Automatic Policy Conflict Analysis for Cross-domain Collaborations Using Semantic Temporal Logic

Zhengping Wu
Department of Computer Science and Engineering,
University of Bridgeport
221 University Avenue, Bridgeport, CT 06604, USA
zhengpiw@bridgeport.edu

Yuanyao Liu
Department of Computer Science and Engineering,
University of Bridgeport
221 University Avenue, Bridgeport, CT 06604, USA
yuaoyaol@bridgeport.edu

Abstract

Policy-based methods simplify the management of cross-domain collaborations by establishing policies to control various cross-domain activities involved in those collaborations. Administrators and users from participant domains can use policies to define control rules and restrictions, and to configure execution environments for these collaborations. To detect and resolve potential dynamic conflicts between different administrative domains, Semantic Temporal Logic (STL) is proposed and implemented in this paper to automatically analyze policy conflicts. STL incorporates relationships between different policy elements into temporal logic using a semantic format (ontology). A prototype system in the web services environment is implemented to illustrate the capability of STL and the dynamic policy analysis framework utilizing STL.

Keyword: Policy analysis, Policy-based management, Security, Trust, Web services

1. Introduction

Collaboration needs two or more participants working together towards a common goal. It involves resource sharing, task distribution and control. Policy-based management is a method that helps participants to manage their resources. Different participants may have different information management policies. Even within the same organization, different departments have different policies to protect and manage their information. In collaborations, information distribution becomes obligatory, but some information can be shared and some cannot. On the other hand, policy-based management can be viewed as an administrative approach to manage system behaviors and rules within a policy domain. The policy domain (as followed domain) is a collection of elements and services, administered in a coordinated fashion [1]. More and more policy management systems are used in network and information management systems.

Web service is a software system designed to support interoperable machine-to-machine interaction over a network [2]. Web services have become a practical data and information exchange platform among domains for collaborations. Management of these collaborations needs a set of controls to accommodate security and trust requirements for exchanged information or data. Policy-based management offers a convenient way to handle this. Web service policies can control dynamic trust behaviors across security domains. To detect and control potential conflicts in cross-domain behaviors, collaboration management systems need to be able to perform dynamic conflict analysis over web service policies.

A policy is a statement that describes what participants can do and how it can be done. In another word, a policy describes a behavior and its attributes. Police-based management systems are suitable solutions for automated online collaborations across domain boundaries, but policies become more and more complicated for cross-domain activities, which require more adaptive management approaches. Since a request or an action from a foreign domain may not have an exact match with an existing policy, several compatible policies can be applied to it. Choosing one of these policies may lead to a conflict with another related policy or violate the principle of least privilege [3]. Even a similar request or action with different parameters may lead to difficulties when applying a previously compatible policy. In general, whenever multiple policies are applied to an action or request, there is the potential for a conflict, but it is essential that multiple policies should be applied in order to cover the diversity of management needs and of different groups of users. There may be different policies related to security, privacy, or other management requirements, which are applied to different aspects reflecting different management needs. So policy conflict analysis (detection and/or resolution) becomes an essential part in the management of policy-based systems.

Policy conflicts [1] can be classified into static conflicts and dynamic conflicts. Analysis of static policy conflicts is time-independent. These conflicts can be resolved by simple logic reasoning or compiling techniques. Dynamic policy analysis is time-dependent, which means the applicability of policies and information will change over time. The major issue about dynamic policy analysis is the changing information. The information covered by policies can be identified as different types of elements. Relations between different elements are critical to conflict detection and resolution.

In most cases, administrators can define a very long policy containing different scopes and for different actions. Therefore, we introduce a general policy

Digital Object Identifier: 10.4108/ICST.COLLABORATECOM2009.8392
http://dx.doi.org/10.4108/ICST.COLLABORATECOM2009.8392

model, which decomposes policies into several policy segments. A policy segment is a small functional policy unit. In each policy segment, there is a subject set, an object set, an action set, and other related information set. This policy model (discussed in section 3) leads to a policy ontology, which will be combined with temporal logic to represent changing policies. Ontology [4] is an explicit specification of a conceptualization, which represents a set of entities and the relationships between these entities in a specific domain. Entities are abstracted from real world objects in that domain; relationships are contacts or effects between objects. The ontology describing this type of information uses a set of formal terms and operators.

