Situation-Aware Access Control for Service-Oriented Autonomous Decentralized Systems

Stephen S. Yau, Yisheng Yao, Vageesh Banga
Department of Computer Science and Engineering
Arizona State University
Email: {yau, yisheng.yao, vageesh.banga}@asu.edu

Abstract

Service-oriented autonomous decentralized systems (S-ADS) have been presented to address the extreme dynamism in large-scale information systems. In S-ADS, various capabilities are independently constructed and managed by different providers as autonomous services that are distributed over various types of networks, including wireless and wired networks. One of the key challenges in S-ADS is to have an effective access control mechanism that can meet the dynamic and diverse security requirements of various users and providers of an S-ADS system.

Current access control mechanisms can hardly meet this challenge due to lack of situation-awareness. In this paper, a situation-aware access control approach is presented, which is middleware-based and integrates situation-awareness capability and Role Based Access Control (RBAC) models to provide a practical solution for access control in S-ADS. The situation-aware RBAC model is designed for specifying dynamic access policies in an S-ADS system. Due to the situation-awareness capability of our approach, flexible and high-grained access policies can be specified and enforced for various providers and users.

Keywords*: Access control, situation awareness, service-oriented computing, role-based access control, middleware-based, smart classroom*

1. Introduction

Recently, service-oriented autonomous decentralized systems (S-ADS) have been presented to address the extreme dynamism in large-scale information systems [16, 18, 21]. In S-ADS, various capabilities are independently incorporated and managed by different providers as autonomous services [11, 16, 19] distributed over various types of networks, including wireless (infrastructure or ad hoc) and wired networks. The roles of users and providers can be dynamically changed in different situations. If necessary, and the autonomous users and providers may form a group, called an *autonomous community*, for cooperation. One of the key challenges in S-ADS is to have an effective access control mechanism that can meet the dynamic and diverse security requirements of various users and providers in S-ADS. This is very important for S-ADS to be acceptable for various applications, where only authorized users can access the services of S-ADS systems. Hence, it is necessary to have an effective access control mechanism that is flexible, scalable and adaptable to the changing user requirements and environment of S-ADS. The inherent ad hoc nature of S-ADS with users coming and leaving frequently does not allow S-ADS to define the access rights in advance. Current approaches to access control in distributed systems fail in S-ADS due to the assumption that S-ADS have only relatively static security requirements [8]. Access control policy specification languages for service-oriented systems have been developed [2, 17] for separating policy specification and policy enforcement so that access control policies can be easily administrated. New access control mechanisms to satisfy these specific requirements are needed in S-ADS systems.

In this paper, we will present a situation-aware access control (SA-AC) model for S-ADS and a middleware-based approach for enforcing SA-AC policies in S-ADS based on our SA-AC model. Our SA-AC model will incorporate situation-awareness constraints into role-based access control model (RBAC) [1, 22], Our middleware-based approach will enable users to enforce SA-AC policies for SA-ADS efficiently. Our approach is based on our SA-AC model, a policy specification language for specifying SA-AC policies, and a situation-aware middleware [25, 26] for monitoring situation changes and providing run-time support for enforcing SA-AC policies in S-ADS.

2. Requirements of Access Control in S-ADS

Before presenting the requirements of access control in S-ADS, let us consider an example with collaborative learning using a Smart Classroom [23] as shown in Figure 1, where students are in a class "Software Engineering Project (SEP)". Students in this class are divided into small groups (with 5-7 students per group) to practice a software engineering project. Each student as well as the instructor in the Smart Classroom carries a PDA or a Tablet PC. In this example, both PDAs and Tablet PCs are considered mobile devices equipped with a situation-aware middleware, like the Reconfigurable Context-sensitive Middleware RCSM [25], and various sensors, such as sensors for location, noise and light. Each student group has a member, called *SQA,* taking care of Software Quality Assurance and another member, called *Outsourcing Manager (OM),* in charge of outsourcing. SQA monitors and coordinates the software development process and evaluates the product. OM manages the outsource contracts of appropriate components to other groups and the subcontracts from other groups. OMs from different groups get together to find the OMs appropriate subcontractors, represented by the dashed line as shown in Figure 1. Students of a group discuss various aspects of analysis and design of the group project, incorporate useful members' inputs, and make necessary changes to the documents. The instructor often participates in the group discussions and gives feedback to the groups. Each group may have a backup for the SQA in case the SQA is not available in the Smart Classroom.

