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.

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-

WWW2005, May 10–14, 2005, Chiba, Japan.

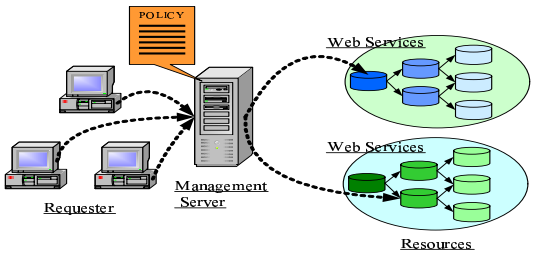

Figure 1: Aggregation Web Services Model

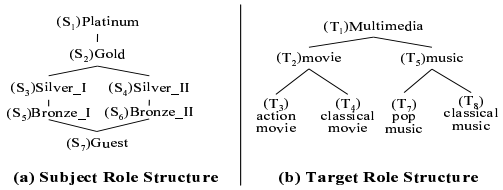

Figure 2: Examples of Role Structures

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:

$$\begin{array}{ll} \text{Policy r1}: & \texttt{Auth}+(\texttt{Bronze\_I}, \texttt{movie}, \texttt{play}) \\ \text{Policy r2}: & \texttt{Auth}-(\texttt{Gold}, \texttt{movie}, \texttt{play}) \end{array}$$

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:

$$\begin{array}{ll} \text{Policy r3}: & \texttt{prop}(\texttt{Auth}+, \mathcal{R} \in \texttt{SRS}, \texttt{Up}) \\ \text{Policy r4}: & \texttt{prop}(\texttt{Auth}-, \mathcal{R} \in \texttt{SRS}, \texttt{Down}) \end{array}$$

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.

$$\begin{array}{lcl} \texttt{r1\_1} & : & \texttt{Auth}+(\texttt{Silver\_I}, \texttt{movie}, \texttt{play}) \\ \texttt{r1\_2} & : & \texttt{Auth}+(\texttt{Gold}, \texttt{movie}, \texttt{play}) \\ \texttt{r1\_3} & : & \texttt{Auth}+(\texttt{Platinum}, \texttt{movie}, \texttt{play}) \\ \texttt{r2\_1} & : & \texttt{Auth}-(\texttt{Silver\_I}, \texttt{movie}, \texttt{play}) \\ \texttt{r2\_2} & : & \texttt{Auth}-(\texttt{Silver\_II}, \texttt{movie}, \texttt{play}) \\ \texttt{r2\_3} & : & \texttt{Auth}-(\texttt{Bronze\_I}, \texttt{movie}, \texttt{play}) \\ \texttt{r2\_4} & : & \texttt{Auth}-(\texttt{Bronze\_II}, \texttt{movie}, \texttt{play}) \\ \texttt{r2\_5} & : & \texttt{Auth}-(\texttt{Guest}, \texttt{movie}, \texttt{play}) \end{array}$$

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 : Auth+(Bronze_II, TR, rsv_travel)
Policy r6 : Auth−(Bronze_II, TR, rsv_air)
Policy r7 : Auth−(Bronze_II, TR, 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 : rsv_travel = rsv_air ∧ 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 : Obli+(Play, Guest, fillout, questionnaire)
Policy r10 : Obli−(Sunday, Guest, login, 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 : CW(Guest, {Bank_A, Bank_B}, view_account)
Policy r12 : SoD(Bronze_I, Auction, {sell, 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 : Auth+(Guest, Bank_A, view_account)
Policy r14 : Auth+(Guest, Bank_B, 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 : Auth+(Gold, movie, play, [00:00, 24:00])
Policy r16 : Auth−(Guest, music, play, [09:00, 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.

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

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.

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

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:

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

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.

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

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.

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

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 $(A \wedge B) \rightarrow C$ is equivalent to $(\neg C \wedge B) \rightarrow \neg A$, 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.

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

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.

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

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.

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

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.

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

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.

$$\begin{array}{lll}\text{Policy r17} & : & \texttt{Obli+}(E_1, S_1, T_1, A_1) \\ \text{Policy r18} & : & \texttt{Auth−}(S_1, T_1, A_1)\end{array}$$

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

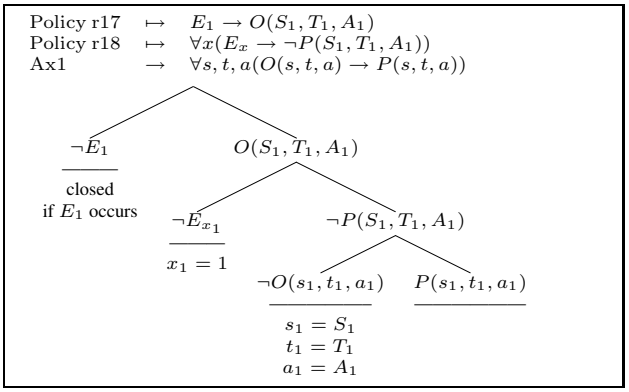

Figure 3: Modality Conflict

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.

