Development and Runtime Support for Situation-Aware Security in Autonomic Computing

Stephen S. Yau, Yisheng Yao, and Min Yan

Department of Computer Science and Engineering
Arizona State University
Tempe, AZ 85287-8809, USA
{yau, yisheng.yao, min.yan}@asu.edu

Abstract. To overcome increasing complexity and dynamic nature of distributed computing system, such as ubiquitous computing systems, it is critical to have computing systems that can manage themselves according to their users' goals. Such systems are called *autonomic computing systems*. It is essential that such systems, especially those for critical applications, have the capability of self-protection from attacks under various situations without much human intervention or guidance. To achieve this goal, situation-aware security (SAS) needs to be considered in the development process. In this paper, a model-driven development framework for SAS in autonomic computing systems is presented. The runtime support for SAS is provided by a situation-aware middleware. The advantages of using the development framework and the situation-aware middleware to build autonomic computing systems with SAS are discussed and illustrated.

Keywords: Autonomic computing, security, security policy, ubiquitous computing, model-driven development framework, situation-aware middleware.

1 Introduction

Ubiquitous computing (ubicomp) provides transparent information access and computing resources for users any time and anywhere. Due to the complex, decentralized and dynamic nature of ubiquitous computing, we may encounter various issues, such as information overload, increased uncertainty and risks of using services provided by unknown or adverse parties, and high burden of system management. Therefore, autonomic computing [1], which focuses on building self-managing computing systems, is critical to avoid the risk of losing control of such ubiquitous complexity, to retain users' confidence in their trustworthiness, and to realize the benefits of ubiquitous computing and information resources. It is essential that such systems, especially those for critical applications, have the capability of situation-aware security (SAS), i.e. self-protection from attacks under various situations without much human intervention or guidance. We define a situation as a set of contexts over a period of time that is relevant to future security-related actions. A context is any instantaneous, detectable, and relevant property of the environment, the system, or users, such as time,

L.T. Yang et al. (Eds.): ATC 2006, LNCS 4158, pp. 173 – 182, 2006.

location, available bandwidth and a user's schedule [2, 3]. To achieve SAS, the system needs to be carefully engineered throughout its life cycle [1]. However, current software development approaches often neglect security in all the development levels, and suffer from lack of tools for incorporating security in development as well as developers' experience with dealing with security during development.

In this paper, we will present a model-driven development framework to provide development support for SAS in autonomic computing systems. A model-driven approach enables developers to separate models of security requirements and technical details of security implementation, and hence simplifies security management. The framework includes a model for specifying, managing and analyzing security policies in autonomic computing systems. A situation-aware middleware is used for providing runtime support for s*ituation-aware security*, i.e. the capability of being aware of situations and adapting system's security behavior based on situation changes.

2 Challenges of SAS in Autonomic Computing

Security in autonomic computing includes two major concerns: (i) *How can a set of autonomic services interact through an untrustworthy network infrastructure to achieve the security goals without human guidance*? (ii) *How can a system protect itself against potential attacks under various situations and maintaining acceptable performance*? In order to address these concerns, various security mechanisms, such as authentication, access control, secure communication, and auditing, should be implemented effectively. However, even if the necessary security mechanisms for protecting autonomic computing systems are available, it is still difficult for the systems to protect themselves against malicious tasks because developers may not be able to apply these security mechanisms properly. To provide security in autonomic computing, the following challenges need to be addressed by the system development approach and system runtime support:

C1) *Heterogeneity and Interoperability*. Devices in autonomic computing may vary from powerful servers to small embedded devices. Hence, applications of different platforms need to communicate with each other for collaboration on situation evaluation and security management across organizational boundaries.

C2) *Usability*. Usability represents a central requirement of security engineering for a secure system. For example, if the specification and management of security policies is too complex and difficult to implement or enforce, errors will inevitably occur, or system developers simply ignore parts of the security requirement.

C3) *Extensibility*. The system should be easily extended for new QoS features, such as real-time and fault-tolerance, required in various applications.

C4) *Scalability and Efficiency*. Autonomic systems may involve a large number of entities, including user, service, process, etc. Interactions among these entities are complex. The developed systems must be efficient with a large number of entities.

