:Mammal*. Constraints on this action are useful in preventing an agent from adding information indirectly.
- **insertSet(A, {Tc})**: Agent A insertSets a set of triples {Tc} if Agent A inserts all the triples in {Tc} into the store together. It is possible that Agent A is not allowed to add the triples in set {Tc} individually. This action can be used to ensure that the agent always inserts a set of triples which are related, for instance an agent may not add an instance of a foaf:Person without providing a foaf:name property and either a fof:mbox or foaf:mbox_sha1sum property .

3.2 Deletions from the store

These actions allow Agents to delete information from the stores

- **remove(A, T):** Agent A directly removes triple T from the graph. This Action would be used by the Agent to remove minimal information from the store, such as ?X *emp:WorksFor* of *foaf:CompanyX.*
- **removeModel(A,T):** Agent A removeModels triple T If Agent A performs **Remove(A,T1)** and the store cannot infer triple T after the removal of T1.
- **removeSet(A, {Tc})**: Agent A removeSets a set of triples {Tc} If Agent A removes all the triples in {Tc} into the store together. It is possible that agent A is not allowed to remove the triples in set {Tc} individually. This action is useful when you do not want the agent to remove something unless it is removing something else too. For instance you might want to enforce a policy that unless you are deleting the entire employee record, the social security number property can not be removed.

3.3 Updates to the store

The update action provides a mechanism to update particular triples in an RDF store. While this could me modeled as a combination of a delete and an insert, it is convenient to have an update that acts as a single transaction.

- **update(A, T1, T2):** Agent A directly replaces the triple T1 with the T2.

The update action is useful in cases when you want the user to have the modification rights without the deletion right as in the case where you want your employees to be able to modify their cell phone triple but not delete it.

3.4 Querying the store

Two actions are defined to describe an agent's actions of querying or searching an RDF store, covering both direct and indirect access.

- **see(A, T):** Agent A sees triple T if it returned in the response to one of A's queries to the store. This action will allow users to browse the knowledge in the store.
- **use(A, T):** Agent A uses triple T if it is used by the store in answering one of A's queries. This action is useful when you want the user to be able to restrict what information is being used to answer agent A's query.

Both these actions are independent of each other, even though it might appear that if Agent A can 'see' triple T, then Agent A can 'use' triple T but that is not the case. For example consider three triples T1, T2 and T3. Let us assume that you can infer T3 only by using T1 and T2. If Agent A can see T1 but cannot use it and can use T2 but cannot see it, then Agent A will not be able to see T3.

4. RDF Store Structure

An RDF store typically contains domain specific RDF schema and RDF data. In the RAP framework, the RDF store is also used to store the policy, represented in RDF, as well as other data and meta-data needed for the policy rules.

The agents are also represented in RDF and are parts of the domain specific knowledge. This representation of agents is used in the policy specifications. The RDF store will also maintain metadata about the triples in the store, like the creator of the triple

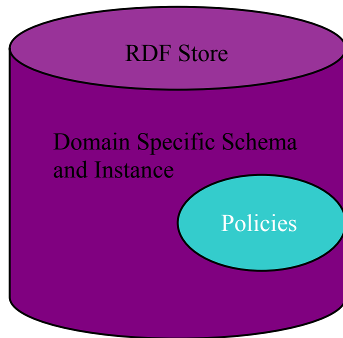


Figure 1: RDF Store

5. Policies

In the RAP framework, a policy is defined by a set of policy rules that together specify if an agent's specific requested action is permitted or prohibited. Following Rei [3,4], a query about the status of an agent's specific action request might have any of four outcomes: unknown, proven to be permitted, proven to be forbidden, and proven to be both permitted and forbidden.

Like Rei, RAP allows a policy to include meta-rules that can be used to resolve the two problematic cases. The two kinds of meta-rules that RAP allows are a *default policy* and a modality preference. Together, these can be thought of as implicit policy constraints.

proven prohibited \ proven permitted	no	yes
no	**?**	**permitted**
yes	**prohibited**	**conflict**

Figure 2. In reasoning about an action, four outcomes are possible. An uncertain or conflicted outcome may be resolved my meta-policy rules

The default policy, if specified, determines what happens in the upper left quadrant of the decision matrix shown in Figure 2. If *default(permitted)* is true then any actions not explicitly prohibited are permitted. If *default(prohibited)* is true, than actions not expressly permitted are prohibited. One of these two default settings must be selected (typically *default(prohibited)*).

The modality preference specifies what to do when we are in the lower right quadrant of the decision matrix. If *prefer(permitted)* is true, then an action that can be proven to be both permitted and prohibited is considered to be permitted*. If prefer(prohibited)* is true, then prohibitions dominate permissions. One of these two settings must be selected, typically the latter.

Explicit policy rules are used to permit or prohibit an agent from performing a class of actions on the RDF store. The general form of a policy rule is "*Modality(Action(A,T)) :- Condition*" where Modality is one of *permit* or *prohibit*, Action names an action, A identifies an agent and T identifies a triple. Condition is a Boolean combination of simple constraints expressed as RDF triples. The Triple (T) represented in the head of the policy has the form (subject, predicate, object). Wild card character "?" can be used in the triple pattern, a triple of the form (?, ?, ?) would thus hold true for all the triples.

The Specification of the agent is defined by the agent representation in the domain knowledge. This allows us to specify policies using agent specific data.

The Condition for the policy can be specified either using the metadata about the triples, the triple data itself, the Agent data or by combing both Agent and triple data. Conditions can be combined using Boolean AND (&), OR (|) operations.