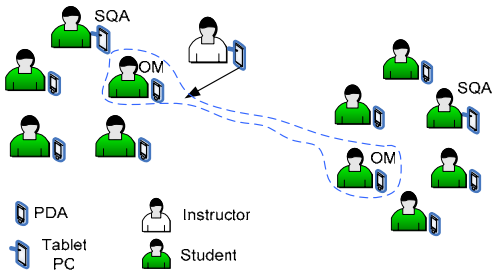

Figure 1. An example -- collaborative learning in Smart Classroom

For this scenario, a set of autonomous services are deployed on each PDA. For illustration purpose, we list three of these services as shown in Table 1**.** During runtime, these PDAs are autonomous, but they often need to collaborate to complete a task. If any service on a PDA becomes unavailable due to some reasons, such as overloaded by other tasks or not installed by mistake, then the PDA can search in the group to find and utilize available related services on other PDAs. Each group can be considered as an autonomous community. The OMs getting together to negotiate the outsourcing contracts form an autonomous community.

The instructor requires that each group should keep its development information confidential, except for the parts required for the subcontractors for outsourcing. In order to do this, the following access control policies need to be enforced by a group:

a) Only when the user with the role of SQA is in the Smart Classroom during the class time, he/she can create a group discussion by calling *CreateGroup* of o_{gm}.
b) When the SQA is not available in the group, the backup SQA can take place for SQA.
c) Only during the class time, the user with the role of the instructor can join any group discussion.
d) Only the OM can call *Request* of o_{om} to send an outsourcing request to the OM of another group.
e) Users can invoke service o_{dm} only during a group discussion and after two or more users joined the group.

Table 1. Sample services for the example

Service Name	Functionality
o_{dm}: Documentation Management	1. *SendDocument*: send documents to all group members *or* specified user(s). 2. *RetrieveDocuments*: retrieve documents from specified user(s) or all group members 3. *EditDocument*: change a specified document
o_{gm}: Group Management	1. *CreateGroup*: initiate a group discussion 2. *JoinGroup*: for user to join a group 3. *Scheduler*: for SQA to arrange the development schedule, including group meetings.
o_{om}: Outsourcing Management	1. *Request*: for OM to send a outsourcing request to an OM of other group 2. *QueryStatus*: for OM to get the development status of the outsourced software component

Assuming that a user is already authenticated by the system utilizing existing technologies, such as Public Key Infrastructure (PKI) or threshold cryptography [9, 14, 27], the following access control requirements are needed for specifying and enforcing the acess control policies.

R1) *Separation of duty: Separation of duty (SoD)* is considered valuable in deterring fraud since frauds can occur if an opportunity exists for collaboration between various users. Users in each autonomous community should be restricted under separation of duty constraints. "Separation of duty" requirement has been well studied in role-based

access control models [1, 4, 10, 13]. It can be either static or dynamic. *Static separation of duty (SSoD)* requirements can be implemented simply by statically restricting assignment of individual users to roles. For example, we can only assign the role of OM to one user of each group in the above example. *Dynamic separation of duty (DSoD)* is more difficult since the requirement can only be satisfied during system operation. However, DSoD allows more flexibility in system operation. For example, at any time instance, the role of SQA should be assigned to only one member of each group in the above example, but when the current SQA becomes unavailable, the backup SQA member will take place of the SQA.

R2) *Decentralized and distributed access control.* As shown in the above example, various access control policies from users, services and their communities need to be considered. These policies can be specified by various parties and stored in a decentralized environment. In order to enforce these policies, a mechanism for locating, retrieving and authenticating the policy components is needed. Policies defined by various parties may not be compatible and a method dealing with the compatibility of different policies related to an access decision request is needed.

R3) *Dynamic and simple access policies.* In S-ADS systems, smooth and secure interactions between the participating users and services require flexible access control policies, which should be able to address the highly dynamic and heterogeneous nature of the S-ADS environment. For example, in the above example, when the project is completed and the group allocation may change. This requires that access control policies be easily updated and understood by users.

3. Current State of the Art

In role-based access control models [1, 22], $RBAC_2$ defines constraints, which can be designated for user-role, role-permission and session-role assignments. RBAC is a promising solution for distributed environment. However, all RBAC models are relatively static [22]. Although most research has focused on specifying constraints for RBAC models [1, 3, 13], these approaches are often very complex and difficult to use to address the dynamism of S-ADS.

Context-based access control systems [7, 15] have been discussed in depth. For example, Covington *et al* [8] defined the environment context as an environment role. Permissions are assigned to the subjects if the environmental roles are evaluated as true based on the current context value. In most of these systems, context defines the current activity under which a subject is trying to access an object and depending on the current context the permissions are restricted. Access control can also depend on a certain sequence of events. Cholweka *et al* presented a context-sensitive access control model, in which the rights are granted based on the actual task [6]. These context-based access control approaches can be considered as special cases of our SA-AC approach because the situation in SA-AC is much broader than the context in these approaches.

For access control in service-oriented computing systems, Johnson, *et al* [12] presented an access control mechanism that enables multiple owners and administrators to define usage policies in the distributed system. Chadwick and Otenko [5] used the role-based access control model to provide access rights for the authenticated users of the system. Periman, *et al* [20] developed a mechanism for providing access to all the members of a community. These access control mechanisms mainly focus on static attributes, where access depends on the identity of the subjects involved. Another problem is that they require a centralized policy repository, which may not be available in S-ADS.