The primary feature of a logic theory is its order, which defines the domain of all formulae described by the logic [5]. In classical logic, propositional logic is based on a set of elementary facts by using a set of logic operators. It indicates a Boolean value set. First order logic (FOL) [6] is an extension of propositional logic. Higher order logic [7] is an extension of first order logic. Temporal logic [5] is an extension of classical propositional logic, which represents the set of domain elementary facts by using a set of logic operators [5]. The classical logic is a clear representation of static state, value, and etc. However, time-dependent characteristics cannot be represented by classical logic. For example, in a situation such as "a user is using a computer," the term "using" is the critical part, it expresses the action is being preformed. The classical logic cannot be used to denote this situation, because the situation is time-dependent. Temporal logic can be used to represent time-dependent situations. In most policy languages, time is always an important property, which could affect other properties. Therefore, time issue should be considered when analyzing policies. In this paper, we introduce a Semantic Temporal Logic (STL), which is based on traditional temporal logic and is a combination of temporal logic and ontology to represent and reason over changing and dynamic policies from multiple domains for cross-domain activities.

2. Policy Model

2.1 General Policy Model

As we discussed before, policies describe behaviors and attributes about these behaviors. For each behavior, there is an executor or a type of executors, an object or a type of objects, and some constraints, which limit and describe the action. So in each policy, there are four major components: subject, object, action and context constraints. In most policy languages, users can define a long policy containing different actions or within different scopes. Although one policy is enforced as a whole, it can be decomposed into policy segments. A segment describes a complete behavior, which contains subject, object, action, and context constraints. The policy's subject is a set of attributes (A) identifying the action's executor, it can be denoted as s={A}; the object is a set of attributes (A) that describes an action's target, it can be denoted as o={A}; context constraints include other attributes within the policy, such as network contexts, execution environments, and etc. We distinguish this type of context information to other attributes for subjects and objects. A policy's action actually represents a temporary binding between subjects and objects, which can be denoted as A(c)=s×o. Table 1 shows the components of a policy segment in this general policy model.

A: attribute	A: the attribute that limit subjects and objects
$s = \{A\}$	s: the subject of a segment
$o = \{A\}$	o: the object of a segment
$A(c) = s\times o$	A(c): the action of a segment
Context: information	Context: the information that hide in network or environment
Segment $=\{s\times o\times A(c)\times \text{Context}\}$	segment: a part of policy

Table 1. Policy Model

Under different situations, one single policy element may play different roles in different policy segments. For example, in Web Service environment, Amazon.com provides Web Services to Alexa.com, which is built on Amazon Web Service. Alexa provides Alexa Web Services to its customers. (To distinguish these two Web Service provider, we call Amazon's Web Service "AWS", call Alexa's Web Service "ALS".) Alexa.com may play two roles in this scenario, the object in user's request, and subject in its request that is sent to AWS. There are two policies to protect these two web sites.

Policy 1:
```
<ALS >
    <allow = user>
    <Timeperiod start ="0000" Expires="2300" />
</ALS >
```
Policy 2:
```
<AWS Maintenance >
   <deny = user, Alexa>
   <allow = administrator>
   <Timeperiod start="2200" Expires="2400" />
 </AWS Maintenance >
```
In Policy 1, ALS provides services from 00:00 to 23:00. In Policy 2, AWS performs maintenance from 21:00 to 23:00. When user request both ALS from Alexa and AWS from Amazon between 22:00 and 23:00, user's requests may be denied by Alexa and Amazon, because Alexa cannot get service from Amazon, and Amazon does not provide service to user.

2.2 Security and Trust Policy Model

Security policies are designed for authentication and authorization. Subjects of security policies are Identifications, which indicate information of requesters, like user names. Objects of security policies are authentication secrets, which are verifiable information, like passwords. Actions of security policies are associations (bindings) between identifications and authentication secrets. Table 2 summarizes a general security policy model.

A: attribute	A: the attribute that limit subjects and objects
s = {A}	s:identification information
o = {A}	o: authentication security
A(c) = s×o	A(c): binding between Identification information and authentication security
Context: information	Context: the information that hide in network or environment
Segment ={ s×o×A(c)×Context}	segment: a part of policy

Table 2. General Security Policy Model

Security policies are designed for authentication and authorization. Dynamic context information is embedded in attributes associated with the identity or policy context. The association between an identity and a set of attributes (including dynamic attributes) may cause all three major types of conflicts:

• A conflict of duty arises when the same subject performs two actions on the same object.