Metadata specific conditions. The conditions in the policy can be specified based on the metadata about the triples that the store maintains. The kind of metadata to be collected is specific to the store implementation.

permit(insert(A,(?,rdfs:type,C))) :- createdNode(A,C)

The above policy will allow Agents to create instances of classes only if they had created those classes. The createdNode (A, C) returns true if Agent A had created triple T which created node C.

Triple specific conditions. The policies can also be specific to the kind of triples being added.

prohibit(see(A,(?,emp:salary,?))
prohibit(see(A,(?,P,?))) :- rdfs:subProperty(P,emp:salary)

These policies will prohibit agents from seeing the value of the emp:salary property, its sub properties or any equivalent property. The rdfs:subProperty(P,emp:salary) returns True if predicate P is defined to be an rdfs:subProperty of emp:salary.

Agent specific conditions. The attributes of the Agent could also be used in the conditions of policy. The Agent's representation would be specific to the domain

permit(see(A,(?,emp:salary,?)):-
existTriple(A,rdfs:type,emp:Auditor)

This policy will permit an Agent A to see anyone's salary as long as the Agent A is an auditor.

Agent and Triple specific conditions. The conditions in the policy could be tied to both the Agent attributes and the triple data being acted upon.

permit(update(A,(P,emp:salary,?),(P,emp:salary,?)) :-
existTriple(A,emp:Supervisor,P)

This policy will permit an Agent A to update salary of P as long as A is the supervisor of P.

Custom Predicates. There are certain custom predicates which might be helpful in writing access policies. Some of them have already been discussed such as createdNode(A,C), rdfs:subProperty(P,emp:salary). Another important predicate is schemaPredicate(P) which would return true if P is a predicate used to define RDF schema level information (e.g., rdfs:subClass, rdfs:domain, etc).

prohibit((insert(A,(?,P,?))) :- schemaPredicate(P).

This policy will prevent Agent A form changing the schema of the RDF store.

Delegation. As the Policies are represented in RDF and are stored in RDF store, delegation of policies can be achieved by creating Meta-polices, which are policies governing the policy triples in the store.

6. Architecture

We believe that the clients should be able to access the RDF store like any other website on Web. To enable this we propose the use of HTTP methods to access the RDF store.

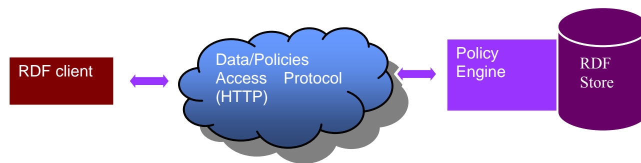


Figure 3: Proposed Architecture

HTTP seemed the optimal choice because of its synergy with current web and its wide acceptance.

We use the different HTTP Methods to access and modify the RDF store, the body of these methods would contain the XML serialized RDF.

The PUT Method is used for inserting the triples. All the triples that are to be inserted are sent in the body of the method. The store treats all these triples as one set and if that is prohibited, it then inserts each triple individually. All those triples which were prohibited from inserting are returned in the response message.

The Delete Method is used for removing the triples. The POST method would be used to query the store, the body of the POST method will contain the SPARQL query.

7. Status and conclusions

We have described a policy based framework to provide access and update control for an RDF store. Access and modifications are governed by a policy expressed as a collection of policy rules. Each rule defines a constraint on a class of actions that can depend on the actor and the content of the triples involved. The framework is currently being implemented using Jena [11].

8. REFERENCES

[1] Daniel Weitzner, Jim Hendler, Tim Berners-Lee, and Dan Connolly, Creating a policy-aware web: Discretionary, rule-based access for the World Wide Web. In Elena Ferrari and Bhavani Thuraisingham, editors, *Web and Information Security,*.

[2] Berners-Lee, T., Hendler, J., and Lassila, O. The Semantic Web, *Scientific American*, May, 2001.

[3] Kagal, L., Paoucci, M., Srinivasan, N., Denker, G., Finin, T., and Sycara, K. (2004). Authorization and Privacy for Semantic Web Services, IEEE Intelligent Systems (Special Issue on Semantic Web Services), July, 2004.

[4] Lalana Kagal (2004). A Policy-Based Approach to Governing Autonomous Behavior in Distributed Environments", Phd Thesis, Department of Computer Science and Electrical Engineering, University of Maryland Baltimore County, September 2004.

[5] J.M. Bradshaw, et al., (2003). Representation and Reasoning for DAML-Based Policy and Domain Services in KAoS and Nomads, Proceedings of the Conference on Autonomous Agents and Multiagent Systems, ACM Press, 2003.

[6] Claudio Gutierrez, Carlos Hurtado, and Alberto Mendelzon. Formal aspects of querying RDF databases, First VLDB Workshop on Semantic Web and Databases, Berlin, Germany, September 7-8, 2003

[7] Berners-Lee, T., Fielding, R. and Frystyk, H. (1996). "Hypertext Transfer Protocol" HTTP/1.0," HTTP Working Group, Feb. 1996.

[8] Ora Lassila and Ralph Swick, Working draft, W3C, 1998. Resource description framework (RDF) model and syntax specification, Edit.

[9] Patrick Hayes, editor (2003). RDF Semantics, W3C Working Draft, 23 January 2003.

[10] Dan Brickley, R.V. Guha. RDF Vocabulary Description Language 1.0: RDF Schema, W3C Working Draft 23 January 2003, Edit.

[11] McBride, B., Jena: a semantic Web toolkit, IEEE Internet Computing, v6n6, pp. 55-59, November 2002.