A Policy Management Framework for Trusted Cross-domain Collaboration

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

Hao Wu, Yuanyao Liu
Department of Computer Science and Engineering
University of Bridgeport
Bridgeport, USA
{wuhao,yuaoyaol}@bridgeport.edu

***Abstract*—Cross-domain collaboration between enterprises needs a trusted execution environment. Policy-based management is a promising and convenient way to construct and manage such an environment. Since management of security and trust policies from collaborating domains has to handle different policies and heterogeneous policy enforcement platforms, an architectural innovation of policy management in this environment is needed. In this paper, a policy management framework is designed and implemented to provide a trusted cross-domain collaboration environment. Merits of this new framework are illustrated through an application in the web services environment.**

Keywords-policy management; cross-domain collaboration; trust; security

I. INTRODUCTION

Cross-domain collaboration between enterprises is a recent trend in complex product development, workflow federation, web/grid service orchestration, and virtual organizations [1,2,3]. Trusted and efficient collaboration over different enterprise domains leads to a much higher competitive capability of the enterprise in the distributed and networked information environment. Dynamic integration and coordination of systems across multiple domains to support collaborations require more capability for each participant system, which includes security control, trust management, adaptability for on-demand creation and self-management of dynamically evolving collaboration tasks spanning domain borders, where the participant entities (enterprises or individuals) share information, resources, and controls.

Policy-based management provides a promising way to support trusted cross-domain collaborations [4]. This requires methods and tools acquire, analyze and enforce trust-related policies from multiple domains to smooth collaboration activities and eliminate trust-related violations. Policy is "a formal statement of direction or guidance as to how an organization will carry out its mandate functions or activities, motivated by determined interests or programs [5]." A policy determines what can be done, what should be done, and how a task can be done for an administrative domain. In terms of the actual managerial contexts, a policy could be any format and any content.

For tools and methods used in online collaborations, policy plays a critical role in control and management. In workflow integration, policies from collaborating domains can regulate what information is shared with partners and what is not. In service orchestration, policies can determine whether components meet functional as well as administrative requirements from service provider domains and service consumer domains. Policy-based approaches have been used in many access control and security systems also. Users of these systems can provide security and privacy protections for their own administrative domains through policy definition.

But issues also exist when interoperation between policy systems and policy-based systems is unavoidable in cross-domain collaboration. For example, how do policies, procedures, and standards currently deployed in one administrative domain control activities that have to be performed by entities from another domain in cross-domain collaborations? Do these policies, procedures, and standards need to be modified or augmented? How can we help translate policies, procedures, and standards into different sectors, environments and contexts for partner domains?

The main problem addressed by the paper is the lack of efficient tools for cross-enterprise and cross-domain policy management. The framework presented in this paper does not aim to replace any domain- specific tools but to bridge the collaboration gap between domains. Therefore it is necessary to be able to adapt to existing systems, tools, applications, and policy management platforms. Section II gives an overview of the architecture for this policy management framework. Sections III, IV, V describe detailed structures of individual layers in this architecture. Section VI illustrates an implementation of this architecture in the web services environment for cross-domain collaborations. Section VII lists several existing systems for policy management. Section VIII compares our framework with existing systems to identify our contributions. Section IX concludes the paper with contribution highlights and future work.

II. FRAMEWORK OVERVIEW

The policy management framework presented in this paper provides an architectural innovation of policy management for trusted cross-domain collaborations. It targets security, trust and other related policies used in cross-domain collaborations. It has a three-tier reference architecture, which includes a policy input layer, a policy analysis layer, and a policy enforcement layer. Figure 1 illustrates this three-tier architecture.

Digital Object Identifier: 10.4108/ICST.COLLABORATECOM2009.8318
http://dx.doi.org/10.4108/ICST.COLLABORATECOM2009.8318

The policy input layer is provided to allow users input policies involved in cross-domain collaboration activities. To handle different policies from different domains, the policy input layer provides a very flexible interface and structure to accommodate different types of policies. It achieves the balance of effectiveness and efficiency by using a policy vocabulary definition module and a unified policy input interface. Then all policies will be stored using an internal format for analysis and enforcement in the other two layers.

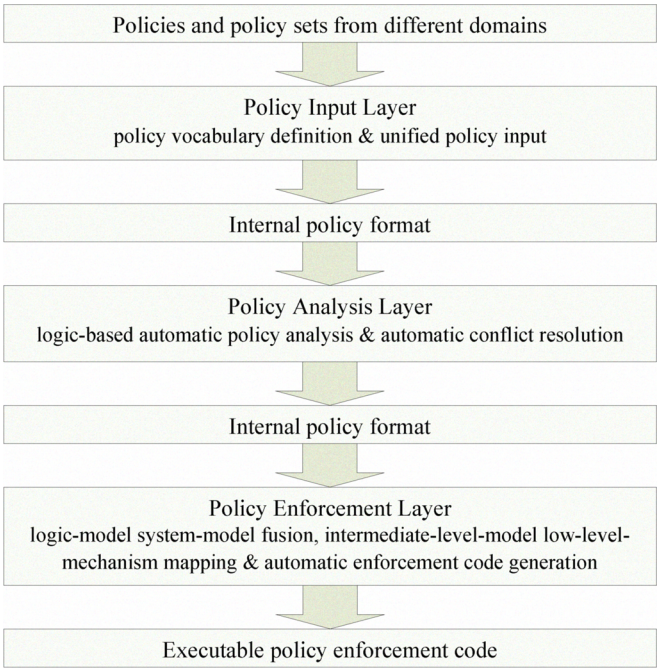


Figure 1. Three-tier architecture for proposed policy management framework

The policy analysis layer analyzes different policies or policy sets from different domains. Through the policy input layer, these policies or policy sets have been transformed into an internal format. Since cross-domain collaboration activities are dynamic. A new format of temporal logic called Semantic Temporal Logic (STL) is designed to analyze and resolve dynamic policy conflicts. Policies are analyzed automatically. But exceptions in automatic analysis are also handled. To provide enough flexibility for complex policy structures in unpredictable exceptions, users (administrators) from different domains can use a policy structure definition interface to define three key components in STL (subject, object and action) for complex policies. This type of semantic information describing subtle structures of policies is used to detect conflicts and resolve conflicts. This layer provides automatic resolution for detected conflicts and exception handling irresolvable conflicts. If an irresolvable conflict happens, an alert notification will be sent to the user (administrator). So this user (administrator) can provide his/her own resolution to override suggested resolution.

