An Adaptable Security Framework for Service-based Systems

Stephen S. Yau, Yisheng Yao, Zhaoji Chen, Luping Zhu
Arizona State University, Tempe, AZ 85287-8809 USA
Email: {yau, yyao, zhaoji.chen,luping.zhu}@asu.edu

Abstract

A major advantage of service-based computing technology is the ability to enable rapid formation of large-scale distributed systems by composing massively available services to achieve the system goals, regardless of the programming languages and platforms used to develop and run these services. In these large-scale service-based systems, various capabilities are provided by different organizations and interconnected by various types of networks, including wireless (infrastructure or ad hoc) and wired networks. For these systems which often involve multiple organizations, high confidence and adaptability are of prime concern to ensure that users can use these systems anywhere, any time, through various devices, knowing that their confidentiality and privacy are well protected under various situations. In this paper, an adaptable security framework for large-scale service-based systems is presented. It includes a core ontology and a logic-based situation-aware security specification language for specifying dynamic security policies for service-based systems, an approach to policy conflict detection and resolution, and tools for generating and deploying security agents to enforce security policies. With this framework, various parties of large-scale service-based systems can rapidly specify, update, verify, and enforce security policies in service-based systems to meet their security requirements under various situations.

Keywords*: Service-based systems, security framework, adaptability, situation-awareness, security policy, security agent*

1. Introduction

Service-based systems [1-3] are becoming more popular in large-scale distributed systems because their capabilities can be independently developed, shared and managed by various providers as services, which are distributed over various types of networks, including wireless (infrastructure or ad hoc) and wired networks. A major advantage of a service-based system is the ability to enable rapid composition of massively available services to fulfill the system mission goals, regardless of the programming languages and platforms that are used to develop and execute these services.

The increasing usage of service-based systems imperatively requires effective technologies for ensuring that only authorized users can access the applications and services in open and dynamic service-based systems. The security techniques for these systems should be flexible, scalable and adaptable to the changing environments and user requirements, and provide security support for dynamic service discovery and execution. Existing security techniques [4-7] can hardly meet this challenge due to lack of a systematic way to deal with the dynamic operating environments of service-based systems.

In this paper, we will present an adaptable security framework for systematically specifying and enforcing flexible security policies in service-based systems using a logical approach to incorporating situation-awareness [8, 10] with security policies. A logical approach is used to provide unambiguous logical policy specifications to facilitate validation and verification. Although logic languages provide high level abstraction which may be difficult to be understood by users, we can easily make policy formulation close to natural languages using an ontology-based technique, which is developed for policy formulation in service-based systems. Formal verification methods can be developed for the specified policies. Situation-awareness is the capability of a system to adapt its behavior to situation changes. A situation is considered as previous device-action and/or the variation of a set of contexts relevant to the application software running on the device over a period of time. A context is an instantaneous, detectable, and relevant condition of the environment or the device, such as time, location, light-intensity,

noise-level, and available bandwidth [8-10]. By incorporating situation awareness in the process of security policy enforcement, the system can evaluate the dynamic trust relationship among various parties of the system, and hence protect related services under various situations.

2. Current state of the art

Much research has been done on distributed trust management, which is related to our adaptable security framework. *PolicyMaker* and *KeyNote* [11, 12] provide a policy enforcement framework, which uses local policies, credentials, and action strings as input to determine whether the request should be granted or denied. However, *PolicyMaker* and *KeyNote* have no support for credentials discovery, negative assertions, policy analysis, and policy composition. *REFEREE* [13] places all critical operations under policy control rather than making arbitrary decisions, which may cause certain dangerous operations to occur. Li *et al* [14] developed a role-based trust management system, called *RT,* based on the semantics of constraint Datalog. *RT* supports distributed credential discovery, which can process access requests with an incomplete set of credentials. However, *RT* only considers access control constraints, and provides no support for other aspects in a trust management system. In these systems, trust relationship is solely depended on the credential verification, and hence it is relatively static.