For security policy specification, several languages to specify security policies in distributed systems have been developed, such as SAML [17] and XACML[2]. SAML is an XML-based security language for exchanging authentication and authorization information, but it puts too much burden on services themselves by requiring them to gather the evidence needed for policy decision. XACML intends to provide a common language for specifying a wide-range of access control policies, but it still needs models for representing and analyzing the conditions of access control policies.

4. Situation-Aware Access Control Model

In this section, we will present our situation-aware access control (SA-AC) model for expressing dynamic access control policies. Figure 2 shows our SA-AC model, which extends the basic RBAC model by including the constraints in user-role and role-permission assignments as situations. A *situation* is defined as "a expression on previous device-action over a period of time and/or the variation of a set of contexts relevant to the application software running on the device over a period of time" [26]. A *context* is defined as "an instantaneous, detectable, and relevant condition of the environment or the device, such as time, location, light-intensity, noise-level, and available bandwidth" [26]. The situation information of a device in S-ADS can represent the access condition of the device and define the dynamic trust associated

with each of its users, thereby determining the access rights granted to him/her

Here, we use set theory to represent our SA-AC model so that it can have a simple syntax for easy understanding.

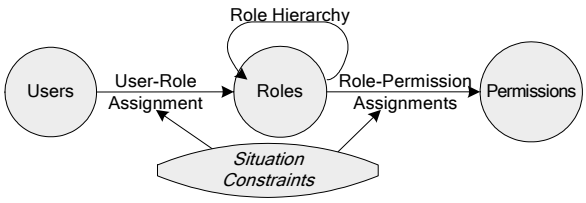

Figure 2. Overview of our SA-AC model

First, we define the following sets which are similar to traditional RBAC models:

- $\boldsymbol{U}$ = the set of users in the autonomous community
- $\boldsymbol{O}$ = the set of services to be provided and utilized in the autonomous community
- $\boldsymbol{R}$ = the set of roles defined inside the autonomous community. 2^R is used to represent the power set of R
- $\boldsymbol{P}$ = the set of permissions (or functions) defined on $\boldsymbol{O}$
- $\boldsymbol{RH}$ ($\subseteq \boldsymbol{R} \times \boldsymbol{R}$), the partial order ($\geq$ dominance) relation on $\boldsymbol{R}$, *i.e.* $\boldsymbol{RH}$ is the hierarchical structure of roles
- $\boldsymbol{UR}$ ($\subseteq \boldsymbol{U} \times \boldsymbol{R}$), the set of user-role assignments
- $\boldsymbol{RP}$ ($\subseteq \boldsymbol{R} \times \boldsymbol{P}$), the set of role-permission assignments

Then, we model SA constraints in user-role and role-permission assignments as follows:

- $\boldsymbol{SE}$ = the set of situation expressions. We use $\boldsymbol{2^{SE}}$ to represent the power set of $\boldsymbol{SE}$.
- $\boldsymbol{SEUR} \subseteq \boldsymbol{2^{SE}} \times \boldsymbol{UR}$, the set of situation-aware user-role assignments. $\boldsymbol{seur} = (\boldsymbol{L_{se}}, (\boldsymbol{u},\boldsymbol{r}))$ ($\in \boldsymbol{SEUR}$), defines that only if all the situation expressions in the situation expression list $\boldsymbol{L_{se}}$ ($\subseteq \boldsymbol{SE}$) are true, the assignment $(\boldsymbol{u}, \boldsymbol{r})$ ($\in \boldsymbol{UR}$) is active.
- $\boldsymbol{SERP} \subseteq \boldsymbol{2^{SE}} \times \boldsymbol{RP}$, the set of situation-aware role-permission assignments. $\boldsymbol{serp} = (\boldsymbol{L_{se}}, (\boldsymbol{r}, \boldsymbol{p}))$ ($\in \boldsymbol{SERP}$), defines that only if all the situations in the situation list $\boldsymbol{L_{se}}$ ($\subseteq \boldsymbol{SE}$) are true, the assignment $(\boldsymbol{r}, \boldsymbol{p})$ ($\in \boldsymbol{RP}$) is active.