The policy enforcement layer combines the independent logical/mathematical model accepted by all participant domains in one collaboration task and the system model derived from a specific policy system used in one administrative domain to form an intermediate-level model with the unanimously accepted logical and mathematically framework and specific policy vocabulary. A semantic mapping method is used in this fusion stage. Then a bi-directional query-based method is used to map the intermediate-level model to low-level enforcement mechanisms such as network services, operating system services, and virtual machine functions so that different formatted policies from different domains can be automatically translated into executable enforcement codes for each participant domain's computing platform. Automatic code generation in this layer can reduce programmers' error-prone activities to read, understand, and translate policies into programming languages or other executable codes.

In the three layers of this architecture, different policies and policy sets are first represented by an internal format that can be easily translated into an augmented logic format for analysis and easily transformed into models for semantic-mapping-based enforcement in the policy input layer; the policy analysis layer uses STL to represent logic relationships between different elements in policies using a semantic augment; the policy enforcement layer uses semantic mapping to generate executable enforcement codes via a model-driven process. Sections III, IV, V provide detailed information about these three layers.

III. POLICY INPUT

To accommodate different types of policies used by different administrative domains, policy input layer provides flexible policy framework definition and policy rule input interfaces for users (administrators from collaboration domains). First, a user can define the framework of one policy set specifying type, vocabulary and constrains of that policy set. Then the system will automatically generate an interface for defining the policy instances (concrete policy rules) of the policy set. Details of the structure of the policy input layer are illustrated in figure 2.

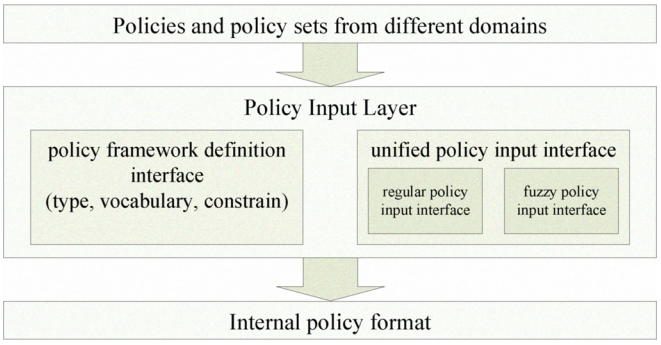


Figure 2. Detailed structure of the policy input layer

As the first step of policy input, users need to define a new policy framework. Users need to choose a name for the type of the policy framework in the policy framework definition interface. This policy framework definition interface supports not only normal policies but also fuzzy policies. Fuzzy policy is a type of policy using fuzzy terms to characterize uncertain conditions, constrains, and actions in rule definition. Normal policies support traditional authorization, obligation, and configuration rules, while fuzzy policies support uncertain

Digital Object Identifier: 10.4108/ICST.COLLABORATECOM2009.8318
http://dx.doi.org/10.4108/ICST.COLLABORATECOM2009.8318

factors in policy rules. After defining the policy framework type, users need to input a vocabulary of the policy and the basic structure of policy rules. As illustrated in figure 3, the basic structure is shown in a tree-view at the left-hand side. Users can add and delete terms in the vocabulary under each appropriate node in that tree structure at any time. When one term is selected, its attributes are shown at the right hand side of the input window. For fuzzy policies, users can choose different fuzzy operators [15] and defuzzification methods [16] for a fuzzy policy set. As shown in the "MembershipFunction" definition window within figure 3, users should also choose one membership function for each fuzzy term. All fuzzy methods and membership functions have default values and can be adjusted following users' specific needs or collaboration contexts.

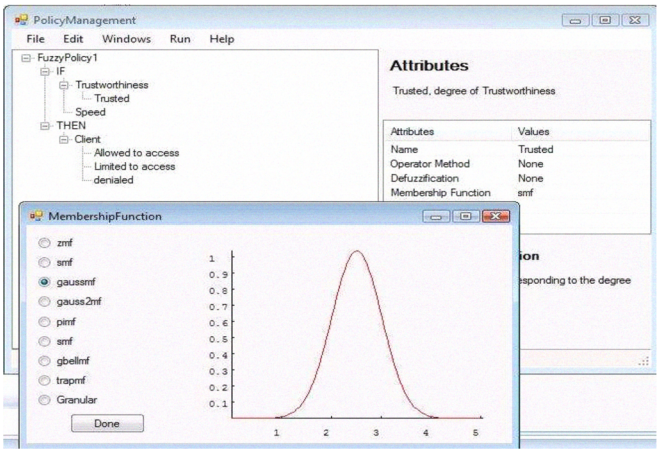


Figure 3. Policy framework definition interface

Once users have defined their policy frameworks, a unified policy input interface is automatically generated for inputting concrete policy rules following the policy framework defined in the previous step. For normal policies, every policy rule follows the same format "if subject S with attribute At and object O with attribute At satisfy context C, then action Ac is performed" no matter whether it is an authorization, obligation, or configuration rule. The action can be a permission, task, or a configuration setting. For fuzzy policy rules, the interface also allows users to change attribute terms and the degrees describing these terms. Any two of the terms can be combined using "and"/"or" fuzzy operators. Furthermore users can modify default membership functions for forming more accurate attribute terms. After the policy framework definition is determined, an internal policy format is used to store concrete policy rules. Then all concrete rules for different policies or policy sets from different administrative domains entered in this policy input layer will be translated into this format and stored in a policy repository.

IV. POLICY ANALYSIS

Applicable policies or policy sets for a collaboration activity may be defined by different domains or be defined for different purposes. Since cross-domain collaboration activities are dynamic and different domains have their own policy formats and structures, a unified policy model is introduced in policy analysis layer to detect and resolve conflicts in policies. There are four major components in this unified policy model: subject, object, action, and context. Policy modeling is the first step in this policy analysis layer, which figures out these different components from a policy set. A logic that can accommodate dynamic contexts called Semantic Temporal Logic (STL) is designed to represent and analyze policy sets. The internal policy format is transformed into this STL format.

Because of the requirements for all temporal-logic-based analysis, subjects, objects, actions, and contexts have to be identified before policy conflict analysis and resolution. Meanwhile, the existence of so many different policy languages, in a dynamic cross-domain collaboration, two or more participant domains may use different policy languages to define various policy rules, and a policy rule is a statement to be used for management or control of a system. So a policy structure definition interface is provided to allow users in one collaboration to identify four major components required in the unified policy model if the policy analysis layer cannot finish the component identification process automatically or the result is incorrect. This is the only interaction between the policy analysis layer and users before actual policy analysis. Figure 4 illustrates the detailed structure of the policy analysis layer.