• A conflict of interest arises when the same subject performs each of the actions on different objects.

• Different subjects perform each of the actions on a single object, and the outcome of each action is incongruent with each other.

Security policies support exchanging multiple security secrets, exchanging and verification of multiple signature formats and encryption technologies, end-to-end message content security, and co-operation over multiple security domains. The key point in security policies is to bind several authentication secrets to an identity. The dynamicity in security policies is reflected in the following situations:

a) Identity may be replaced by role information (a type of authentication secret) to act as the subject in a governance policy for a federated action;
b) The identity information is provisional;
c) The authentication secret is provisional;
d) The association between an authentication secret and an identity is provisional;
e) Authentication secrets are constrained by or hidden in network or security domain contexts.

Trust policies are designed for verifying trust relation between a requester and a responder. Subjects of trust policies are identification information; objects of trust policies are trust requirements, which requesters have to achieve. Actions of trust policies are associations between identifications and requirements. Table 4 provides the general trust policy model.

A: attribute	A: the attribute that limit subjects and objects
s = {A}	s: Identification information
o = {A}	o: Trust requirements
A(c) = s×o	A(c): association between identification information and trust requirements
Context: information	Context: the information that hide in network or environment
Segment ={ s×o×A(c)×Context}	segment: a part of policy

Table 3. General Trust Policy Model

Trust policies are defined for trust relationships between entities or domains. A general trust model requires that an incoming request proves a set of trust claims such as identity, encryption keys, permissions, capabilities, etc., which are established beforehand. The model includes three parties: a requester, a responder and an authority. Trust relationships in this model are supported by various trust claims. The relationship between requester and responder is temporally established by the authority through issuing of an authority certification to the requester, which is the proof of credentials accepted by the responder.

Similar policy conflict situations also apply to trust policies. However, there are some specific situations for trust policies within a dynamic environment:

a) The claims may be provisional;
b) The security token used to convey claims is provisional;
c) Claims are constrained by or hidden in network or security domain contexts;
d) Other limitations imposed by third party authorities or security token services.

3. Semantic Temporal Logic (STL)

3.1 Temporal Logic and Ontology

One major issue about dynamic policy analysis is that fluent information will change over time. The fluent is anything whose value is subject to change over time. [8] The dynamic policy analysis has to keep track of the change of fluent information.

It is easy to represent static states and values using classical logic. However, time-dependent situations cannot be represented by classical logic. Temporal logic also called tense logic can handle these time-dependent situations. Temporal logic is an extension of classical propositional logic, which represents a set of domain elementary facts using a set of logic operators [1]. It has been broadly used to cover all approaches to the representation of temporal information within a logical framework [9].

We can express the examples we brought in Section 3 using temporal logic: Policy 1 indicates the ALS (fluent) will be available from 00:00 (time point) to 23:00 (time point), 23 hours (time period); Policy 2 indicates AWS is not available from 22:00(time point) to 24:00(time point), 2 hour (time period). Obviously, there is a conflict in this policy set from 22:00 to 23:00. Figure 1 shows the timeline of these two policies.

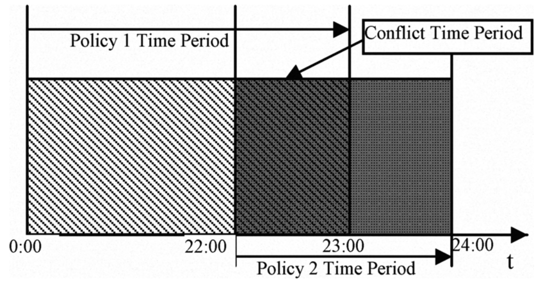

Figure 1. Timeline of Different Policies

To detect conflicts, the logic mechanism has to deal with time. Temporal logic, for example Event Calculus, can handle this situation. This situation can be presented as follow:

Policy 1: Initiates (user, t1) $\wedge$Clipped (user, t3);
Policy 2: Clipped (user, t2) $\wedge$Initiates (user, t4);
(t1=0:00; t2=22:00; t3=23:00; t4=24:00)

In many situations, relations between different elements (subjects, objects, attributes) are very complicated. A single formula cannot express all possible meanings of a behavior. As the example above, users play different roles in different policies. Although there is no problem if these policies are enforced separately in different systems, if they are enforced by Amazon and Alexa, there may be a problem when users try to request an ALS between 22:00 and 23:00. However we cannot get any information between user, Alexa and Amazon, which also plays different roles, from these formulas. To detect policy conflict, subjects and objects usually are changed under different contexts or execution environments. In the above example, the subject in Policy 2 is the object in Policy 1. (Alexa is also a user for Amazon) When we analyze these policies, this element (Alexa) would play different roles in different policies. When administrators define these two policies and enforce them, the conflict may affect the reliability of the entire system.