C5) *Decentralized and Distributed*. Information needed for making security decisions are usually distributed on multiple systems. Hence, the security solution for autonomic computing should support decentralized architecture.

C6) *Flexibility*. The system must determine the access right for any subjects with respect to any actions on services under any situation. As situation changes, the security requirements of the system are often continuously evolving.

3 Current State of Art

The Model-Driven Architecture (MDA) [4], together with UML, provides an approach to improving the quality of complex software systems by creating high-level system models and automatically generating system architectures from the models. The MDA paradigm has been specialized in the model driven security [5 -8] as a methodology for developing secure systems. Using this methodology, a developer can build the system models along with security requirements, and automatically generate a configured security enforcement infrastructure from these models.

Recently, UML has been extended for representing security requirement models [9-12]. Epstein and Sandhu [9] utilized UML as a language to represent role-based access control (RBAC) models. Shin, et al [10] presented an alternative technique to utilize UML to describe RBAC models. Some UML extensions [11-12] accommodate security requirements in software requirement modeling and development. However, existing research on this aspect does not support situation aware security.

On security requirement specification, industrial standards, such as Security Assertion Markup Language (SAML) [13], eXtensible Access Control Markup Language (XACML) [14], WS-Security [15] and WS-SecurityPolicy [16] have been established. These XML-based security policy specification languages provide fine granularity of security policy specification. These languages need additional formal semantics for formal analysis on security policies, such as the semantics for WS-SecurityPolicy [17]. Logic-based security policy specification languages have attracted much attention for their unambiguous semantics, flexible expression formats and various types of reasoning support [18-20]. However, to address the above challenges, formal approaches to policy composition and transformation are needed for these logic-based security policy specification languages.

4 Development Support for SAS in Autonomic Computing

4.1 Our Development Framework

Our development framework is based on the model-driven architecture (MDA) perspective [4]. The system requirements are first modeled using Platform-Independent Models (PIM), such as PIM-Business and PIM-Security Policy. A PIM exhibits a specific degree of platform independence, and hence it is suitable for use with a number of different platforms of similar types. The PIM is then transformed into Platform Specific Models (PSM) in order to specify how the system can use a specific type of platform.

As shown in Figure 1, the functional requirements of the system are modeled and transformed to PSM following the standard model-driven development process. Similarly, using a security enhanced UML (seUML), the security requirements are modeled as a PIM-Security Policy, which is then mapped to PSM- executable security policies in a process calculus, such as the AS^3 calculus [21]. Similar to SecureUML [11], seUML is an improved UML. But, seUML has facilities to model situation constraints based on our situation-aware access control (SA-AC) model [22]. As shown in Figure 2, SA-AC extends the basic RBAC model by including the constraints in user-role and role-permission assignments as situations. The system situation information can represent the security condition of the system. With situation constraints in security policies, security decisions can be self-adapted to current system situations. An example of using seUML will be presented in Section 6.

Various decomposition rules and transformation rules are needed in the mapping process. The *decomposition rules* represent the domain knowledge on how to efficiently enforce security policies in the system. The *transformation rules* represent the domain knowledge on how to transform high-level security policies to executable security policies. High-level security policies specify the security goals without detail information on underlying system implementation. For example, "*the communication among collaborators should be protected*" is a high-level security policy, whereas "*the communication among the applications should be encrypted using AES-128*" is an executable security policy.

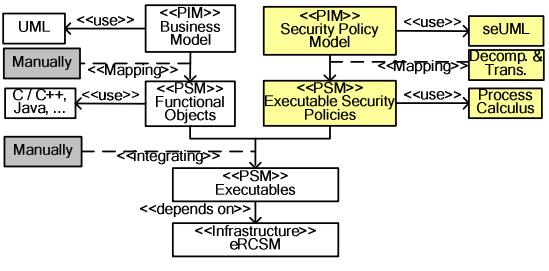

Fig. 1. Our development framework for SAS

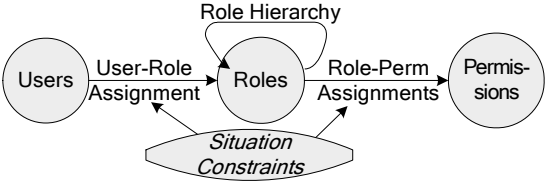

Fig. 2. An overview of our SA-AC model

Functional objects and executable security policies are integrated in PSMs, which can be run on specific computing platforms. We have extended our previous situation-aware middleware in ubicomp [2, 3, 22], to provide runtime support for SAS. This extension will be presented in Section 5.