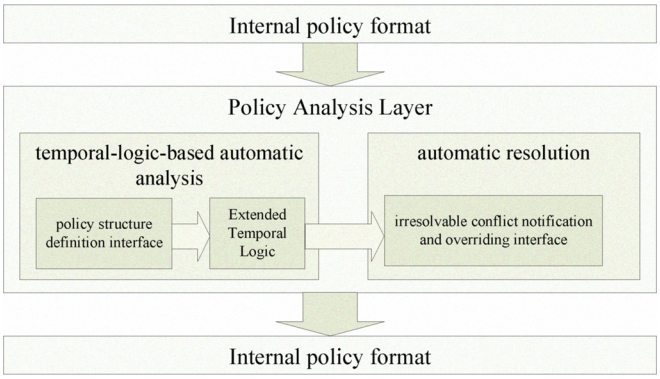


Figure 4. Detailed structure of the policy analysis layer

Existence of policy conflicts in heterogeneous policy sets defined by different participant domains for a collaboration task is not uncommon. Before collaboration happens, different domains define and enforce their own policies using their own rules and formats. These policies may play well in their own domains, but there may very possible be conflicts when these policies are applied to cross-domain activities for a dynamic collaboration task. Three types of conflicts are most common:

- A conflict of duty arises when the same subject performs both actions on the same object (e.g. a collaborative service makes a resource access request and approves it).
- A conflict of interest arises when the same subject performs "incompatible" actions on different objects. (e.g. a collaborative service lets one service delete a shared file whilst asking another service to read that file).
- Different subjects perform different actions on a single object and the outcome of each action is incongruent with the other. (e.g. spooling a job to a printer and shutting the same printer down).

Digital Object Identifier: 10.4108/ICST.COLLABORATECOM2009.8318
http://dx.doi.org/10.4108/ICST.COLLABORATECOM2009.8318

To support various types of dynamic conflicts, STL is proposed and implemented to augment temporal logic representation and reasoning with rich semantic expressions. These semantic expressions are used to help analyze and resolve different types of dynamic conflicts. STL is a temporal logic enhanced with descriptions of relationships between its elements so that the semantic meaning of each element such as subject and object can be easily identified in dynamic conflict analysis. STL incorporates timing requirements into the relationships between policy elements to perform reasoning and detect conflicting elements or policy rules. The detailed structure of STL is discussed in appendix A, and a number of example conflict detection rules are also discussed there. The final resolution is based on the result of conflict detection. The conflict resolution module decides the appropriate resolution actions based on the type of conflict identified by policy analysis module. If a conflict is detected but the conflict type cannot be identified, this policy analysis layer will notify users from corresponding domains with the specific conflict part and provide resolution suggestions. Users can follow the suggestions or modify their policies manually.

V. POLICY ENFORCEMENT

After policy conflict analysis and resolution, some relationships between elements in a policy language (vocabulary) have been established by identification of the relationships between subjects, objects, actions, and etc. Then a system model of the policy language is formed by linking these relationships. For example, if an element is a subject in one relationship and is an object in another relationship, these two relationships will be fused to the same element. So a web of relationships is formed by these fusion activities. This web of relationships is the system model of the policy language, which is represented in ontology.

In policy enforcement layer, an interface to input the formal logic or mathematical model for collaboration requirements is also provided. This model is derived from business logic or collaboration agreements for a specific cross-domain collaboration. This model is also in the format of ontology. Then a mapping is performed between the logic or mathematical model and the system model of the policy language to form an intermediate-level model. The key points in this semantic mapping are that the approvable relationships in the logic or mathematical model are being kept in the intermediate-level model and the terms in business logic or collaboration agreements are replaced or augmented by terms in the policy language. So this intermediate-level model not only includes logic relationships in collaboration but also represents these relationships using elements in a policy language. Each domain has its own intermediate-level model for enforcement. Figure 5 illustrates the detailed structure of the policy enforcement layer.

Then the last step is to automatically generate executable policy enforcement codes by mapping the intermediate-level model to low-level enforcement functions and services in operating systems or computing platforms. Total automatic code generation may not be possible, so a semi-automatic code generation method is proposed here. First, a query-based mapping between the intermediate-level model and the low-level functions or services is performed. A searchable documentation or database for all the available functions or services in the underlying computing platform is assumed, such as Java documentation for Java virtual machine and MSDN for .NET framework. Second, if more than one possible enforcement function or service is found in the underlying computing platform, a notification will be sent to the user (administrator) for a manual choice. After that, all policies defined by collaboration users will be translated into executable enforcement code automatically. Thus different policies defined by different domains' users or administrators can be enforced across computing platforms so that the entire policy management framework for cross-domain collaboration is complete.

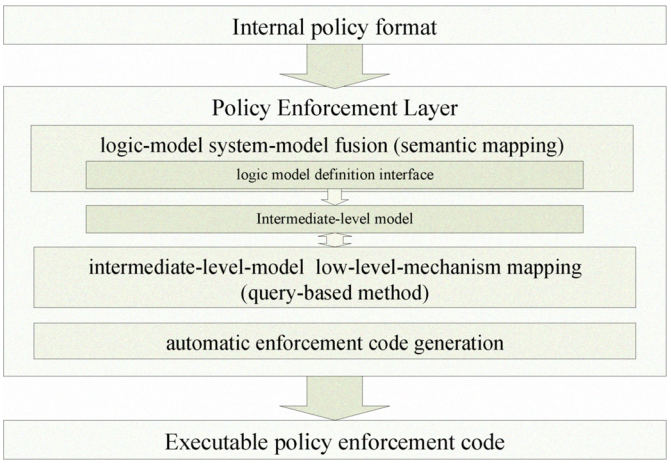


Figure 5. Detailed structure of the policy enforcement layer

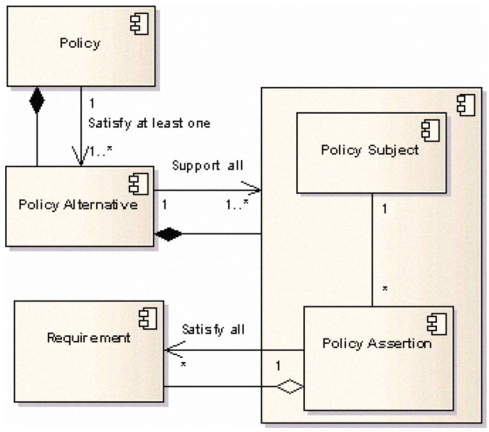
VI. IMPLEMENTATION IN A WEB SERVICES ENVIRONMENT

We implement this policy management framework for cross-domain collaboration in a web services environment. To construct a trusted collaboration environment in the web services environment, several types of policies need to be managed and enforced. We use web service security (WS-Security) and web service trust (WS-Trust) policy languages as examples to illustrate the usage of our proposed policy management framework for trusted cross-domain collaborations. In this implementation, security policies are specified by one domain (domain A), which puts more weight on the security side of its web service interactions. Trust policies are specified by another domain (domain B), which emphasizes trust relationships for its web service interactions. Domain A uses Microsoft .NET framework as their computing platform, which executes all enforcement codes. Domain B runs applications on Java virtual machine.