Ontology defines a set of representations of classes, instances, attributes, and relationships among them. In one policy domain, there are relations among policy elements (subjects and objects). When collaboration happens, ontology will be built to define elements among different policy domain, and maintain relations between these elements. Ontology is used to remove ambiguity in policy analysis processes. It is a good support for our proposed Semantic Temporal Logic (STL).

3.2 Semantic Temporal Logic (STL)

Using temporal logic to process cross-domain or multi-domain policies, semantic relationships among different elements are unavoidable, which reduce the accuracy and efficiency of policy analysis. In different domains, an element may have different definitions, which indicate the role information of this element. This element may be also involved in different relationships. Ambiguous definitions and role information will obviously affect efficiency of policy analysis. STL provides a bridge over this barrier by applying an ontology-based knowledge representation onto temporal logic itself. This representation provides not only information of individual entities from specific domains but also the relationships among these entities, which are key issues for policy analysis.

The ontology-based knowledge representation stores the information of specific policy domain, which contains each entity's information and relationships among several entities. Back to Policy 1 and Policy 2 described in the previous section, there are some attributes within these policies:

Attributes :	ALS.time={0:00 to23:00};
	ALS.available={true};
	AWS.time={22:00 to24:00};
	AWS.available={false};
	User.time={0:00 to 24:00};
	User.available= {true};

The relationship between AWS and ALS is that user can request both ALS and AWS, and there are some restrictions.

Relation: not_request (user, ALS, t);

To build the representation foundation, we use ontology to describe the model of policy domain and track relations within that policy domain. Different ontology languages provide different facilities. In OWL-Full, ALS is a class, which has some properties such as "receive request" and "available from 00:00 to 23:00". The AWS is another class, which has properties such as "receive request" and "unavailable from 22:00 to 24:00". The relation between ALS and AWS is "ALS can request to AWS", and the property on this relation is a time period (00:00 to 122:00). These two classes can be express as follows:

class(pp:ALS partial
restriction(pp:isAvailable, pp:time period))
class(pp:AWS complete
restriction(pp:not_respond
someValueFrom(intersectionOf(pp:ALS
restriction(pp:time periord)))))

This express indicates the attributes and relation we discussed before.

4. Automatic conflict analysis

4.1 Automatic conflict analysis

Automatic policy conflict analysis is then based on logic reasoning and Boolean function comparison. The later is an approach to evaluate policy similarity, which is computationally expensive and does not support on-demand requests. The complexity of policy rules and the environment fluent always affect the result of analysis. And also, the policy conflict analysis needs lots of manual work to adjust semantic relationships for accuracy, especially distinguishing subjects and objects in policy segments. Temporal logic can be used to perform automatic policy conflict analysis for dynamic policies, but the accuracy cannot be guaranteed due to the lack of capability to support semantic relationships among entities. The proposed Semantic Temporal Logic (STL) uses ontology as the knowledge supplier, which improves accuracy and reduces ambiguity in analysis results. The ontology, which records the policy model and relationships within the policy model, helps the logical reasoning mechanism identify elements and relationships between elements. For example, Policy 1 and Policy 2 are two access control policies express the relations between two elements, ALS, AWS and users. Figure 2 shows the properties and relationships between these three elements.

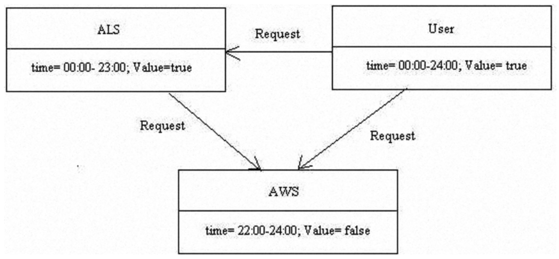
ALS
time= 00:00- 23:00; Value=true
Request
User
time= 00:00-24:00; Value= true
Request
Request
AWS
time= 22:00-24:00; Value= false

Figure 2. Properties of Policy 1, 2

Logical expressions for this example are:

HoldsAt(user,ALS,request(),t1);
Terminate (ALS,t2);
00:00 < t1 <23:00;
22:00 < t2<24:00;
Relation: not_request(user, ALS,t2) ;

In this example, elements are easy to indentify. But if we introduce another policy, elements and their relationships are so clear.