4.2 Challenges Addressed by Our Development Framework

Our development framework addresses the challenges C1-C3 of SAS in autonomic computing discussed in Section 2 as follows:

- To address C1, our development framework provides support for rapid MDD of SAS in autonomic computing systems. The computing system is first depicted in PIM, which can easily be transformed to PSM executables on different platforms.

- To address C2, our framework automates many tasks in security management by encapsulating the complexity of security policy management and situation-awareness (SAW) processing in the middleware platform. The seUML provides a powerful tool for security administrators to specify high-level security policies without the need of knowing how it is implemented. Therefore, the security administrators of computing systems can concentrate on setting the security policy, and the framework will figure the implementation details of specified security policies by model transformation.
- To address C3, our framework can be easily extended to include new QoS properties based on the extensible MDA architecture and the extensible seUML.

5 Runtime Support for SAS in Autonomic Computing

5.1 Major Components for Providing Runtime Support

Our previous RCSM [2, 3] has a CORBA compliant architecture, providing the support of situation-awareness, distributed object invocation and situation-aware communication. We have extended our previous RCSM to eRCSM to support SAS in autonomic computing systems. The architecture of our eRCSM is shown in Figure 3 and consists of the following four major components:

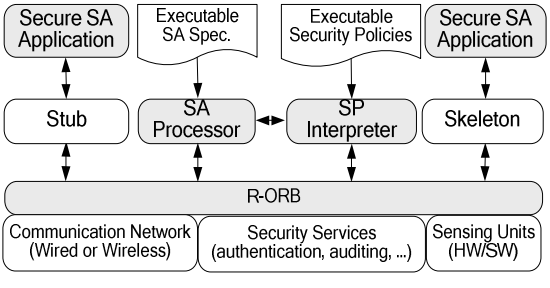

Fig. 3. The architecture of our eRCSM for providing SAS runtime support

1) *Secure SA Application* (sSAA), which is developed by modeling security and functional requirements in PIMs, transforming PIMs into PSM executables, as shown in Figure 1. In this development process, sSAA skeletons/stubs, as well as the corresponding executable security policies, are also generated.
2) *Security Policy Interpreter (SP Interpreter)*, which makes security decisions on an object invocation request based on the enforceable security policies, current security context and system situation.
3) *SA Processor*, which manages SA requirement specification, maintains the context and action history, detects situation based on context and action history, and updates the security policy interpreter on the current system situation.
4) *RCSM Object Request Broker (R-ORB)*, which provides interfaces for context discovery, collection and propagation. It also discovers and invokes the appropriate object implementation for the requests from sSAA's. For security, upon receiving an object invocation request, R-ORB enforces the security decisions made by the SP Interpreter.

5.2 The Execution Model of Our eRCSM for Providing SAS

The execution model of our eRCSM for providing SAS includes SAW processing model and security decision evaluation and enforcement model.

A) Situation-awareness processing

The flowchart for SAW processing is shown in Figure 4, and includes the following steps:

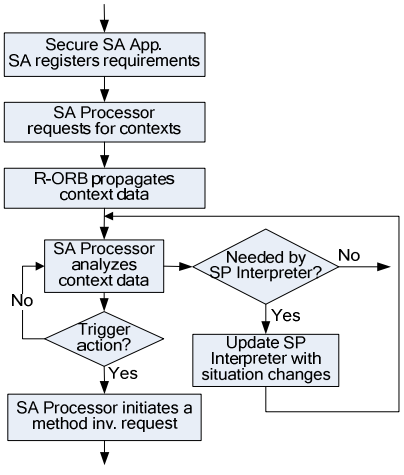

Fig. 4. The flowchart for SAW processing

1) When an sSAA starts on eRCSM, it registers SAW requirements with SA Processor.
2) SA Processor sends requests for the contexts needed for processing the registered SAW requirements.
3) The hardware/software sensing units collect the necessary contexts, including some security-related data.
4) R-ORB propagates the collected context data to SA Processors. R-ORB also provides transparent support for communications among various components.
5) SA Processor analyzes the collected context data according to our situation-awareness model [24].
6) When certain situation change is detected, if this situation is needed for security decision evaluation, SA Processor updates SP Interpreter with the situation change. If the situation triggers certain action, SA Processor will initiate action requests to related sSAA.