The following utility functions are defined on the above sets for facilitating the evaluation of access control policies:

- *situation*($\boldsymbol{seur}$): $\boldsymbol{SEUR} \rightarrow \{true, false\}$ is a function returning the conjunction of all the situation expressions in an $\boldsymbol{seur}$, i.e. *situation*($\boldsymbol{seur}$): $= \wedge \{\boldsymbol{se_i} \mid \boldsymbol{seur}=(\boldsymbol{L_{se}},(\boldsymbol{u},\boldsymbol{r})) \wedge \boldsymbol{se_i} \in \boldsymbol{L_{se}}\}$. We call *situation*($\boldsymbol{seur}$) the `status` of the assignment $\boldsymbol{seur}$. If it returns true then the assignment $\boldsymbol{seur}$ is `active`. Otherwise, $\boldsymbol{seur}$ is `inactive`.
- *situation*($\boldsymbol{u}$, $\boldsymbol{r}$): $\boldsymbol{UR} \rightarrow \{true, false\}$ is a function returning the disjunction of the status of all $\boldsymbol{seur}$ related to ($\boldsymbol{u}$, $\boldsymbol{r}$), i.e. *situation* ($\boldsymbol{u}$, $\boldsymbol{r}$) $= \vee\{$*situation*($\boldsymbol{seur}$) $\mid \boldsymbol{seur} \in \boldsymbol{SEUR}\}$. We call *situation*($\boldsymbol{u}$, $\boldsymbol{r}$) as the `status` of the assignment ($\boldsymbol{u}$, $\boldsymbol{r}$). If it returns true, then the assignment ($\boldsymbol{u}$, $\boldsymbol{r}$) is `active`. Otherwise, ($\boldsymbol{u}$, $\boldsymbol{r}$) is `inactive`.
- *situation*($\boldsymbol{serp}$): $\boldsymbol{SERP} \rightarrow \{true, false\}$ is similar to the function *situation*($\boldsymbol{seur}$). We call *situation*($\boldsymbol{serp}$) the `status` of the assignment $\boldsymbol{serp}$. If it returns true, then the assignment $\boldsymbol{serp}$ is `active`. Otherwise, $\boldsymbol{serp}$ is `inactive`.
- *situation*($\boldsymbol{r}$, $\boldsymbol{p}$): $\boldsymbol{RP} \rightarrow \{true, false\}$ is similar to the function *situation*($\boldsymbol{u}$, $\boldsymbol{r}$). We call *situation*($\boldsymbol{r}$, $\boldsymbol{p}$) the `status` of the role-permission assignment ($\boldsymbol{r}$, $\boldsymbol{p}$). If it returns true, then ($\boldsymbol{r}$, $\boldsymbol{p}$) is `active`. Otherwise, ($\boldsymbol{r}$, $\boldsymbol{p}$) is `inactive`.
- *roles*($\boldsymbol{u}$): $\boldsymbol{U} \rightarrow \boldsymbol{2^R}$ is a function returning the roles assigned to the user $\boldsymbol{u}$ under the current situations, i.e. *roles*($\boldsymbol{u}$) $= \{\boldsymbol{r} \mid (\exists \boldsymbol{r'})\ [\boldsymbol{r'} \in \boldsymbol{R} \wedge (\boldsymbol{r'} \geq \boldsymbol{r}) \wedge$ *situation*($\boldsymbol{u}$, $\boldsymbol{r'}$)]$\}$

Finally, a situation-aware access control policy decision $\boldsymbol{sapd}$ is defined on $\boldsymbol{U} \times \boldsymbol{P}$ as follows:

$$\boldsymbol{sapd} = (\{(\boldsymbol{u}, \boldsymbol{p}) \mid (\exists \boldsymbol{r})\ [\boldsymbol{r} \in \textit{roles}(\boldsymbol{u}) \wedge \textit{situation}(\boldsymbol{u}, \boldsymbol{r}) \wedge \textit{situation}(\boldsymbol{r}, \boldsymbol{p})]\} \neq \{\})$$

The functions *situation*($\boldsymbol{u}$, $\boldsymbol{r}$) and *situation*($\boldsymbol{r}$, $\boldsymbol{p}$) search for all active user-role and role-permission assignments that are related to the specified $\boldsymbol{u}$ and $\boldsymbol{p}$. $\boldsymbol{sapd}$ checks whether there is a role that is activated under current situation for $\boldsymbol{u}$ to acquire $\boldsymbol{p}$. If we can find such a role in the role hierarchy, then $\boldsymbol{sapd}$ will be true, and the access decision will be positive.

5. An SA-AC Policy Language

We have developed an XML-based SA-AC language for specifying flexible SA-AC policies in S-ADS systems based on our SA-AC model. We believe our SA-AC policy specification language is simpler and easy-to-use than XACML [2], which intends to be a common language for expressing security policy. Moreover, the policies specified by SA-AC policy specification language can be automatically translated into XACML and is interoperable with XACML.

An SA-AC policy specification includes the following parts:

- *User* elements, which specify the ID, name and detail description of each user.
- *Role* elements, which specify the ID, name and the *parent* role identity of each role.
- *Permission* elements, which specify the ID, name and detailed description of each permission.
- *Situation* elements, which specify the ID, logical expression and the detail description of a situation expression. In order to make the policy easy-to-understand, we use the same notation for situation expressions in [26] and include them in SA agents. The situation ID will be referred as situation expression identity in the situation-aware user-role assignments and role-permission assignments.
- *SEUR* elements, which specify the situation expressions related to a user-role assignment. When all the situation expressions are true under

the current situation, the specified user in *SEUR* could activate the specified role. Different *SEUR* elements with the same user and same role will be disjunctively evaluated in the policy evaluation.