Policy 3: user can request AWS from 00:00 to 22:00;
Policy 3:

```
< user>
    <from = user>
    <Timeperiod start ="0000" Expires="2200" />
</ user>
```

The logical expressions become:

HoldsAt(user,AWS,request(),t3);
0:00<t3<22:00
Relation:request(user, AWS,t3);

User can request AWS from Amazon.com to process its task. And this task needs service both from AWS and ALS. When using temporal logic to perform reasoning among these policies, the role or identity of each element is needed. So the STL rules for this example become:

HoldsAt(user, ALS,request(), t1) $\wedge$
Terminate(user, t2) $\wedge$
HoldsAt(user,AWS, request(),t3)
$\rightarrow$*Conflict(request(user,AWS)&&use(user,ALS),t)*
Relation: not_request(user,ALS,t2);
request(user,AWS,t3);

So in a rule of STL, semantic relationships are combined with logic predicates to represent policies from multiple domains with dynamic elements. This rule in the above example uses the relationships between different entities to detect dynamic conflicts among different policies.

4.2 Web Service Policy Modeling

Web services become a practical data and information exchange platform among security domains for collaboration activities. The Web Service Policy (WS-Policy) provides a general purpose model and syntax to describe and communicate the policies of a Web service. The Web Service Security policy and Web Service Trust policy present a set of requirements that have to meet to consume a web service.

We use the general policy model illustrated in table 1 to model Web Service Policy and exemplify the model in WS-Security Policy and WS-Trust Policy. The WS-Policy specifies a set of requirements that a service requester has to meet in order to consume a web service. These requirements could be generally applicable for any web services, or they could be domain specific for certain applications. WS-Policy has a concise model to convey conditions on an interaction between entities in web service based systems from general to specific purposes.

WS-Policy is a collection of policy alternatives, and single policy alternative is a set of policy assertion (segment). The policy assertion is the smallest functional unit. It represents an individual requirement, capability, or other property of a behavior [10]. We can select subject, object, action and other information for assertions. Certain assertions in WS-Security policy include behaviors, but some assertions are just additional descriptions for other assertions. Independent assertions, which indicate complete behaviors, constitute the main body of this security policy model. There are some components in these independent assertions: subject, object, action, and context. The subject and object are the action's source and target; they both are represented by sets of attributes. The action is an association of a subject and an object. The context is a set of attributes related to a subject, object, and action. The attribute is the information that restricts the assertion. So that a policy can be expressed as follows:

A: attribute	A: the attribute within the assertion
s ={A}	s: the subject of an assertion
o={A}	o: the object of an assertion
A(c)= {s×o}	A(c): the action of an assertion
Context: information	Context: the information that hide in network or environment
Assertion={ s×o×A(c)× Context}	Assertion: a set of action and requirement
Alternative= {Assertions }	Alternative: a set of assertions
P= {Alternatives;}	P: policy, a set of alternatives

Table 4. WS-Policy Model

The WS-Security Policy and WS-Trust Policy are under the Web Service Policy framework. Within this framework, to support a policy, an entity needs to satisfy at least one of the policy alternatives contained in the policy under the current context. To satisfy a policy alternative, the entity needs to support all the policy assertions contained in that policy alternative. And to support a policy assertion, the entity needs to meet all the requirements described in that policy assertion. So we can represent our basic policy enforcement rules for WS-Policy using STL.

The Web Service Security policies define several authentication secret bindings to an identity. The dynamic environment could affect the enforceability of policies. This dynamic environment could be expressed by ontology. The ontology contains relations, attributes and other restrictions. The information within this ontology supports the temporal logic evaluations.