B) Security enforcement

The flowchart for security decision evaluation and enforcement is shown in Figure 5 and includes the following steps:

1) An sSAA or an SA Processor initiates a method invocation.
2) R-ORB locates a remote sSAA.
3) R-ORB receives the request and consults the SP Interpreter for checking security.
4) The SP Interpreter evaluates the request initiated by the sSAA or SA Processor. Decisions on whether to trust the request are made according

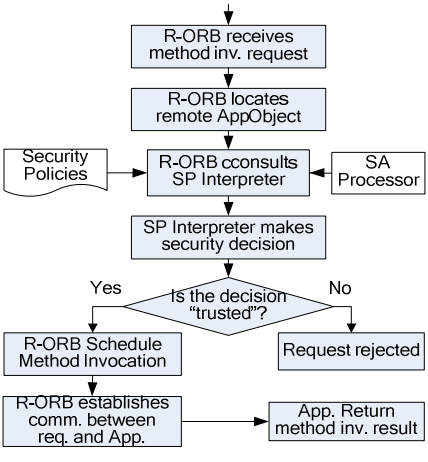

Fig. 5. The flowchart for security decision evaluation and enforcement

to executable security policies and current system situations. Various underlying security services may be invoked through a standard interface of R-ORB for decision evaluation.

5) If the security decision is 'trusted', R-ORB schedules the method invocation.
6) R-ORB establishes a channel between the requesting sSAA and the remote sSAA. After processing the request, the remote sSAA returns the output data back to the requesting sSAA.

5.3 Challenges Addressed in eRCSM

Our eRCSM can address challenges C4-C6 of SAS in autonomic computing discussed in Section 2 as follows:

- To address C4, a number of SA Processors and SP Interpreters will be deployed. Each responsible for processing different SAW requirements and security policies. Moreover, the publication/subscription style of situation updates between SA Processors and SP Interpreters reduces the overhead of situation constraint computation.
- To address C5, SA Processors and SP Interpreters can be deployed over the network for managing the security policies and SAW requirements.
- To address C6, situation awareness is incorporated into security decision evaluation. SAS security policies can be specified, based on which security decision on a request may be changed when the situation changes. Moreover, new emerging security mechanisms can be plugged in the system as underlying security services. When security requirements change, new security policies can be specified and transformed to executable security policies, and activated by SP Interpreters immediately.

6 An Example

Let us consider a situation-aware net meeting system (SANM) with the following functionalities for users to easily have a net meeting: A meeting organizer can organize a net meeting by inputting desired meeting date and time duration, indicating attendees and specifying security policies. To arrange the meeting time, the system queries the calendar service on the computing devices of the expected attendees, and gets approval from them. The system will automatically distribute meeting notice to all expected attendees. An attendee's device of SANM can join the meeting session, when the specified meeting situations are detected. Documents will be delivered to related attendees during (or before) the meeting, as needed.

By implementing an sSAA object `oMeeting` with a set of situations and roles, SANM can self-manage the security of the system. Meeting `attendees` can `join` the meeting session `M` and send secure message by calling either method `msg256` (with strong encryption) or method `msg64` (with fast encryption) of `oMeeting`. Likewise, an organizer can securely distribute documents using either `doc256` or `doc64`. The `kickout` method is triggered to remove an untrusted attendee from `M`.

To provide security in this system, the administrator of SANM may set the following high level security policies for M:

a) To protect privacy, attendees should not be in public places, such as shopping mall.
b) Confidentiality of the messages and documents sent over the network must be ensured. To keep good performance of the meeting, the protection can be flexible. If the system detects that the response of a user's mobile device is too slow when using 256-bit encryption, the system can establish a new security context to use 64-bit fast encryption.
c) When a device is reported to be stolen, the device cannot continue the meeting.