- *SERP* elements, which specify the situation expressions related to a role-permission assignment. When all the situation expressions are true under current situation, the users with the specified role could invoke the specified permission. Different *SERP* elements with the same role and same permission will be disjunctively evaluated in the policy evaluation.

Negation operator can be applied to the situation expressions to specify negative access policies. Figure 3 shows a fragment of the XML schema that defines our SA-AC specification language.

```
<?xml version="1.0" encoding="utf-8" ?>
<xs:schema targetNamespace="http://dpse.asu.edu/SAACPolicySchema.xsd" …>
<xs:element name="SAACPolicies">
 <xs:complexType>
  <xs:sequence>
   <xs:element name="Title" type="xs:string" minOccurs="0" />
    <xs:element name="user" minOccurs="1" maxOccurs="unbounded">
     <xs:complexType>
      <xs:sequence>
       <xs:element name="UserID" type="xs:string" minOccurs="1" />
        …
    <xs:element name="Role" minOccurs="1" maxOccurs="unbounded">
     <xs:complexType>
      <xs:sequence>
       <xs:element name="RoleID" type="xs:string" minOccurs="1" />
         …
    <xs:element name="Permission" minOccurs="1" maxOccurs="unbounded">
     <xs:complexType>
       <xs:sequence>
        <xs:element name="PermissionID" type="xs:string" minOccurs="1" />
        …
    <xs:element name="Situation" minOccurs="1" maxOccurs="unbounded">
     <xs:complexType>
      <xs:sequence>
       <xs:element name="SituationID" type="xs:string" minOccurs="1" />
       …
    <xs:element name="SEUR" minOccurs="1" maxOccurs="unbounded">
     <xs:complexType>
      <xs:sequence>
       <xs:element name="SituationID" type="xs:string" minOccurs="0"
                                                    maxOccurs="unbounded" />
       <xs:element name="UserID" type="xs:string" minOccurs="1" maxOccurs="1" />
       <xs:element name="RoleID" type="xs:string" minOccurs="1" maxOccurs="1" />
      </xs:sequence>
     </xs:complexType>
    </xs:element>
    <xs:element name="SERP" minOccurs="1" maxOccurs="unbounded">
     <xs:complexType>
      <xs:sequence>
       … //similar to SEUR element
    </xs:element>
   </xs:sequence>
  </xs:complexType>
 </xs:element>
</xs:schema>
```

Figure 3. A fragment of the XML Schema of SA-AC policy specification language

6. Our Approach to Enforcing SA-AC policies in S-ADS

Our approach is middleware-based and more efficient and error-prone in enforcing SA-AC policies in S-ADS because the enforcement can be systematically implemented using a situation-aware middleware [26]. We will discuss this in subsequent subsections.

6.1. Components for Enforcing SA-AC

As shown in Figure 4, the following components are implemented in the situation-aware middleware for S-ADS systems to support the enforcement of SA-AC policies:

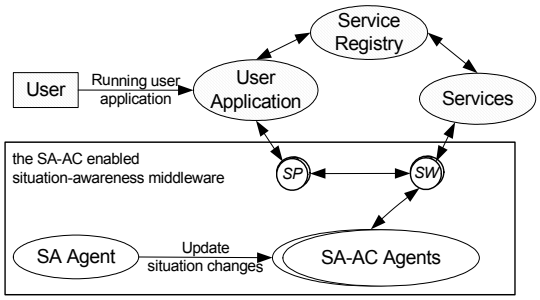

Figure 4. Overview of our approach to enforcing SA-AC in S-ADS

(1) *SA-AC Agents*: SA-AC agents are developed for evaluating access control policies and returning access decisions for secure S-ADS. Given the inputs for access control policies, requested access permission and the security properties of the access requester, SA-AC agents return the access decision under current situation according to the SA-AC model described in Section 4.

At run-time, SA-AC agents manage user-role and role-permission assignments as state machines. State transitions are triggered by changes in situations, which are monitored by an SA agent. The current state of the state machine defines the active role for each user and the active permissions for each role.

(2) *Situation-Awareness (SA) Agent*: SA agents are developed based our middleware for ubiquitous computing, RCSM [25, 26]. During runtime, SA-AC agents initially register the situations involved in the policies with SA agents, which continuously update changes of these situations to the SA-AC agents for dynamically activating the situation-aware user-role and role-permissions assignments.

(3) *Service Registries*: Service registries are distributed over all the computing nodes. A discovery protocol

is implemented in RCSM to discover the available services in autonomous community [24].