One important aspect in security policy enforcement is to define security policies representing the security requirements of users. Industrial standards have been proposed to specifying security policies for Web services, such as WS-Security [6], WS-SecurityPolicy [7], Security Assertion Markup Language (SAML) [15] and eXtensible Access Control Markup Language (XACML) [16]. WS-Security provides an XML framework for exchanging authentication and authorization information to secure interactions among Web services and service invokers. WS-SecurityPolicy is used to describe the security policies in terms of their characteristics and supported features, such as required encryption algorithms and privacy rules. SAML has the same purpose as WS-Security, but requires a third-party or services themselves to gather the evidence needed for policy decision. XACML is an XML framework for specifying access control policies for Web-based resources and can potentially be applied to secure service-based systems. Although, XML-based security policy specification languages provide powerful expressiveness, these languages have no support for formal analysis on security policies.

Logic-based policy specification languages have attracted more interests for their unambiguous semantics, flexible expression formats and various types of reasoning support. Default logic [17] is a very flexible policy specification language that incorporates the Bell-LaPadula model and the complexity of decision evaluation is in quadratic time. Temporal Authorization Bases (TABs) [18] labels each authorization policy with a periodic temporal expression. Although with periodic constraints TABs is still decidable, it becomes undecidable if we add more flexible temporal constructs since time instance is unbounded. Flexible Authorization Framework [19] introduces the idea of hierarchical structure and allows users to specify how to resolve policy conflicts. Description logic (DL) [20] is used to specify security policies for its clean structure and powerful reasoning support. However, to make the reasoning bounded, many restrictions are introduced, and hence security policy cannot be easily written in an intuitive way. Note that none of these approaches can address temporal constraints, situation-awareness, and hierarchical structure at the same time.

3. Overview of the security framework

Our adaptable security framework is intended for large-scale service-based systems that span multiple autonomous administrative domains without a central management authority. Multiple parties of a service-based system, such as service providers and service users, may impose their own security policies and use different authentication mechanisms, such as Kerberos or X.509 certificates. The individual security requirements of each party are reflected in domain-specific security policies. Our adaptable security framework consists of the following major components, as shown in Figure 1:

1) A core ontology for representing security policies in service-based systems.
2) A logical programming language for specifying formal security policies.
3) An algorithm for decomposing and partitioning the specified security policies into multiple policy sets.
4) Algorithms for checking the consistency, completeness and redundancy of security policies.
5) Tools and runtime support for distributed security policy enforcement, including
   - A compiler for generating security agents from the decomposed policy sets
   - A platform for deploying the security agents.

- An agent discovery protocol for securely discovering related security agents to enforce security policies on service requests.
- Enabling security techniques, such as encryption algorithms and authentication mechanisms.
- Situation-awareness agents for providing situation information needed in security policy evaluation.

In the following sections, we will discuss the detail of each major component of this framework.

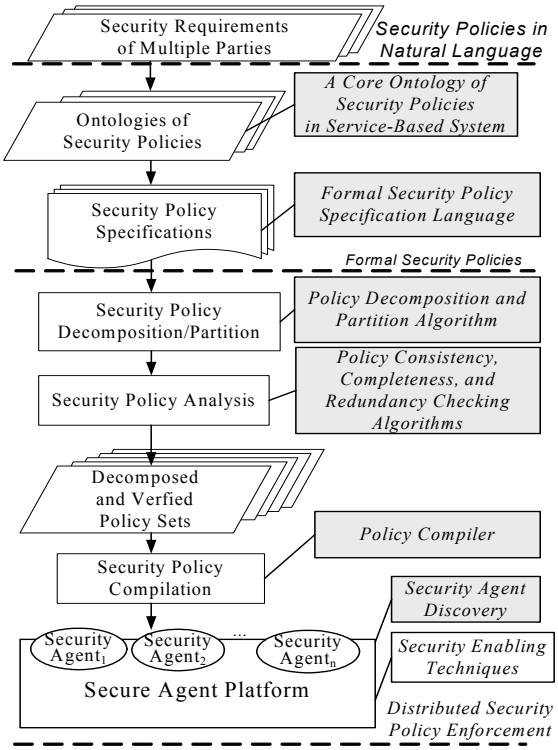

Figure 1. The adaptable security framework

4. Specifying adaptable security policies in service-based systems

In this section, we will discuss components 1) and 2) in our framework for the formulation of adaptable security policies for various security requirements in service-based systems. Security policies are rules defining security requirements. Security requirements for service-based systems have been discussed in [2, 3]. Specifically, the following security requirements need to be satisfied in service-based systems:

SR1) The client and provider of a service must be authenticated.

SR2) Confidentiality of communication among services and users should be protected.

SR3) Service access activities should be authorized. Service providers can have control over their services. They may specify security policies for protecting their services under various situations and may delegate access rights of services to third parties. The security policies must determine under any situation, the access rights from any subjects with respect to any actions on services.

Formal specifications of security policies are increasingly used for implementing flexible security solutions for adaptive large-scale distributed systems [17-22]. As service-oriented architecture (SOA) can be considered as the natural evolution of object-oriented model to specify security policies, we need to consider the structural aspects of service-based systems, such as service inter-dependencies, user and permission hierarchies. Other important features which should also be considered in the specification language include

- *Manageability*. The specified security policies should be easily understandable so that they can be easily read and updated.
- *Efficiency*. The specified security policies should be enforced efficiently.

In the following subsections, we will present a core ontology of security policies and our approach to specifying security requirements SR1-SR3.

4.1. A core ontology of security policies

To facilitate security policy specification for the security requirements in service-based systems, we have developed a core ontology of security policies. An ontology of security policies is a representation of a specific concept model that describes complex relationships between entities involved in security policies. The core ontology incorporates a global and extensible model for representing security requirements of a service-based system.

As shown in Figure 2, we define the following generic classes in the core ontology for modeling all three security requirements S1-S3:

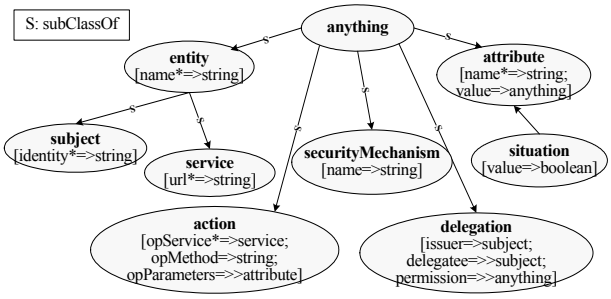

Figure 2. The security policy core ontology

- *anything* is the root type of the class hierarchy for policy specification. It does not have any real meaning, but serves as a root class for other classes.
- *entity* is a generic security entity class which is derived from *anything*.
- *attribute* is a class derived from *anything* to describe those named values.
- *subject* is a generic class for the subjects of an access request.
- *service* is a generic class for the services provided. A service may perform several different actions.
- *action* is the class for describing the requested actions which invoke a method on a service with certain parameters. The parameters also determine the object of the actions.
- *situation* is a generic class derived from *attribute* to name a situation expression used in the security policies. The value of the situation expression is provided by situation-awareness processors described in our earlier work [8, 9, 10].
- *securityMechanism* is a class for specifying which security mechanism should be used.
- *delegation* is a class for specifying a delegation policy. *issuer* is the person who previously has the right; *delegatee* is the person who will be granted the right through delegation; and *permission* describes the action which the delegatee is able to perform after the delegation.