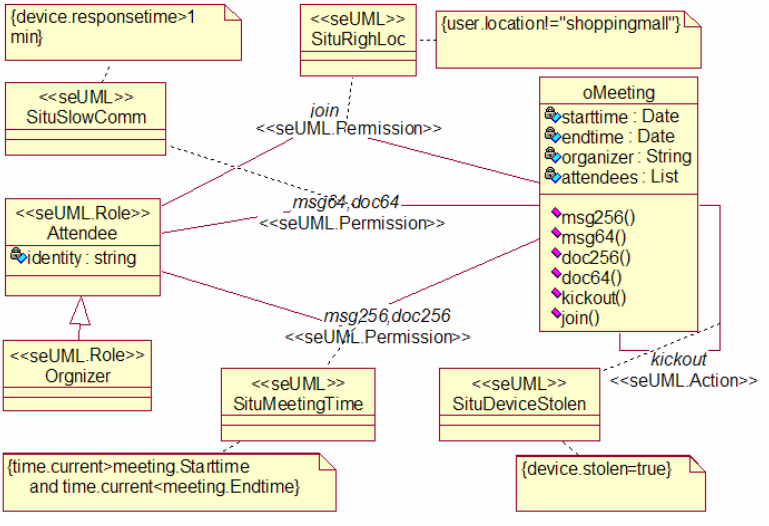

Fig. 6. seUML specification for Policies a)-c)

These policies show that the system should self-reconfigured for security purpose when the situation changes. Using our framework, this can be achieved as follows:

A) Security policy specification using seUML

Following our SA-AC model, system administrators can specify the above security policies in seUML, as partially shown in Figure 6. Permissions, such as "`join`", "`msg256,doc256`", "`msg64,doc64`", and "`kickout`" are specified as the associations among roles and objects. For Policy a), `SituRightLoc` (*Attendee is located in an appropriate meeting environment*) is needed. `SituMeetingTime` (*The meeting is ongoing*) is needed for Policy b). For Policy c), we need `SituDeviceStolen` (*Device of an attendee is reported stolen*) and `SituSlowComm` (*Device response time is more than 1 minute when processing 256-bit encryption).*

B) Security policy decomposition, transformation and enforcement

B1) *Decomposition.* Security policies in seUML can be exported to XML-based specifications, and can be further decomposed to different sets according to their entities. In this example, if the policy decomposition rule is "*attendees' access and*

system actions should be controlled by two separate sets of security policies", these policies can be decomposed to multiple policy sets as follows:

- *Policy Set A)*:

```
a)An attendee can enter join the meeting if
  SituRightLoc is true.
b-1)When SituMeetingTime is true, attendee can send message
    or documents using msg256 or doc256.
b-2)When SituSlowComm is true, attendee can send message or
    documents using msg64 or doc64.
```

- *Policy Set B):*

```
c)When SituStolen is true, oMeeting can remove the related
  attendee using kickout.
```

B2) *Transformation*. Each set of security policies is transformed to platform-dependent executable security policies. A fragment of executable security policies in AS^3 calculus [21] for policy set A) is shown below.

```
Fix G = (principal, permission) || (SituSituRightLoc)                    //process G implemets Policy set A)
                        || (SituMeetingTime) || (SituSlowComm) .    //get security request and situations
     if ( (hasAuthenticate(principal) == 'true') and
         ((permission=='join'  and SituRightLoc == 'true') or
          (permission=='msg256,doc256' and SituMeetingTime == 'true') or
          (permission=='msg64,doc64' and SituSlowComm == 'true'))    //check permissions and situations
       ) then <"trusted">.G                                                    //output security decisions
     else <"denied">.G
```

B3) *Enforcement*. The executable security policies will be deployed on eRCSM. At runtime, upon an access request from a principal, the SP Interpreter on eRCSM will execute related executable security policies. Security decisions are made according to executable security policies and current situation values

7 Conclusion and Future Work

In this paper, we have presented a model driven development framework and a middleware-based runtime support for SAS in autonomic computing systems. We have discussed how our framework and middleware-based runtime support address the challenges to support SAS in autonomic computing systems. As illustrated by the example, the above support can greatly simplify the development effort and facilitate the autonomic execution and reconfiguration of security in dynamic and complex systems, such as ubiquitous computing systems. Future work includes analysis of the expressiveness of seUML, and development and runtime support for satisfying more QoS requirements, such as real-time, fault-tolerance, survivability.