The administrator of domain A can input WS-Security policy with the guidance of its official specification. The vocabulary of WS-Security policies is defined in its specification, so a user can input its vocabulary and all other possible components through our policy vocabulary definition interface. All possible combinations of WS-Security components are also inputted through the policy framework definition interface. As the innovative part, if there is any

Digital Object Identifier: 10.4108/ICST.COLLABORATECOM2009.8318
http://dx.doi.org/10.4108/ICST.COLLABORATECOM2009.8318

uncertainty on a component, the administrator can attachment a membership function to describe that uncertain factor. Since both WS-Security and WS-Trust polices share the web service policy (WS-Policy) framework, the basic structures of these two languages are similar with differences on vocabularies and policy models. Policy models will be discussed in policy analysis stage. The WS-Policy framework is illustrated in figure 6.



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

Figure 6. Policy framework of WS-Policy

After input of both policies' frameworks, policy analysis starts from the internal XML structure of a WS-Security policy file. The tree-like structure (or graph-like structure with interlinked URIs) of an XML file has an internal structure of policy components defined in policy rules. But most of the time, this internal structure does not reflect full relationships (or semantic relationships) between policy components. To use temporal logic to process cross-domain or multi-domain policies, this relationship barrier cannot be avoided, which reduces the accuracy and efficiency of policy analysis. STL provides a bridge over this barrier by applying an ontology-based knowledge representation model onto temporal logic itself. This model provides not only information of individual entities from specific domains but also the relationships among these entities, which are key issues for policy analysis.

Through the policy structure definition interface of the policy analysis layer, administrators can modify the policy structure automatically constructed from the internal XML structure and augment more relationships between different policy components in the semantic format discussed in section 4. This ontology-based knowledge representation stores the information of specific policy domain, which contains each entity's information and relationships among several entities. We chose OWL-Full [17] as the representation format to store relation information. The OWL-Full is an expressive sub language from OWL ontology language. To build the representation foundation, we use ontology to describe the model of policy domain and relations within that policy domain. This ontology keeps track of the relations between elements in the policy domain. For example, there are two policies:

"The computer lab will open from 8am to 8 pm."(Policy 1-defined by facility manager)
"A student can use computers in lab from 9 am to 9 pm."(Policy 2-defined by system administrator)
In these two policies, computer is a class, which has some properties such as "in a computer lab" and "available from 8am to 8pm". The student is another class, which has properties such as "can use lab computer" and "from 8am to 9pm". The relation between computer and student is "student can use computer", and the property on this relation is a time period (8am to 9pm). These two classes can be express as follows:

class(pp:computer partial
restriction(pp:isAvailable, pp:inLab))
class(pp:student complete
restriction(pp:use
someValueFrom(intersectionOf(pp:computer
restriction(pp:inLab)))))

The first situation (Policy 1) indicates the computer (fluent) will not available from 8 pm (time point); the second situation (Policy 2) indicates students (user/fluent) can use computer till 9 pm. Obviously, there will be a conflict in this policy set for the time period between 8 pm and 9pm. To detect this conflict, the logic mechanism has to deal with time. Temporal logic can easily handle this issue. A Temporal Logic representation of the above situation is as follows:

Policy 1: Initiates(computer, t1)∧Clipped(computer, t2);
Policy 2: Initiates(computer, t1)∧Clipped(computer, t3);
(t1=8 am; t2=8 pm; t3=9 pm). (Obviously , there is a conflict between t2 and t3:Conflict(computer, t) ∧ t2<t<t3)

However, in many situations, relations between different elements (subjects, objects, attributes) are very complicated. A single formula cannot express all possible meanings of behaviors. For example, when using Temporal Logic as a reasoning tool to detect policy conflict, subjects and objects usually are changed under different contexts or execution environments. In the above example, the subject in policy 1 is the object in policy 2. When we analyze these policies, this element (computer) would play different roles in different policies. When users define these two policies and enforce them in a collaboration activity, the conflict may affect the reliability and stability of the entire system. So basic rules of STL are represented as follows using temporal logic predicates.

Digital Object Identifier: 10.4108/ICST.COLLABORATECOM2009.8318
http://dx.doi.org/10.4108/ICST.COLLABORATECOM2009.8318

HoldsAt(subject,t) ∧ HoldsAt(object,t) ∧ HoldsAt(Action(c),t) ∧ HoldsAt(relation(subject, object, Action(c), t) ∧ (t<t') → HoldsAt(subject, object, Action(c), t)

And the rule of STL to detect conflict between Policy 1 and Policy can be represented as:

HoldsAt(student,computer,use(),t) ∧ Terminate(computer,t') ∧ 8am<t<9am ∧ 9am < t'

Relation: use(student, computer) at t

The detailed rules to detect conflicts in WS-Security policies are described in appendix A also.

After conflict detection, conflicting policies will be automatically modified (splitting time constrains). If the time constrains in a context cannot be modified, the policy analysis will notify administrators to modify policies manually in order to eliminate detected conflicts. Then the policy enforcement layer will process modified policy sets or policies.

The goal of policy enforcement layer is to generate executable enforcement codes for both Microsoft .NET framework and Java virtual machine. The system model of WS-Security policy language can be constructed through the vocabulary as well as the information inputted by users at policy analysis layer. The logic model of security control in web services is clearly specified in WS-Security specifications. WS-Security policies are defined to provide confidentiality and integrity for shared or exchanged information between domains. Since WS-Security tries to bind several authentication secrets to an identity, the direct association of the identity with several related authentication secrets can serve this purpose. This binding information can be conveyed in security tokens. In the logic model of security policies, identity information, encryption methods, signature types, the type of binding, and other required attributes are essential. And these items in the ontology corresponding to the logic model can have direct mappings to the elements in the system model and its corresponding common part ontology. To provide flexibility, the attachment element in WS-Security specification can be treated as the special part ontology. So combining the mapped common part ontology and special part ontology of WS-Security, the logic model of security policies evolve into the intermediate-level model for WS-Security, which includes identity, XML encryption methods, XML signatures, security token type, attribute claims, and SOAP message attachments. Figure 7 illustrates the workflow of this intermediate-level model formation.