(4) *Service Proxies (SP)*: A service proxy is designed to handle the security information related to any service requests from the user. An *SP* generator, called *SPGen*, is provided to generate service proxies based on service specifications for user applications. The generated proxy will automatically bind a service request to the user's security properties, such as digital certificates that have been authenticated by other members in the autonomous community in advance.

(5) *Service Wrappers (SW)*: A service wrapper is a software entity guarding the access to the original services by interacting with SA-AC agents and other security services (e.g. authentication) before responding to any access request. A service wrapper generator, called *SWGen,* is provide to service providers during service deployment, to generate a SW for protecting the service.

At runtime, any service request ***req*** from the user application to the service will be bound to the user identity by the *SP*. Upon receiving the request, the *SW* and the SA-AC agents will follow the following process as to enforce the access policies:

(a) The *SW* uses service discovery mechanism in RCSM to discover the nearest SA-AC agent. If no SA-AC agent is discovered, goto (f).
(b) The *SW* initiate and send an policy evaluation request to the discovered SA-AC agent regarding ***req***
(c) The discovered SA-AC agent broadcast a policy discovery request to all SA-Agents through a secure channel for initializing a policy discovery process to find all applicable access policies related to the request ***req*** and the user.
(d) The nearest SA-AC agent determines whether the request ***req*** should be fulfilled according to the user identity and all the discovered access policies. If not, goto (f).
(e) Forward the request ***req*** to the service and send result from the service to the client. Goto (g).
(f) Deny the service request
(g) End of handling the service request.

In (d), the SA-AC agent will return the access permission decision for the users under the current situation according to our SA-AC model.

6.2. Enforcing SA-AC in S-ADS

Based on the above middleware components developed in RCSM, SA-AC policies in an S-ADS can be enforced by the following process:

(1) *Specify situation-aware access policy*. Using our SA-AC specification language, users of S-ADS can specify their access policies for securing the services and their applications. Send the specified access policies (as an XML document) to a local SA-AC agent or another SA-AC agent available on some computing nodes in the autonomous community through a secure channel. The access policies can be specified during development and updated at runtime.

(2) *Generate and deploy service wrappers*. Service providers use *SWGen* to generate *SW*s and deploy the generated *SW*s on the service hosts where the services are hosted. *SWGen* will remove the direct access to the services. After the deployment, only *SW*s can access the corresponding services.

(3) *Generate service proxies.* Developers of applications use *SPGen* to generate an SP for the user application. *SPGen* will automatically intercept users' service requests and binds them to users' security properties.

(4) *Run user application* When a user application is started, the *SP* is automatically loaded and intercepts all service requests from the user application to the service. Upon receiving the service request, the *SW* will validate the security information and enforce access control policies by communicating with SA-AC agents as described in Steps (a)-(g)

Among these four steps, step 2 to 4 can be automated using *SPGen* and *SWGen*. Step 1 needs to be specified manually. In Section 7 we will present an example to show how to follow these steps.

7. SA-AC for the Collaborative Learning Example

To illustrate our SA-AC approach, we have implemented SA-AC for the example described in Section 2.