Security policy conflict situations discussed in section 3.2 are also suitable for WS-Security Policy. Here is an STL formula example that can be applied to detect security policy conflict in WS-Security Policy environment:

- *HoldsAt(permit(Role1(sub),obj,A1(c)),t)* $\wedge$ *HoldsAt(permit(Role2(sub),obj,A2(c)),t)* $\wedge$ *A1<>A2* $\rightarrow$ *HoldsAt(overlapConflict(conflictOfDifsubject,overlaps(permit(Role1(sub),obj,A1(c)),permit(Role2(sub),obj,A2(c)),t)* $\rightarrow$ *Trajectory(permit(Role1(sub),obj,A1(c)),t,deny(Role2(sub),obj,A2(c)),d)* $\vee$ *Trajectory(permit(Role2(sub),obj,A2(c)),t,deny(Role1(sub),obj,A1(c)),d)*
  Relation: *sub.attribute.role1* $\neq$ *sub.attribute.role2;*
  begin<t<finial;

[The situation a: the subject (sub) can have different two roles (Role1(sub) and Role2(sub)). When different roles perform different actions (A1(c) and A2(c)) toward the same object (obj) at time t, then an overlap conflict may occur. Only one action will be permitted.]

Web service trust policies describe the way two or more potential partners can use to construct and verify online trust relationships. The dynamicity in web service trust policies may cause different types of conflicts too.

WS-Trust Policy conflicts can also be detected by using trust policy conflict rules, which are built on the trust policy conflict situations we discussed in section 3.2. Here is an STL formula example that can be applied to detect security policy conflict in WS-Trust Policy environment:

- *HoldsAt(doAction(sub,attribute1,A(c)),t1)* $\wedge$ *HoldsAt(doAction (sub,attribute2,A(c)),t2))* $\wedge$ *(Clipped(claim,t)* $\wedge$ *doAction(Claim,sub,change(Authority),t))* $\wedge$ *t1<t<t2*$\rightarrow$*HoldsAt(OverlapConflict(conflictofInerest,Overlaps(doAction(sub,attribute1,A(c),t1),doAction(sub, attribute2,A(c),t2),Clipped(claim,t)),t)*$\rightarrow$*Trajectory(permit(sub,obj,doAction(sub,attribute1,A(c)),t),d)* $\wedge$ *deny(sub, obj, doAction(sub, attribute2, A(c)),t),t)* $\vee$*Trajectory(deny(sub,obj,doAction(sub,attribute1,A(c)),t),d)* $\wedge$ *permit(sub,obj,doAction(sub,attribute2,A(c)),t),t)*
  Relation: *at time t, object.attribute1* $\neq$ *object.attribute2;*
  t1<t<t2;

[The situation a: after time t1, a claim with changing attributes (attribute1 and attribute2) is checked against polices. Between time t1 and t2, the subject (sub) requests to performance a serial of actions toward different attributes and the claim is clipped in this period, then an overlap conflict (conflict of interest) may occur. Only some requests will be accepted and others will be denied.]

5. Implementation and Experiments

5.1 WS-Security Policy & WS-Trust Policy Conflict Analysis

We design a prototype system that can automatically analyze Web Service policy to detect potential conflicts within WS-Security Policy and WS-Trust Policy from multiple domains. The system is implemented in Microsoft .NET platform with a main interface for displaying analysis results and a user interface to input relationships between policy elements. We choose Event Calculus (EC) [11] as an example temporal logic tool for representing policy models and as the reasoning tool for conflict analysis. EC is a type of temporal logic to represent dynamic actions, contexts, and their effects. Web Ontology Language (OWL) is a language for defining and instantiating Web ontology [4]. OWL is stored in a XML file, which is used to store policy files. The XML format is convenient to process. We have a XML read-write module to deal with it. We use its subset OWL-Full to store and retrieve our ontology-based temporal logic STL.

This prototype is designed to analyze cross-domain WS-Policies and indicate whether there is any conflict when a cross-domain activity happens. The prototype contains two parts, one is temporal logic evaluation, which is the logic mechanism, and the other one is ontology model, which contains relations within policy model.

5.2 Experiments

The policy set used in the experiment contains 9 pairs of WS-Policy segments; include 5 pair of WS-Security Policy and 4 pairs of WS-Trust Policy. Each pair of policy segments is designed to protect the same service, but the constraints are conflict. And these policies indicate 9 types of conflicts we mentioned before. There are total 10 security policy instances. It could be 45 different combinations, if randomly chose two policies, to be analyzed. The conflicts within these policy pairs are not as much as the number of pairs. The number of conflicts of these 45 pairs is 21. And there are total 8 trust policy instances, which can be randomly combined into 28 combinations. The prototype analyzes these policy pairs based on different types of conflicts, which we discussed before. If use temporal logic to analyze these policy pairs without ontology part, the accuracy will decrease. Table 5 and 6 show a comparison of the accuracy when using STL and using general temporal logic analysis result.