4.2. Specifying Security Policies

Supported by the above core ontology, we can use a logic programming language, such as Description Logic [20] or F-Logic [25], to specify security policies for security requirements SR1-SR3. In this paper, to consider the object-oriented structural aspects of service-based systems, we use F-logic as our policy specification language. F-logic is originally introduced in deductive database systems and is capable of representing virtually all aspects of the object-oriented paradigm [28]. F-logic adopts a syntax similar to Prolog, where complex expressions are formed by connecting formulas through logic connectors:¬, ∧, ∨ and quantifiers:∀ and ∃. In F-logic: *a:B* means *a* is a member of *B*, and *A::B* means *A* is a subclass of *B*. `->` is used for non-inheritable scalar expression and `->>` is used for non-inheritable set-value expression. `*->` and `*->>` have the similar meaning as `->` and `->>`, except that they are used for inheritable expressions. `=>` and `=>>` are used to describe type constraints for scalar and set-value signature expressions. More complicated rules can be constructed using inference `<-`, and logic connectors and quantifiers.

Security requirements SR1 and SR2 will be considered as situation-aware secure communication policies, which define what security mechanisms, such as encryption, authentication requirement, group key management, should be used under various situations. Figure 3 shows how to model these policies. When situation expression object st_1 has the value `B`, then security mechanism `sm` should be applied. For example, the policy, "when it is exam time, the student should use `ADH-AES128-SHA` to access the exam service", can be specified as shown in Figure 4.

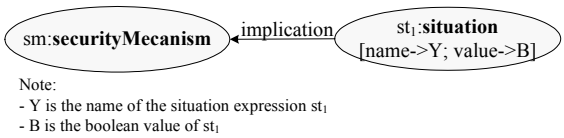

Figure 3. Modeling security policies regarding requirements S1 and S2

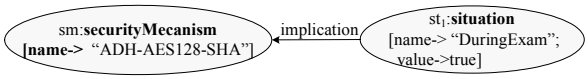

Figure 4. An example of modeling security policy

Security requirement SR3 will be specified as situation-aware access control policies. Since a general access control policy is defined as "Someone has/has-no access to perform certain action on something under certain situation". To specify this, we first model an access authorization request as shown in Figure 5.

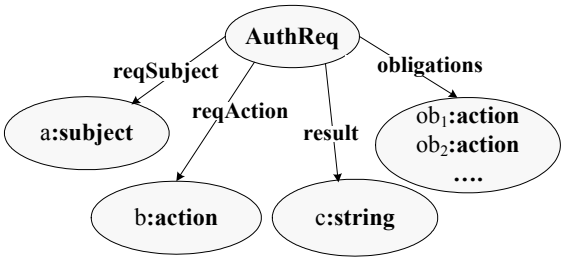

Figure 5. The ontology of access request

In the request, *reqSubject* is the entity that requests the permission; *reqAction* specifies the action being requested; the result is a string either "*Permit*" or "*Deny*", which will be determined by our adaptable security framework; and obligations are the actions that the requester needs to complete after performing the requested action. Obligations can be empty for which the requester can just do the things he requests without any other requirement.

The policies are specified as F-logic rules in the following form:

$$R[result \rightarrow "Permit";$$
$$obligations->> Z:action] \leftarrow R:AuthReq$$
$$[$$
$$reqSubject \rightarrow X:subject;$$
$$reqAction \rightarrow Y:action$$
$$] \wedge s:situation[value->true] \wedge \ldots .$$

When the first two fields match one of the predefined policies, the security framework will generate the policy decision and put it into the `result` field, with `obligation` field if any. Our approach allows specifications of both positive policies and negative policies. Negative policies can be specified in the same way with `result` set to "`Deny`". When a conflict occurs between these policies, the strategy for policy conflict resolution can be specified during the security agent generation.

Our approach for specifying security policies can be summarized in the following steps:

PS1) Describe security policies in natural languages.

PS2) Abstract the ontology of security policies based on the above core ontology as follows:
   1) Determine what kind of situations we need to consider when specifying the policies.
   2) Specify service information by creating subclasses of the `service` class.
   3) Specify the entities in the policies, and create subclasses of the `subject` class.

PS3) Instantiate these subclasses to express the facts we know about the system.

PS4) Specify the policies as F-logic rules by combining the entities, services and the corresponding situation constraints together.

In Section 6, an example will be given to illustrate how this security policy specification approach works.