The first step is to specify the SA-AC policies described in Section 2 using our SA-AC specification language in terms of users, roles, permissions, situation expressions, situation-aware user-role assignments, and situation-aware role-permission assignments for each group (or autonomous community). Figure 5 shows the SA-AC policies for policies a)-c) of one group in the example described in Section 2.

```
<?xml version="1.0" encoding="utf-8" ?>
<SAACPolicies xmlns="http://dpse.asu.edu/SAACPolicies"
              xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
              xsi:schemaLocation="http://dpse.asu.edu/SAACPolicies
              http://dpse.asu.edu/SAACPolicySchema.xsd">
    <Title>SA AC Policies for the Collaborative Learning Example</Title>
    <Description> This specification represents the following policies for the group
with Stephen S. Yau as the instructor, Yisheng Yao and Vageesh Banga as the two
of the group members: a) Only when the user with the role of SQA is in the Smart
Classroom during the class time, he/she can create a group discussion by calling
CreateGroup of o_gm. b) When the SQA is not available in the group, the backup SQA
```

```
can take place for SQA. c) Only during the class time, the user with the role of the
instructor can join any group discussion.
    </Description>
    <User>
        <UserID>ssyau</UserID> <UserName>Stephen S Yau</UserName> …
        </User>
    <User>
        <UserID>syao</UserID> <UserName>Yisheng Yao</UserName> …
        </User>
    <User>
        <UserID>vbanga</UserID> <UserName>Vageesh Banga</UserName> …
        </User>
    <!-- We omit other users here to save page space here -->

    <Role>
        <RoleID>R1</RoleID>  <RoleName>Instructor</RoleName> </Role>
    <Role>
        <RoleID>R2</RoleID>  <RoleName>SQA</RoleName> </Role>
    <!—We omit other roles to save space here -->

    <Permission>
        <PermissionID>Podm</PermissionID>
        <PermName>ManageDocument</PermName>
        <Description> … </Description> </Permission>
    <Permission>
        <PermissionID>Pogm.create</PermissionID>
          …     </Permission>
    <Permission>
        <PermissionID>Pogm.schedule</PermissionID> …</Permission>
    <!—We omit other roles to save space here -->

    <!-- Situation expressions are presented for users to easily understand the policy.
        They are just copied from the SA specifications in SA agents -->
   <Situation>
        <SituationID>S1</SituationID>
        <SituationExpression>(-5,0)[Location="BYENG468"]</SituationExpression>
        <Decription>During last five time units, user's location is in the Smart
                    Classroom BYENG468</Decription> </Situation>
    <Situation>
        <SituationID>S2</SituationID>
        <SituationExpression>[Time in (9:00am, 10:15am)]</SituationExpression>
        <Decription>Now, the time is class time for CSE461 (from 9:00am to
                                        10:15am) </Decription> </Situation>
    <Situation>
        <SituationID>S3</SituationID>
        <SituationExpression>S1 and S2</SituationExpression>
        <Decription>Current the user in the Smart Classroom for the course
                    CSE461</Decription> </Situation>
    <Situation>
        <SituationID>S4</SituationID>
        <SituationExpression>(-5,0)[john in Neiborlist]</SituationExpression>
        <Decription>John is not in the user's neiborlist.</Decription>
    </Situation>
    <!—we omit other situations to save space here -->

    <!-- specify the policy a), b) and c) listed in Section 2,
        omit other policies to save space-->
    <SEUR>   <SituationID>S3</SituationID>  <UserID>john</UserID>
             <RoleID>R2</RoleID>     </SEUR>
    <SEUR>   <SituationID>S4</SituationID>  <UserID>vbanga</UserID>
             <RoleID>R2</RoleID>     </SEUR>
    <SEUR>   <SituationID>S3</SituationID>  <UserID>ssyau</UserID>
             <RoleID>R1</RoleID>     </SEUR>
    <SERP>   <SituationID>S3</SituationID>  <RoleID>R2</RoleID>
             <PermissionID>Pogm.create</PermissionID>    </SERP>
    <SERP>   <SituationID>S3</SituationID>  <RoleID>R1</RoleID>
             <PermissionID>Podm</PermissionID>  </SERP>
</SAACPolicies>
```

Figure 5. SA-AC policies for the example

By applying four simple situation expressions in the situation-aware user-role and role-permission assignments, we can express policies a)-c) easily as shown in Figure 5. For simplicity, we omitted detail specification for other policies like policies d) and e), which can also be specified by using the SA-AC policy specification language, since the action history can be specified as situation expressions. As emphasized before, we specify situation expression in SA-AC policies only for the purpose of easy understanding. The SA-AC agents will not process these situation expressions. What SA-AC agents need to do is to let the SA agents know, which situation expressions (identified by the <SituationID> elements) they are interested in.

After the policies are specified for a group, they can be sent to every SA-AC agent available inside the group. Then the policies can be enforced following Steps (2) to (4) described in Section 6.1.

- Generate service wrappers all services listed in Section 2 and deploy these service wrappers and service on corresponding PDAs.
- Generate service proxies for user applications
- During runtime, the service wrappers will enforce these access control policies by discovering a nearest SA-AC agent and initiating and sending policy evaluation requests to the discovered SA-AC agent. The SA-AC agent will initiate a policy discovery process to discover all applicable policies for the service request and evaluate the policies and return the policy evaluation result to the corresponding service wrapper.

As policy discovery and evaluation is performed on the fly, policies can be updated and enforced on the fly. For example, when all the OMs need to get together for negotiating possible outsourcing contracts, a new autonomous community is formed. New policies for this group can be specified and enforced immediately.

8. Conclusion

In this paper, we have presented a situation-aware access control approach (SA-AC) for service-oriented autonomous decentralized systems. Because of the situation-awareness feature, this approach can dynamically enforce very flexible access policies as shown in the implementation of our example. A middleware-based approach is developed for easily enforcing SA-AC policies in S-ADS systems. Currently, we are developing domain ontologies for access control policies, which will greatly improve the interoperability among our SA-AC model and other access-control models. Additional work needs to be done in this area: Techniques for checking the

consistency of SA-AC policies and handling the incompleteness of SA-AC policies. The consistency of SA-AC policies is needed for ensuring correctness of policy decisions. Incompleteness of SA-AC policies needs to be dealt with because it is often impossible for users to define complete SA-AC policies in advance due to the dynamics of S-ADS.

Acknowledgment

This work is supported in part by National Science Foundation under grant numbers ANI 0123980 and ITR-CYBERTRUST 0430565. We would like to thank Dazhi Huang and Huan Jin for many helpful discussions.

References

[1] G.-J. Ahn and R. Sandhu, "Role-based authorization constraints specification," *ACM Trans. on Information and System Security (TISSEC)*, vol. 3(4), 2000, pp. 207--226.