	Designed Security Policy pair	Random Security Policy pair
Number of pairs	5	45
Number of conflicts	5	21
Number of conflicts reported by STL	5	21
Number of conflicts reported by TL	4	36

Table 5. Analysis Results of STL and TL

	Designed Trust Policy pair	Random Trust Policy pair
Number of pairs	4	28
Number of conflicts	4	9
Number of conflicts reported by STL	4	9
Number of conflicts reported by TL	4	12

Table 6. Analysis Results of STL and TL

There are 5 pairs of designed security policy pairs and 4 pairs of trust policy, which are designed in conflict, and 45 random security policy pairs and 28 random trust policy pairs, which are the random combinations of designed policy instances. The STL could easily analyze both designed policy pairs and random policy pairs. However, the general temporal logic mechanism reports fewer conflict alarms in conjugated policy pair test and more conflict alarms in random policy pair test. The result indicates that STL has higher accuracy by using ontology-based knowledge representation. The ontology part of STL provides a good reference to logic mechanism parts through filter unuseful relations with policy pairs.

6. Related Work

If there are positive and negative authorization or obligation policies applied on the same subjects, targets and actions, modality conflicts may arise [12]. Modality conflicts can be detected through static analysis of the policy specifications. Once all policy-driven actions of a system have been analyzed and these different types of conflicts that can arise have been identified, it is possible to define rules that can be used to recognize conflict situations in a policy set. Then these rules can be invoked during a conflict detection process prior to policy deployment and enforcement to identify potential inconsistencies. This is known as static conflict detection and takes place at specification time. Most existing work in detecting conflicts is related to modality conflicts. But sometimes application-specific conflicts cannot be detected directly from policy specifications without additional information. Conflicts may arise from the semantics of the policies as well. Thus, a metapolicy [13] concept is introduced to represent this additional information, which is a constraint or context associated with related policies. Yet there is still a possibility of conflicts even when there is not any shared subject, targets or actions between two policies. There may be implicit relationships between subjects, targets and actions when policies are applied to them. Thus a metapolicy-based approach cannot resolve all conflicts, because an administrator would have to specify metapolicies for all possible runtime conflict situations. It is almost impossible to predict all implicit relationships that may cause conflicts in the system by an administrator.

Temporal logic has been used to analyze required properties in trust policies, such as the FTPL mechanism [14], which is a first order temporal logic being used to check if a SPKI policy state satisfies a property specified in FTPL. But this properties check is for static policies and static properties, which is not sufficient for federation activities. In 2003, Event Calculus was proposed to analyze a combination of authorization and management policies [15]. But the authors do not identify the special capability of Event Calculus for transient properties in various activities. We apply Event Calculus in STL for analyzing transient properties and associations (relationships) in collaboration activities among web-service-based systems, and find its capability to detect and resolve dynamic conflicts.

Meanwhile, static and dynamic conflict detection and resolution in policy-based management for large open distributed systems are discussed in [15]. The model used in that paper does not differentiate static and dynamic policy conflicts. Although that model can be used to find some dynamic conflicts, if the specific start and finish time points of an event cannot be identified, the conflicts involving that event cannot be detected or resolved.

A general model of security policies has been discussed in [16]. Detection and reconciliation of security policy conflicts following that model are restrained by the complexity of the policy set to be reconciled. And only two-party conflict reconciliation can be tractable. Applications of the two-party conflict detection and reconciliation method to KeyNote [17] and GAA-API [18] systems are also described. But the capability of dynamic conflict detection and reconciliation is still unclear. Table 7 illustrates these comparisons.

Although the general problem of provisioning policy reconciliation is intractable [16], the ontology provides an information foundation for policy analysis, which makes the two parties policy analysis tractable and the time complexity is O(n) by using STL algorithm.