5. Enforcing security policies by distributed agents

In this section, we will discuss components 3), 4), and 5) in our framework and the process of using these components to enforce adaptable security policies in large-scale service-based systems. Security policies may be specified by different users in different security domains, as shown in Figure 6. Each security domain can have one or more security policy repositories. The specified security policies of each security domain are stored in its corresponding security policy repository. To improve the enforcement performance, we use distributed agents to enforce the distributed security policies in service-based systems. *Secure Infrastructure for Networked Systems* (*SINS*) developed at the US Naval Research Laboratory [23] is integrated in our framework. SINS is based on distributed agent technology and a synchronous programming language, called *Secure Operations Language* (*SOL*) [24]. A SOL application comprises a set of SOL modules, each of which can be compiled to generate SOL agent for running on SINS. Being used in service-based systems, SINS can provide runtime support for discovering, agentifying and composing services.

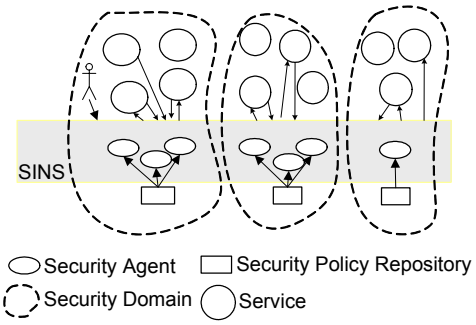

Figure 6. An architecture for security policy enforcement

The process of enforcing security using our adaptable security framework is shown in Figure 7. It includes the following three parts:

PE1) *Decompose and partition security policies and check policy consistency.* Security policies are decomposed and partitioned into different sets of security policies by analyzing the entities in the security policies. The consistency of each set of the security policies is checked based on F-logic proof theory [25]. An algorithm is currently being developed for checking security policy consistency. Once a conflict occurs between these policies, conflict resolution policies can be specified by the users, such as "permission take precedence", "denials take precedence", or "no conflicts allowed".

PE2) *Generate SOL modules of security agents.* The decomposed policies with conflict resolution approach specification can be automatically compiled to security agent specifications in the form of SOL modules, which is a security automaton representing a set of decomposed security policies.

PE3) *Generate and deploy security agent.* SOL modules of security agents are compiled to generate security agents, which will be deployed on the SINS infrastructure in the related domain.

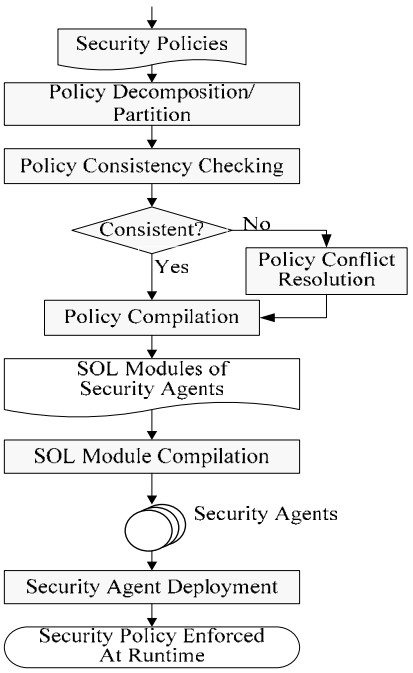

Figure 7. Enforcing security policies using our framework

During runtime, service requests are sent to the SINS infrastructure, instead of being sent to the services directly. A discovery mechanism will be provided in SINS to discover the related security agents to enforce security policies on the requests. The security agents will utilize the enabling security techniques and interact with situation-awareness agents for deciding whether the requests are authorized. If the requests are authorized, SINS will forward these requests to the services and return the service responses to the requesters. Otherwise, it will deny the requests and return error information.