The intermediate model is then mapped to enforcement platforms through query-based mapping. A query engine is built between the intermediate-level model and low-level mechanisms. We illustrate this mapping process on .NET platform and on Java virtual machine. Following the architecture of .NET framework, the intermediate-level model is mapped to role-based security mechanisms first. If role-based security mechanisms cannot satisfy all the requirements, extra support is sought in the code access security mechanisms.

(1) *Role-based security mechanisms: The identity and role information of the authenticated requester is bond through a principal object, which is attached to the current web request. Then service provider can use "HttpContext.Current.User" property for retrieving principal objects. The principal objects can be mapped to security tokens in the WS-Security system model for role information. Once identity and principal objects are defined, service provider's code can use imperative or declarative security checks to determine whether a particular principal object contains a known role, has a known identity, or represents known identity acting in a role. During the security check, the CLR examines the requester's principal object to determine whether its identity and role match those represented by the "principal permission" being demanded.*

(2) *Code access security mechanisms: .NET framework can also perform code level security checks, which can be mapped to the ontology without corresponding elements in role-based security mechanisms. Four types of elements are used in .NET code access security mechanisms: Permission, Evidence, Policy (There are four policy levels: Enterprise, Machine, User, and Application domain), and Code group (A code group consists of two elements: Membership condition, Permission set). In this layer, digital signature or organization information is mapped to identity in the WS-Security system model. Other types of evidence and permissions are mapped to authentication secrets. And code groups can be used as "security tokens" in this layer to bond identity and authentication secrets.*

At the final step, the enforcement code can be automatically generated using these mapping results. Thus we finish the entire enforcement process for WS-Security policies over .NET framework. For Java virtual machine, the mapping of intermediate-level model to low-level mechanisms is similar. Java documentation is treated as a database, and every element in the intermediate-level model is queried into this "database" to find a match. If there is more than one match found, a notification will be prompted to administrators for manual justification.

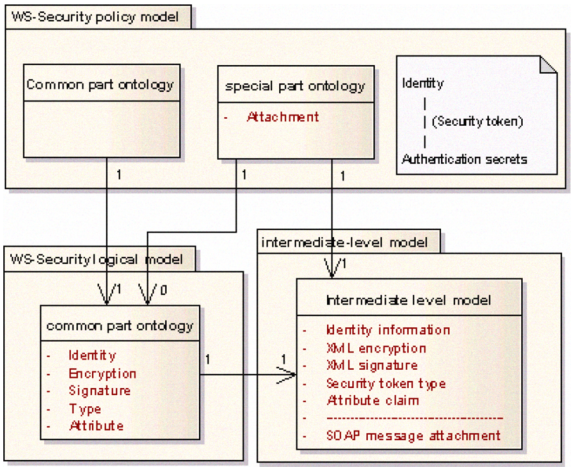


Figure 7. Workflow of intermediate-level model formation for WS-Security

The workflow of policy management for WS-Trust is almost the same as the one for WS-Security. The input of WS-Trust policy can be guided by the guidance of its official specification. Rules of conflict detection and resolution are similar to the ones for WS-Security policy language described in appendix A. The major difference is the logic-model system-model fusion stage. Figure 8 illustrates the workflow of this logic-model system-model fusion stage. After that, the same steps are taken to map intermediate-level model to low-level mechanisms on .NET platform and on Java virtual machine.

Digital Object Identifier: 10.4108/ICST.COLLABORATECOM2009.8318
http://dx.doi.org/10.4108/ICST.COLLABORATECOM2009.8318

Then enforcement code can be automatically generated using mapping results. Thus, policies defined by different administrative domains running on different computing platforms can be put together for a collaboration activity and enforced across computing platforms after conflict detection and resolution.

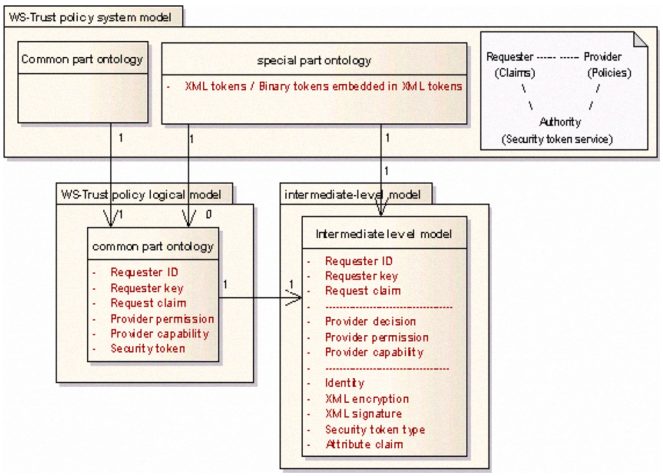


Figure 8. Workflow of logic-model system-model fusion for WS-Trust

VII. RELATED WORK

There are several software solutions on the market (e.g. PDMConnect [6], OpenPDM [7], SAP NetWeaver [8], IBM WebSPhere [9], MS Biztalk [10]) that allow the exchange of data across the boundaries of enterprises. The focus of these solutions is to exchange data between certain systems of one domain. These solutions are not able to provide an integrated view and control on the exchanged data, but allow direct mappings of data between the involved systems. Thus important meta-data or control information is lost within the exchange process for cross-domain communication since the target system is not able to trust all information needed within the collaboration.

On the other hand, several policy systems have been designed and developed for cross-domain collaborations, which try to incorporate meta-data or control information for constructing a trusted collaboration environment. CDCIE system is developed by the US Naval Research Lab (NRL), which uses hybrid architecture to accommodate multilevel security (MLS) technologies into military cross-domain collaboration environments comprised of multiple security levels (MSL) [11]. Policy management and enforcement follow a relatively simple principle, which states that provision policies at resource or data owner's domain will always overrule policies from other collaboration domains. Policies are classified into four categories for fast assessment and enforcement, which are data protection policies, resource protection policies, performance policies, and non-technical policies. If a policy from a collaboration domain is not compatible with resource or data owner's policies, this policy will be ignored. Such design may provide strict security protections, but it may also reject many possible collaboration activities controlled under policies with different formats, vocabulary, or granules. No policy input and analysis are provided in this system.