[2] A. Anderson, "eXtensible Access Control Markup Language Version 2.0 (working draft)," 2004.

[3] E. Bertino, P. A. Bonatti and E. Ferrari, "TRBAC: A temporal role-based access control model," *ACM Trans. on Information and System Security (TISSEC)*, vol. 4(3), 2001, pp. 191--233.

[4] E. Bertino, E. Ferrari and V. Atluri, "The specification and enforcement of authorization constraints in workflow management systems," *ACM Trans. on Information and System Security (TISSEC)*, vol. 2(1), 1999, pp. 65--104.

[5] D. W. Chadwick and A. Otenko, "The PERMIS X.509 role based privilege management infrastructure," *Proc. 7th ACM Symp. on Access Control Models and Ttechnologies*, 2002, pp. 135--140.

[6] D. G. Cholewka, R. A. Botha and J. H. P. Eloff, "A Context-sensitive Access Control Model and Prototype Implementation," *Proc. IFIP TX 11 16th Annual Working Conf. on Information Security*, 2000, pp. 53-66.

[7] A. Corradi, R. Montanari and D. Tibaldi, "Context-based Access Control for Ubiquitous Service Provisioning," *Proc. 28th Annual Int'l Computer Software and Application Conf. (COMPSAC)*, 2004, pp. 444-451.

[8] M. J. Covington, P. Fogla, Z. Zhan and M. Ahamad, "A context-aware security architecture for emerging applications," *Proc. 18th Annual Int'l Computer Security Applications Conf.*, 2002, pp. 249-258.

[9] Y. G. Desmedt, "Threshold cryptography," *Proc. the 3rd Symp. on State and Progress of Research in Cryptography*, 1993, pp. 110-122.

[10] J. Hoagland, R. Pandey and K. N. Levitt, "Security Policy Specification Using a Graphical Approach," *Technical Report CSE-98-3*, University of California, Davis, 1998.

[11] IBM, "IBM Web Services architecture overview," 2004.

[12] W. Johnston, S. Mudumbai and M. Thompson, "Authorization and attribute certificates for widely distributed access control," *Proc. 7th IEEE Int'l Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises*, 1998, pp. 340-345.

[13] J. Joshi, "A Generalized Temporal Role Based Access Control Model for Developing Secure Systems," *Ph. D. Dissertation*, CERIAS, Purdue University, 2003.

[14] J. Kong, P. Zerfos, H. Luo, S. Lu and L. Zhang, "Providing robust and ubiquitous security support for mobile ad-hoc networks," *Proc. 9th Int'l Conf. on Network Protocols*, 2001, pp. 251-260.

[15] A. Kumar, N. Karnik and G. Chafle, "Context sensitivity in role-based access control," *ACM SIGOPS Operating Systems Review*, vol. 36(3), 2002, pp. 53-66.

[16] K. Mori, "Autonomy and community," *Proc. 6th Int'l Symp. on Autonomous Decentralized Systems*, 2003, pp. 330.

[17] OASIS, "Security Assertion Markup Language (Specification 1.1)," 2003.

[18] T. Ono, N. Kaji, Y. Horikoshi, H. Kuriyama, K. Ragab and K. Mori, "Autonomous decentralized community construction technology to assure quality of services," *Proc. 10th IEEE Int'l Workshop on Future Trends of Distributed Computing Systems*, 2004, pp. 299-305.

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

[20] L. Pearlman, V. Welch, I. Foster, C. Kesselman and S. Tuecke, "A community authorization service for group collaboration," *Proc. 3rd Int'l Workshop on Policies for Distributed Systems and Networks*, 2002, pp. 50-59.

[21] K. Ragab, N. Kaji and K. Mori, "Service-oriented autonomous decentralized community communication technique for a complex adaptive information system," *Proc. IEEE/WIC Int'l Conf. on Web Intelligence*, 2003, pp. 323-329.

[22] R. S. Sandhu, E. J. Coyne, H. L. Feinstein and C. E. Youman, "Role-based access control models," *Computer*, vol. 29(2), 1996, pp. 38-47.

[23] S. S. Yau, S. K. S. Gupta, F. Karim, S. I. Ahamed, Y. Wang and B. Wang, "Smart Classroom: Enhancing Collaborative Learning Using Pervasive Computing Technology," *Proc. 6th WFEO World Congress on Engineering Education & 2nd ASEE Int'l Colloquium on Engineering Education*, 2003, pp.

[24] S. S. Yau and F. Karim, "An energy-efficient object discovery protocol for context-sensitive middleware for ubiquitous computing," *IEEE Trans. on Parallel and Distributed Systems*, vol. 14(11), 2003, pp. 1074-1085.

[25] 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.

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

[27] L. Zhou and Z. J. Haas, "Securing ad hoc networks," *IEEE Network*, vol. 13(6), 1999, pp. 24-30.