6. An Example

To illustrate the process of using our adaptable security framework for service-based systems, let us consider a classroom *L* on campus used for examinations. A service-based examination system SES is needed to control the access of *L* and the examinations. The SES is built on the following three services: *gateAccess*, *logging*, *eExam*. The *gateAccess* service includes an *open* method to control the entrance of *L*. The *logging* service keeps track of all the activities in *L* and maintains a blacklist with all the students who have been suspended for having serious violations before. The *eExam* service has four methods: *view*, *submit*, *create* and *grade*. Students can take examinations by invoking the *view* method and submit answers to the examinations in electronic format by invoking the *submit* method. A teacher can create an examination and grade the electronic answers of the examination by invoking the corresponding methods.

To make this examination system secure, the following security policies need to be enforced:

a. Students can invoke *open* method if their examination will begin in the next 15 minutes. However, if a student is not supposed to take the examination, or is on the blacklist for suspension, the student's request will be denied.
b. A student in *L* during the examination time can *view* the exam questions and his/her answers.
c. A teacher can invoke all operations in *eExam* for the examination of his/her class, but can only invoke the *view* operation, not other operations in *eExam*, during the examination time,
d. A teaching assistant can invoke the *grade* method if his/her supervisor, the teacher, delegates the right to the teaching assistant.
e. All other requests not explicitly being specified will be denied.

PS1) Define the above security policies in natural languages.

PS2) Abstract the ontology from the security policies in PS1 as follows:

1) Determine the situations we need to consider when specifying the policies in PS1. For Policy a, we need to consider whether the student is supposed to take the examination and the examination will begin in the next 15 minutes. For Policies b and c, we need to consider whether the student or teacher is in *L*. Thus, we have
   - S1::situation[name→"IsRegistered"]: S1 specifies if the student is registered the corresponding course and supposed to take the examination.
   - S2::situation[name->"examIsComing"]: S2 specifies if the examination will begin in 15 minutes.
   - S3::situation[name->"InRoomL"]: S2 specifies if a student or teacher is in *L*.
2) Specify service information by creating subclasses of the *service* class. There are three services in this example, and hence we have

```
gateAccess::service[open=>boolean]          //the gateAccess service.
logging::service[get_blacklist=>list]       //the logging service.
eExam::service[ view@integer=>boolean;
                submit=>boolean;
                create@integer=>boolean;
                grade@integer=>boolean]     //the eExam service.
```

3) Specify the entities in the policies, and create corresponding subclasses of the *subject* class.

The following entities are involved in the example:

```
teacher::subject[]   //the teacher class which derives from subject.
student::subject[ isRegistered=>S1;
                  isExamComing=>S2;
                  isInRoom=>S3]   //the student class, containing the
//information about what situation constraints we need to consider in
//security policies regarding students.
tassistant::subject[supervisor=>teacher] //the teaching assistant class,
//containing information about the teacher that she/he is working with.
```

PS3) Instantiate these subclasses to express the facts we know about the system. In this example, we have the following people for this class: teacher Marissa, teaching assistant Michael, and forty students: Joey, Ross, Chandler ... Thus, we have

```
Marissa:teacher[]
Michael:tassistant[supervisor->Marissa]
Joey:student[name->"Joey"]
Ross:student[name->"Ross"]
Chandler:student[name->"Chandler"]
...
Logger:logging;
Exam:eExam;
Gate:gateAccess;
```

PS4) Specify the policies as rules by combining the entities, services and the corresponding situation constraints together.

Based on these classes, we specify Policy a as follows:

$$R[result \rightarrow "Permit"] \leftarrow R:AuthReq$$
$$[$$
$$reqSubject \rightarrow X:student;$$
$$reqAction \rightarrow Y:action[opService \rightarrow gateAccess;$$
$$opMethod \rightarrow "open";$$
$$]$$
$$] \wedge (X.isRegistered \rightarrow value) \wedge (X.isExamComing \rightarrow value)$$
$$\wedge \neg inlist(X.name, logger \rightarrow get\_blacklist).$$

and Policy d as follows:

$$R[result \rightarrow "Permit"] \leftarrow R:AuthReq$$
$$[$$
$$reqSubject \rightarrow X:tassistant;$$
$$reqAction \rightarrow Y:action[opService \rightarrow eExam;$$
$$opMethod \rightarrow "grade"]$$
$$] \wedge EXIST\ D:delegation[issuer \rightarrow T:teacher;$$
$$delegatee \rightarrow X;$$
$$permission \rightarrow A:action[opService \rightarrow eExam;$$
$$opMethod \rightarrow "grade"]$$
$$].$$