Rei is a policy language designed for pervasive computing applications, which is based on deontic concepts and includes constructs for rights, prohibitions, obligations, and dispensations across domains [12]. Rei policy language includes certain domain independent ontology and accepts domain dependent ontology. Certain domain independent ontology includes concepts for permissions, obligations, actions, speech acts, operators, and etc. Rei policy language also accepts domain dependent ontology shared by the entities in a specific system, which defines domain classes and properties associated with those classes. Rei allows users to extend the basic ontology with additional domain dependent ontology to express concepts and resources that are peculiar to a certain domain. Although Rei provides a good support for policy normalization, but it does not provide complete policy management services. Rei is only a policy language.

Ponder is a declarative, object-oriented language for specifying different types of policies, for grouping policies into roles and relationships, and then defining configurations of roles and relationships as management structures for cross-domain collaborations [13]. Ponder categorizes policies into basic policies, composite policies, and meta-policies. Basic policies include authorization policies, obligation policies, refrain policies, and delegation policies. Composite policies provide four types of policies: groups, roles, relationships, and management structures. Meta-policies are policies about which policies can coexist in the system or what are permitted attribute values for a valid policy. Ponder policies rely on the key concept of management domain. Management domain provides a means of grouping objects to entities that policies apply, and can be used to partition objects in a large system according to geographical boundary, object type, responsibility and authority, or the convenience of human managers. Ponder policy subjects and targets are defined in terms of domain scope expressions that allow combinations of domains to form composite set of objects, and to identify a single named object within individual domains. Similar to Rei, Ponder is only a policy language rather than a policy management framework. Compare to our system, it supports policy normalization and input but no policy enforcement.

KAoS is a policy service framework that has been adapted to run on a variety of cross-domain agents, web services, and traditional distributed computing platforms, and across a variety of industrial, military, and space applications [14]. In addition to services directly related to policy management, KAoS also provides the basic services for distributed computing, including message transmission and directory services. The KAoS architecture has three layers: human interface layer, policy management layer, and policy monitor and enforcement layer. Human interface layer is used to take the policy specification in the form of natural English language sentences for authorization and obligation policies. Policy management layer just uses ontology (encoded in OWL format) to represent policies. Although KAoS system uses a three-layer architecture to support policy management, the functionalities of these three layers are quite different from our framework. The pros and cons compared to our system will be

Digital Object Identifier: 10.4108/ICST.COLLABORATECOM2009.8318
http://dx.doi.org/10.4108/ICST.COLLABORATECOM2009.8318

discussed in next section from the exact functionalities offered by each layer in their three-layer architecture.

VIII. DISCUSSION

In this section, we will discuss the contributions of our policy management framework through a comparison with existing representative policy management systems in three aspects, policy input, policy reasoning, and policy deployment, which are critical for all policy management systems.

CDCIE system, Rei and Ponder all have their specific policy languages, so these systems cannot support different policy languages used by different administrative domains other than systems' own policy languages. So policy language definition is unnecessary. KAoS framework supports ontology-guided policy language input, which help users establish a policy vocabulary through a set of pre-defined ontology elements. Compared to KAoS system, our policy management framework provides a flexible yet controlled way to input policies. It allows formal vocabulary input without constrains. And it does not require natural language processing for policy rule input, which is still an immature technology.

CDCIE system, Rei and Ponder do not support policy reasoning, which makes definition and enforcement of heterogeneous policies or policy sets defined by different domains impossible. We design STL to include semantic descriptions of relationships between policy elements into temporal logic representations to accommodate dynamicity and heterogeneity of policies used in a collaboration environment. The policy analysis layer in our framework can adapt to any reasoning tools with the support from policy structure definition interface.

The most innovative part of our policy management framework is the policy enforcement layer for policy deployment, which is not provided by CDCIE system, Rei, Ponder and KAoS framework. The model-driven policy enforcement in our policy management framework leads to automatic code generation, which avoids error-prone manual coding. And the most important feature is that it can be deployed over heterogeneous platforms in a collaboration activity so that heterogeneous policies or policy sets can be enforced over different computing platforms.

IX. CONCLUSION

A new architectural design of policy management is presented in this paper for cross-domain collaboration. Through the policy management framework, which includes a policy input layer, a policy analysis layer and a policy enforcement layer, a trusted collaboration environment is achieved by policy-based management. An implementation in the web services environment for its security and trust policies together with a comparison of several existing policy management systems illustrates the novelty of the entire framework. Our further work will focus on more applications on more computing platforms.

REFERENCES

[1] Jill Gemmill, John-Paul Robinson, Tom Scavo, Purushotham Bangalore, "Cross-domain authorization for federated virtual organizations using the myVocs collaboration environment," Concurrency and Computation: Practice and Experience, Vol. 21, No. 4, 2009, pp. 509-532.

[2] Alexander Mahl, Anatoli Semenenko, Jivka Ovtcharova, "Virtual Organisation In Cross Domain Engineering," Proceedings of the 8th IFIP Working Conference on Virtual Enterprises, Portugal, September 2007, pp. 601-608.

[3] D. Green, J. Dallaire, C. Westrick, "The Integrated Information Framework: supporting interoperability between collaborative enterprise systems," Proceedings of the 2007 International Symposium on Collaborative Technologies and Systems, USA, May 2007, pp. 329-336.

[4] Adam J. Lee, Ting Yu, "Towards a dynamic and composable model of trust," Proceedings of the 14th ACM symposium on Access control models and technologies, Italy, 2009, pp. 217-226.

[5] Anne Gilliland, Nadav Rouche, Lori Lindberg, Joanne Evans, "Towards a 21st Century Metadata Infrastructure Supporting the Creation, Preservation and Use of Trustworthy Records: Developing the InterPARES 2 Metadata Schema Registry," Archival Science, Vol. 5, No. 1, 2005, pp. 43-78.

[6] Roland Eckert, Wolfgang Mansel, Günther Specht, "STEP AP233+Standard PDM=Systems Engineering PDM?," Proceedings of the 11th International Conference on Concurrent Enterprising, Germany, 2005, pp. 405-412.

[7] Jeongsam Yang, Soonhung Han, Matthias Grau, Duhwan Mun, "OpenPDM-based product data exchange among heterogeneous PDM systems in a distributed environment," International Journal of Advanced Manufacturing Technology, Vol. 40, No. 9-10, 2009, pp. 1033-1043.

[8] Loren Heilig, Steffen Karch, Oliver Bttcher, Christiane Hofmann, Roland Pfennig, "SAP NetWeaver Master Data Management," SAP PRESS, 2007.

[9] Xiaole Song, "Comparing Microsoft Speech Server 2004 and IBM WebSphere Voice Server V4.2," 2005. http://www.developer.com/voice/article.php/b3381851.html

[10] W.L. Currie, X. Wang, V. Weerakkody, "Developing Web services using the Microsoft. Net platform: technical and business challenges," Journal of Enterprise Information Management, Vol. 17, No. 5, 2004, pp. 335-350.