Acknowledgment

This work was supported by National Science Foundation under grant number CNS-0524736. We would like to thank Dazhi Huang for many helpful discussions.

References

1. J. O. Kephart and D. M. Chess, "The vision of autonomic computing," *IEEE Computer*, vol. 36, no.1, 2003, pp.41-50.
2. S. S. Yau, Y. Wang and F. Karim, "Development of Situation-Aware Application Software for Ubiquitous Computing Environments", *Proc. 26th IEEE Int'l Computer Software and Applications Conf*, 2002, pp. 233-238.
3. S. S. Yau, *at el*, "Reconfigurable Context-Sensitive Middleware for Pervasive Computing," *IEEE Pervasive Computing*, Vol. 1, No. 3, 2002, pp. 33-40.
4. OMG, "MDA Guide Version 1.01", URL: http://www.omg.org/, accessed on 03/18/2006.
5. C.C. Burt, *et al*, "Model driven security: unification of authorization models for fine-grain access control," *Proc. 7th IEEE Int'l Enterprise Distributed Object Computing Conf.*, 2003. pp. 159- 171.
6. D. Basin, J. r. Doser, and T. Lodderstedt, "Model driven security for process-oriented systems," *Proc. 8th ACM Symp. Access Control Models and Tech.*, 2003, pp. 100-109.
7. J. Jürjens., "Model-Based Security Engineering with UML," *Lecture Notes in Computer Science*, Volume 3655, 2005, pp. 42 - 77.
8. Y. Nakamura, *et al*, "Model-Driven Security Based on a Web Services Security Architecture," *Proc. 2005 IEEE Int'l Conf. on Services Computing*, 2005, pp.7-15.
9. P. Epstein and R. Sandhu, "Towards a UML based approach to role engineering," *Proc. 4th ACM Workshop on Role-Based Access Control*, 1999, pp. 135-143.
10. M. E. Shin and G.-J. Ahn, "UML-Based Representation of Role-Based Access Control," *Proc. 9th IEEE Int'l Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises*, 2000 pp. 195-200.
11. T. Lodderstedt, D. A. Basin, and J. Doser, "SecureUML: A UML-Based Modeling Language for Model-Driven Security," *Proc. 5th Int' l Conf. on the Unified Modeling Language*, 2002, pp.426-441.
12. T. Doan, *et al*, "MAC and UML for secure software design," *Proc. ACM Workshop on Formal Methods in Security Eng.,* 2004, pp. 75-85.
13. OASIS, "Security Assertion Markup Language (SAML) Version 2.0," URL: http://www.oasis-open.org/, accessed on 03/18/2006.
14. OASIS, "eXtensible Access Control Markup Language (XACML) version 2.0," URL: http://docs.oasis-open.org/xacml/, accessed on 03/18/2006.
15. WS-Security, URL: http://www.ibm.com/developerworks/, accessed on 03/18/2006.
16. WS Security Policy, URL: http://www.ibm.com/developerworks/, accessed on 03/18/2006.
17. K. Bhargavan, C. Fournet, and A.D. Gordon, "A semantics for web services authentication," *Proc. 31st ACM Symp. on Principles of Programming Languages*, 2004, pp.198-209.
18. E. Bertino, *et al*, "Temporal Authorization Bases: From Specification to Integration," *Jour. Computer Security*, vol. 8(4), 2000, pp. 309-354.
19. S. Jajodia, *et al*, "Flexible Supporting for Multiple Access Control Policies," *ACM Trans. on Database Systems*, vol. 26(2), 2001, pp.214-260.
20. A. Uszok, *et al*, "KAoS policy and domain services: toward a description-logic approach to policy representation, deconfliction, and enforcement," *Proc. IEEE 4th Int'l Workshop on Policies for Distributed Systems and Networks*, 2003, pp. 93-96.
21. S. S. Yau, *at el* "Automated Agent Synthesis for Situation-Aware Service Coordination in Service-based Systems", *Technical Report,* Arizona State University, August, 2005.
22. S. S. Yau, Y. Yao and V. Banga, "Situation-Aware Access Control for Service-Oriented Autonomous Decentralized Systems", *Proc. 7th Int'l Symp. on Autonomous Decentralized Systems*, 2005, pp. 17-24.