The process of enforcing these specified policies is given below:

PE1) *Decompose and partition security policies and check policy consistency.* When these policies are all specified, they can be decomposed and partitioned into different sets according to their entities. In this example, these policies can be decomposed to the following three sets of policies according to their subjects:

- Policies regarding students' access: {a, b, e};
- Policies regarding teachers' access: {c, e};
- Policies regarding TAs' access: {d, e}.

The consistency of each set of these policies can be reasoned using F-logic proof theory.

PE2) *Generate SOL modules of security agents.* After consistency checking, each set of policies together with conflict resolution approach specification will be compiled to SOL modules of security agents. A fragment of a SOL module regarding the secure gate access for students is shown below.

```
module SecureStudentGateAccess {
    type definitions
        yString = string;
        yPermission = {denied, permitted};
        yBool = bool;
    interfaces
        void Gate.open();
    internal variables
        yBool bPermissionActivated;
    monitored variables
        yString principal;
        yBool isInExam;
    controlled variables
        yPermission permission;
    Definitions
        bPermissionActivated = initially false then
        if {
              [] (isExamComing==true
                  & !inList(principal, Logger.get_blacklist)&principal==student)-> true;
              otherwise false;
         }
        permission =
        if {
              []@Gate.Open when (bPermissionActivated==true)-> permitted
              otherwise denied;
        }
}; //  end module
```

PE3) *Generate and deploy security agents.* The SOL modules of security agents can be compiled to security agents. The security agents will be deployed on SINS. At runtime, upon an access request from a principal, the SINS infrastructure will discover related security agents to ensure the security of the system.

7. Discussion

In this paper, we have presented an adaptable security framework for service-based systems. The core ontology for security policies in our adaptable security framework can facilitate security policy specification for the security requirements in service-based systems and make the specified policies easily understandable. Due to the situation-awareness feature, our framework can dynamically enforce flexible security policies. We have also introduced an F-logic based security policy specification language. F-logic includes the concept of object, class hierarchy and inheritance. Because F-logic supports a top-down, incremental structure, policies in our framework can be organized according to the subject, role or service

hierarchies in policies, which are easy to build and maintain.

Our framework has the major advantage of specifying and enforcing dynamic and flexible security policies in a dynamic service-based environment due to the support provided in the framework for specifying and enforcing adaptable security policies. Currently, we are improving the runtime support for enforcing security policies and developing the algorithms for policy consistency and redundancy checking, security agent deployment and discovery.

Acknowledgment

This work is supported by the Department of Defense/Office of Naval Research under the Multidisciplinary Research Program of the University Research Initiative, Contract No. N00014-04-1-0723. We would like to thank Ramesh Bharadwaj of US Naval Research Laboratory, Supratik Mukhopadhyay of West Virginia University, Hasan Davulcu, Dazhi Huang and Haishan Gong of Arizona State University for many helpful discussions.

References

[1] M. P Papazoglou, "Service-oriented computing: concepts, characteristics and directions," *Proc. 4th Int'l Conf. on Web Info. Systems Engineering*, 2003, pp.3-12.

[2] IBM Web Services Architecture Team, "IBM Web Services architecture overview," *http://www-106.ibm.com/developerworks/web/library/w-ovr/?dwzone=ibm*, 2000.

[3] D. Booth, H. Haas, and F McCabe, "Web Services Architecture," *http://www.w3.org/TR/2004/* , 2004.

[4] Y. Nakamur, S.Hada and R. Neyama, "Towards the integration of Web services security on enterprise environments," *Proc. Symp. on Applications and the Internet (SAINT) Workshops (2002)*, 2002, pp.166-175.