[11] B. Fletcher and D. Hare, "Multi-National Information Sharing: Cross-Domain Collaborative Information Environment (CDCIE) Solution," Proceedings of the 10th International Command and Control Research and Technology Symposium, USA, 2005.

[12] L. Kagal, "A Policy-Based Approach to Governing Autonomous Behavior in Distributed Environments," PhD Thesis, University of Maryland, 2004.

[13] L. Lymberopoulos, E. Lupu, M. Sloman, "Ponder Policy Implementation and Validation in a CIM and Differentiated Services Framework," Proceedings of the 2004 IFIP / IEEE Network Operations and Management Symposium, Korea, 2004, pp. 31-44.

[14] A. Uszok, J. Bradshaw, J. Lott, M. Breedy, L. Bunch, P. Feltovich, M. Johnson, H. Jung, "New developments in ontology-based policy management: Increasing the practicality and comprehensiveness of KAoS," Proceedings of the 2008 Workshop on Policies for Distributed Systems and Networks, USA, 2008, pp. 145-152.

[15] Victor Balopoulos, Anestis G. Hatzimichailidis, Basil K. Papadopoulos, "Distance and similarity measures for fuzzy operators," Information Sciences, Vol. 177, No. 11, 2007, pp. 2336-2348.

[16] Hepu Deng, Chung-Hsing Yeh, "Simulation-based evaluation of defuzzification-based approaches to fuzzy multiattribute decision making," IEEE Transactions on Systems, Man and Cybernetics (Part A: Systems and Humans), Vol. 36, No. 5, 2006, pp. 968-977.

[17] Sean Bechhofer, etc., "OWL Web Ontology Language Reference," February 2004. http://www.w3.org/TR/owl-ref/

[18] P. Bellini, R. Mattolini, and P. Nesi, "Temporal logics for real-time system specification," ACM Computing Survey, vol. 32, no. 1, 2000, pp. 12-42.

Digital Object Identifier: 10.4108/ICST.COLLABORATECOM2009.8318
http://dx.doi.org/10.4108/ICST.COLLABORATECOM2009.8318

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

[20] Matthew Horridge, Simon Jupp, "A Practical Guide To Building OWL Ontologies Using The Protege-OWL Plugin and CO-ODE Tools", 2004. http://www.co-ode.org/resources/tutorials/ProtegeOWLTutorial.pdf

APPENDIX A. SEMATIC TEMPORAL LOGIC (STL)

A.1 Structure

The major issue about dynamic policy analysis is the fluent and information will change over time. The dynamic policy analysis has to keep track of the change or fluent, and information. A new logic called Semantic Temporal Logic (STL) is proposed to handle this type of dynamic collaboration events.

Classical logic is a representation of static state, value, and etc. It is easy to represent state status 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 is built for representing the set of domain elementary facts using a set of logic operators [18]. It has been broadly used to cover all approaches to the representation of temporal information within a logical framework [19].

In temporal logic, we can express the following examples: "The computer lab will open from 8am to 8 pm."(Policy 1) and "A student can use computers in lab from 9 am to 9 pm."(Policy 2). The first situation (Policy 1) indicates the computer (fluent) will not available from 8 pm (time point); the second situation (Policy 2) indicates students (user/fluent) can use computer till 9 pm. Obviously, there will be a conflict in this policy set for the time period between 8 pm and 9pm. Figure A-1 shows the timeline of these two policies.

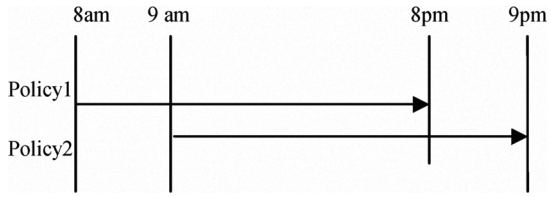


Figure A-1. Timeline of Different Policies

To detect this conflict, the logic mechanism has to deal with time. Temporal logic can be used to represent above situations as follows, which easily handles this issue.

Policy 1: Initiates(computer, t1)∧Clipped(computer, t2);
Policy 2: Initiates(computer, t1)∧Clipped(computer, t3);
(t1=8 am; t2=8 pm; t3=9 pm)
Obviously , there is a conflict between t2 and t3:
Conflict(computer, t)∧t2<t<t3;

However, in many situations, relations between different elements (subjects, objects, attributes) are very complicated. A single formula cannot express all possible meanings of behaviors. For example, when using Temporal Logic as a reasoning tool to detect policy conflict, subjects and objects usually are changed under different contexts or execution environments. In the above example, the subject in policy 1 is the object in policy 2. When we analyze these policies, this element (computer) would play different roles in different policies. When users define these two policies and enforce them in an online collaboration, the conflict may affect the reliability and stability of the entire system.

Ontology defines a set of representations of classes, instances, attributes, and relationships among them. Therefore, the ontology-based knowledge representation is a good tool for the purpose of information transferring, sharing, and analysis. It is a good support for our proposed extended temporal logic too. OWL is a language for defining and instantiating Web ontology. We use its subset OWL-Full to store and retrieve our ontology-based temporal logic STL.

We use web service security policy (WS-Security) as an example to illustrate some details of STL. The dynamicity in WS-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.

A.2 Format

Using temporal logic to process cross-domain or multi-domain policies, the semantic relationships among different components are unavoidable barriers, which reduce the accuracy and efficiency of policy analysis. STL provides a bridge over this barrier by applying an ontology-based knowledge representation model onto temporal logic itself. This model 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 the Policy 1 and Policy 2, some attributes within these policies can be represented as follows.

Attributes : *computer.time={8am to8pm};*
computer.location={lab};
computer.available={true};
student.time={8am to 9 pm};
student.authority={true};
student.action={use};

The relationship between computer and student is that student can use computer. The restriction on this relationship can be expressed as follows.

Relation: *use(students, computers, t);*

This relationship indicates student is a subject and the computer is an object. With the help of this relationship, the policy conflict analysis mechanism can handle conflict detection and resolution.