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

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:

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

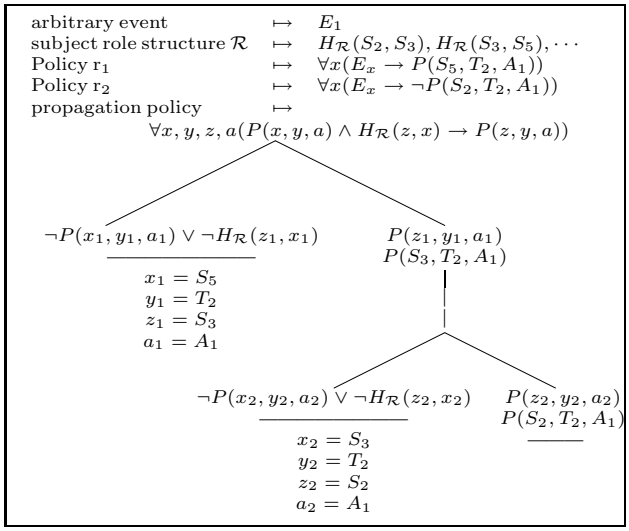

Figure 4: Conflict Caused by Propagation

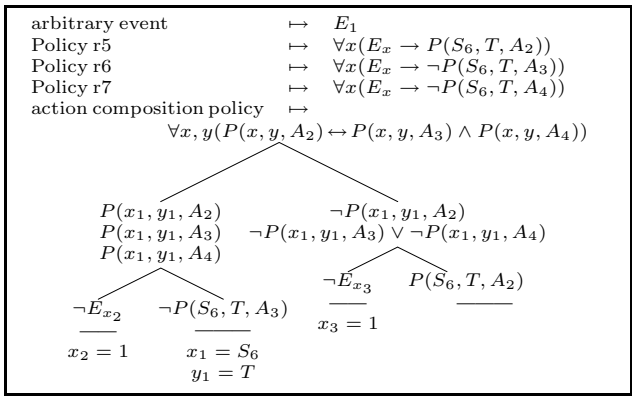

Figure 5: Conflict Caused by Action Composition

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.

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

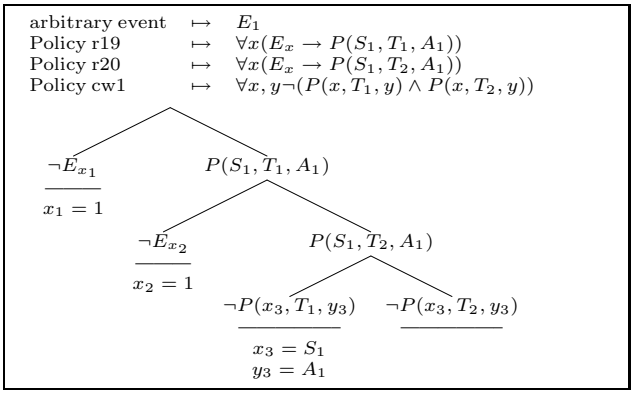

Figure 6: Conflict Caused by Chinese Wall Policy

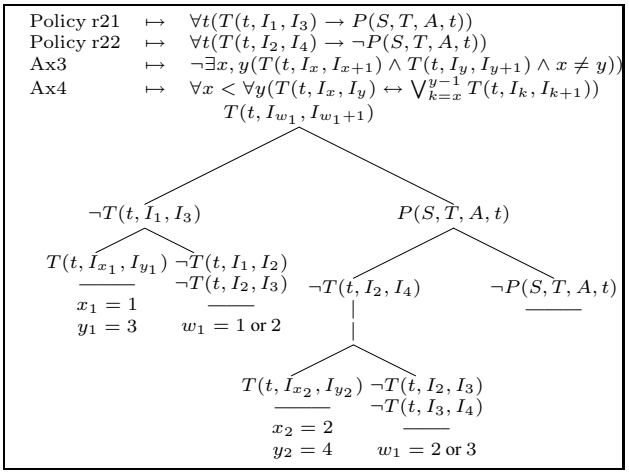

Figure 7: Conflict Caused by Time Constraint Policy

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.

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

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. leanTAP: 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.