The FTPL and the model in [15] can only analyze single domain policy. It is in collaboration environment. The Ismene in[16] can analyze multiple policies, but it will be intractable if more than two parties. The STL can analyze both dynamic and static policies crossing multiple domains using ontology

	Dynamicity	Policy domain	Conflict detection and resolution	Time complexity
STL	Dynamic and Static	Multiple	Detection and resolution suggestion	O(n)
FTPL	Static	Single	Detection	N/A
Conflict detection in [15]	Static and Partial dynamicity	Single	Detection and resolution suggestion	N/A
Ismene in [16]	N/A	Multiple	Detection and resolution suggestion	NP

Table 7. Comparison of Different Policy Analysis

7. Conclusions

In this paper, we propose a Semantic Temporal Logic for automatic conflict analysis in cross-domain collaborations. Ontology-based knowledge representation provides a function that can reduce ambiguity. Through user-defined ontology, policy analysis has become more adaptable. Through the experiments on our prototype system, the improvement on capability and accuracy by Semantic Temporal Logic for automatic policy analysis is confirmed. A comparison of different policy analysis mechanisms shows that this ontology-augmented policy analysis system is more flexible, accurate, and applicable.

8. References

[1] A. Westerinen, J. Schnizlein, "RFC3198 - Terminology for Policy-Based Management," 2002. http://www.faqs.org/rfcs/rfc3198.html

[2] David Booth, Hugo Haas,"Web Services Architecture," 2004. http://www.w3.org/TR/ws-arch/

[3] Liang Chen, Jason Crampton, "Inter-domain role mapping and least privilege," *Proceedings of the 12th ACM symposium on Access control models and technologies*, pp. 157-162, 2007.

[4] B. Orgun, M. Dras, A. Nayak, G. James, "Approaches for semantic interoperability between domain ontologies," *Proceedings of the second Australasian workshop on Advances in ontologies,* Vol. 72, PP: 41-50, 2006.

[5] P. Bellini, R. Mattolini, and P. Nesi, "Temporal logics for real-time system specification," *ACM Comput. Surv.*, vol. 32, no. 1, pp. 12-42, March 2000.

[6] Nir Friedman, Joseph Y. Halpern, Daphne Koller, "First-order conditional logic for default reasoning revisited," *ACM Transactions on Computational Logic (TOCL),* Vol. 1, Issue 2, PP. 175 – 207, 2000.

[7] Andrews P.B., "An Introduction to Mathematical Logic and Type Theory: To Truth Through Proof, 2nd ed." *Kluwer Academic Publishers,* PP: 201-203, 2002.

[8] Murray Shanahan, "The Event Calculus Explained," *Lecture Notes in Computer Science,* Springer, 1999.

[9] Edward N. Zalta, "Temporal Logic", *the Stanford Encyclopedia of Philosophy*, Fall 2008 Edition. http://plato.stanford.edu/archives/fall2008/entries/logic-temporal/

[10] Siddharth Bajaj "Web Services Policy1.5 Framework (WS-Policy)," September 2007. http://specs.xmlsoap.org/ws/2004/09/policy/ws-policy0904.pdf

[11] A.K. Bandara, E.C. Lupu, and A. Russo, "Using event calculus to formalise policy specification and analysis," *Proceedings of IEEE 4th International Workshop on Policies for Distributed Systems and Networks*, pp. 26- 39, 2003.

[12] M.Sloman, "Policy Specification for Programmable Networks," *Proceedings of the First International Working Conference on Active Networks*, pp. 73 – 84, 1999.

[13] M. Sloman, E. Lupu, "Security and Management Policy Specification," *IEEE Network*, vol. 16, no. 2, pp. 10-19, March/April 2002.

[14] Arun K. Eamani, A. Prasad Sistla, "Language based policy analysis in a SPKI Trust Management System," *Journal of Computer Security*, Vol. 14, No. 4, pp. 327-357, 2006.

[15] Nicole Dunlop, Jadwiga Indulska, Kerry Raymond, "Methods for Conflict Resolution in Policy-Based Management Systems," *Proceedings of 7th International Conference on Enterprise Distributed Object Computing*, pp. 98-109, 2003.

[16] Patrick McDaniel and Atul Prakash, "Methods and limitations of security policy reconciliation," *ACM Transactions on Information and System Security,* Vol. 9, No. 3, pp. 259-291, 2006.

[17] M. Blaze, J. Feigenbaum, and Jack Lacy, "Decentralized Trust Management," *Proceedings of 1996 IEEE Symposium on Security and Privacy,* pp. 164 -173, 1996.

[18] Tatyana Ryutov, Clifford Neuman, "The Specification and Enforcement of Advanced security Policies," *Proceedings of the 2002 Conference on Policies for Distributed Systems and Networks (POLICY 2002)*, pp.128, 2002.

Digital Object Identifier: 10.4108/ICST.COLLABORATECOM2009.8392
http://dx.doi.org/10.4108/ICST.COLLABORATECOM2009.8392