Digital Object Identifier: 10.4108/ICST.COLLABORATECOM2009.8318
http://dx.doi.org/10.4108/ICST.COLLABORATECOM2009.8318

To build the representation foundation, we use ontology to describe the model of policy domain and relations within that policy domain. This ontology keeps track of the relations between elements in the policy domain. Ontology describes the elements in a domain and also the relationships between these elements [20]. Different ontology languages provide different facilities. Using OWL-Full, the above policy example, computer is a class, which has some properties such as "in a computer lab" and "available from 8am to 8pm". The student is another class, which has properties such as "can use lab computer" and "from 8am to 9pm". The relation between computer and student is "student can use computer", and the property on this relation is a time period (8am to 9pm). These two classes can be express as follows:

class(pp:computer partial
restriction(pp:isAvailable, pp:inLab))
class(pp:student complete
restriction(pp:use
someValueFrom(intersectionOf(pp:computer
restriction(pp:inLab)))))

This express indicates the attributes and relation we discussed before.

A.3 Analysis rules

Basic rules of Semantic Temporal Logic are represented as follows using temporal logic predicates.

- *HoldsAt(subject,t) ∧ HoldsAt(object,t) ∧ HoldsAt(Action(c),t) ∧ HoldsAt(relation(subject, object, Action(c), t) ∧ (t<t') →HoldsAt(subject, object, Action(c), t)*

To detect the potential WS-Policy conflicts, the follow rules are employed, which represent five different situations described in section A.1.

- *HoldsAt(permit(Role1(sub),obj,A1(c)),t) ∧ HoldsAt(permit(Role2(sub),obj,A2(c)),t) ∧ A1<>A2 → HoldsAt(overlapConflict(conflictOfDifsubject,overlaps(permit(Role1(sub),obj,A1(c)),permit(Role2(sub),obj,A2(c)),t) → Trajectory(permit(Role1(sub),obj,A1(c)),t,deny(Role2(sub),obj,A2(c)),d) ∨ Trajectory(permit(Role2(sub),obj,A2(c)),t,deny(Role1(sub),obj,A1(c)),d)*
  Relation: sub.attribute.role1 ≠sub.attribute.role2;
  begin<t<finial;

[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.]

- *HoldsAt(doAction(sub1,obj,A1(c)),t) ∧ HoldsAt(doAction(sub2,obj,A2(c)),t) ∧ A1<>A2 → HoldsAt(dynamicConflict(conflictOfDifsubject,Overlaps(doAction(sub1,obj,A1(c)),doAction(sub2,obj,A2(c)),t),t) → Trajectory(permit(sub1,obj,A1(c)),t,deny(sub2,obj,A2(c)),d) ∨ Trajectory(permit(sub2,obj,A2(c)),t,deny(sub1,obj,A1(c)),d)*
  Relation: at time t,sub1.attribute.Id ≠sub2.attribute.Id;
  HoldsAt(sub1,obj,A1(c),t);
  HoldsAt(sub2,obj,A2(c),t);
  begin<t<finalt

[When two different subjects (sub1 and sub2) use same authentication secret perform different actions toward the same object(obj) at the same time t, another type of overlap conflict may occur. Only one action will be performed on the object.]

- *HoldsAt(permit(Sub,Obj1,A1(c),t1)) ∧ HoldsAt(permit(Sub,Obj2,A2(c),t2)) ∧ t1<=t<=t2 ∧ Clipped(t1,authorize(sub,secret,t),t2) → HoldsAt(dynamicConflict(conflictofInterest,overlaps(permit(Sub,Obj1,A1(c),t1),permit(Sub,Obj2,A2(c),t2),Clipped(t1,authorize(sub,secret,t),t2)),t),t) →Trajectory(permit(Sub,Obj1,A1(c),t1),t,deny(Sub,Obj2,A2(c),t2),d) ∨Trajectory(permit(Sub,Obj2,A2(c),t2),t,deny(Sub,Obj1,A1(c),t1),d)*
  Relation: HoldsAt(sub,obj1,A1(c),t1);
  change(obj1,obj2,t);
  t1<t<t2;

[A subject (sub) may perform different actions (A1(c) and A2(c)) toward a changing object (authentication secret) in the time period between t1 and t2. A conflict of interest may occur. Then only one action will be permitted.]

- *HoldsAt(permit(sub,obj,A1(c),t)) ∧ HoldsAt(permit(sub,obj,A2(c),t)) → HoldsAt(dynamicConflict(conflictofDuty, overlaps(permit(sub,obj,A1(c)),permit(sub, obj, A2(c)), t), t) → Trajectory(permit(sub,obj,A1(c)),t,deny(sub,obj,A2(c)),d) ∨ Trajectory(permit(sub,obj,A2(c)),t,deny(sub,obj,A1(c)),d)*
  Relation: relation(sub, obj, t) ≠relation'(sub, obj,t);

[A subject (sub) may perform different actions association between an authentication secret and an identity) to the same object (obj) at time t. An overlap conflict (conflict of duty) may occur. Then only one action will be permitted.]

- *Obj(obj1,obj2,obj3,….) ∧ HoldsAt(ValidateTrustContext(sub,obj1,A1(c)),t) ∧ HoldsAt(ValidateTrustContext(sub,obj2,A2(c)),t) ∧ … → HoldsAt(dynamicConflict(conflictOfDifobject, Overlaps(ValidateTrustContext(sub,obj1,A1(c)),ValidateTrustContext(sub,obj2,A2(c)),….),t),t) → (permit(sub,objn,An(c)) ∧ HoldsAt(ValidateTrustContext(sub,objn,An(c)),t) ∧ n=1,2,…) ∧ (m=1,2,… ∧ deny(sub,objm,Am(c)) ∧ ¬HoldsAt(ValidateTrustContext(sub,objm,Am(c)),t)))*
- *HoldsAt(ValidateTrustContext(Role1(sub),obj,A1(c)),t) ∧ HoldsAt(ValidateTrustContext(Role2(sub),obj,A2(c)),t) ∧ … → HoldsAt(dynamicConflict(conflictOfDifsubject, Overlaps(ValidateTrustContext(Role1(sub),obj,A1(c)),ValidateTrustContext(Role2(sub),obj,A2(c)),….),t),t) → (permit(Rolen(sub),obj,An(c)) ∧ HoldsAt(ValidateTrustContext(Rolen(sub),obj,An(c)),t) ∧ n=1,2,…) ∧ (m=1,2,… ∧ deny(Rolem(sub),obj,Am(c)) ∧ ¬HoldsAt(ValidateTrustContext(Rolem(sub),obj,Am(c)),t)))*
  Relation: context of subject is not compatible ;
  context of subject is not compatible.

[Different objects (authentication secrets) or different contexts may cause overlap conflicts too. Then only valid action(s) under that specific trust context will be permitted.]

Digital Object Identifier: 10.4108/ICST.COLLABORATECOM2009.8318
http://dx.doi.org/10.4108/ICST.COLLABORATECOM2009.8318