[5] M. Naedele, "Standards for XML and Web services security," *IEEE Computer*, Vol.36(4), 2003, pp.96-98.

[6] WS-Security, *http://www-106.ibm.com/developerworks/webservices/library/ws-secure/*.

[7] WS Security Policy, *http://www-106.ibm.com/developerworks/library/ws-secpol/*

[8] S. S. Yau, F. Karim, Y. Wang, B. Wang and S. K. S. Gupta, "Reconfigurable context-sensitive middleware for pervasive computing," *IEEE Pervasive Computing*, vol. 1(3), 2002, pp. 33-40.

[9] S. S. Yau, Y. Wang, D. Huang and H. P., "Situation-aware contract specification language for middleware for ubiquitous computing," *Proc. 9th IEEE Workshop on Future Trends of Distributed Computing Systems*, 2003, pp. 93-99.

[10] S. S. Yau, Y. Wang and F. Karim, "Development of situation-aware application software for ubiquitous computing environments," *Proc. 26th Int'l Computer Software and Applications Conf.*, 2002, pp. 233-238.

[11] M. Blaze, et al., "The KeyNote Trust Management System (version 2)," *RFC2704*, 1999

[12] M. Blaze, J. Feigenbaum, and M. Strauss, "Compliance Checking in the PolicyMaker Trust Management System," *Proc. 2nd Int'l Conf. of Financial Cryptography*, 1998, pp. 254-274.

[13]Y. Chu, et al., "REFEREE: Trust Management for Web Applications." *World Wide Web Jour*, vol 2(3), 1997, pp. 127-139.

[14] N. Li and J. Mitchell. "RT: A Role-Based Trust Management Framework," *Proc. 3rd DARPA Info. Survivability Conf. and Exposition*, 2003, pp. 201-212.

[15] OASIS, "Security Assertion Markup Language (SAML) Version 2.0," *http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=security*

[16] OASIS, "eXtensible Access Control Markup Language (XACML) version 2.0," *http://docs.oasis-open.org/xacml/access_control-xacml-2_0-core-spec-cd-04.pdf*

[17] R. Reiter, "A Logic for Default Reasoning," *Artificial Intelligence*, vol. 13, 1980, pp.81-132.

[18] E. Bertino, P. A. Bonatti, E. Ferrari and M. L. Sapino, "Temporal Authorization Bases: From Specification to Integration," *J. Computer Security*, vol. 8(4), 2000, pp. 309-354.

[19] S. Jajodia, Pamarati. S, M.L. Sapino, and V.S. Subrahmanian, "Flexible Supporting for Multiple Access Control Policies," *ACM Trans. on Database Systems*, vol. 26(2), 2001, pp.214-260.

[20] F. Baader, D. Calvanese, D.L. McGuinness, D. Nardi and P.F. Patel-Schneider, editors, "The Description Logic Handbook," *Cambridge University Press* 2003

[21] T. Y. C. Woo and S.S. Lam, "Designing a Distributed Authorization Service," *Proc. IEEE INFOCOM '98*, 1998. pp. 419-429.

[22] M. Sloman and E. Lupu, "Security and management policy specification," *IEEE Network*, vol. 16(2), 2002, pp. 10-19.

[23] R. Bharadwaj, "Secure Middleware for Situation-Aware Naval C^2 and combat Systems," *Proc. 9th Int'l Workshop on Future Trends of Distributed Computing System (FTDCS 2003)*, 2003, pp. 233-240.

[24] R. Bharadwaj, "SOL: A Verifiable Synchronous Language for Reactive Systems," *Proc. Synchronous Languages, Applications, and Programming (SLAP' 02). http://chacs.nrl.navy.mil/publications/CHACS/2002/2002bharadwaj-entcs.pdf*

[25] M. Kifer, G. Lausen, and J. Wu, "Logic Foundations of Object-Oriented and Frame-Based Languages," *J. ACM*, vol. 42(4), 1995. pp. 741-843.