2008 IEEE International Conference on Services Computing

Adaptive Secure Access to Remote Services *

Hanping Lufei, Weisong Shi and Vipin Chaudhary†

Department of Computer Science
Wayne State University and †University at Buffalo, SUNY
{hlufei, weisong}@wayne.edu, vipin@buffalo.edu

Abstract

Since the inception of service-oriented computing paradigm, we have witnessed a plethora of services deployed across a broad spectrum of applications, ranging from conventional RPC-based services to SOAP-based Web services. Likewise, the proliferation of mobile devices has enabled the remote "on the move" access of these services from anywhere at any time. Secure access to these services is challenging especially in a mobile computing environment with heterogeneous modalities. Conventional static access control mechanisms are not able to accommodate complex secure access requirements. In this paper, we propose an adaptive secure access *mechanism to address this problem. Our mechanism, which consists of two components:* an adaptive access control module *and* an adaptive function invocation module*, not only adapts access control policies to diverse requirements, but also introduces function invocation adaptation during access. We have successfully applied the proposed adaptive secure access mechanism to a computer-assisted surgery application called UbiCAS. Performance evaluation shows that with limited overhead, our technique enforces secure access to the services provided by the UbiCAS system in a flexible way.*

1 Introduction

Service-oriented computing [3, 4, 22] is one of the main approach to build distributed applications on the web. With the growth of heterogeneity in mobile computing environments, secure access to services is becoming more challenging in the design of these applications. We abstract the requirements of secure access to remote services as follows:

a) **Adaptation:** In a heterogeneous environment, like the Internet, it is very difficult, if not impossible, to build a one-size-fits-all approach that accommodates all diverse requirements. Adaptation has been considered as a general approach to address the mismatch problem between clients and servers [15, 21]. For secure access, there are two diverse requirements. On the user side, different configurations, such as diverse devices and network bandwidths are coexisting. On the service side, there are different data formats, security requirements, and so on. Hence, we have to adapt access control policies to such diverse requirements.

b) **Efficiency:** For some applications, secure access enforcement incurs negligible system overhead. However, a poorly designed technique will not scale well and perform poorly with increasing system size. Hence, we need to meticulously design the enforcement algorithm that introduces minimum performance overhead.

c) **Evolvability:** System requirements usually keep evolving over time. A good secure access control framework should have the capability to extend current access policies. A user friendly interface should also be defined for administrators who may need add/update access control policies frequently to meet the evolved system requirements.

In this paper, we propose an adaptive secure access mechanism for remote services. Our approach contains two main modules: *an adaptive secure access control module* and *an adaptive secure function invocation module*. In the adaptive access control module, several definitions, such as context term, context instance, are formalized. An access control model is proposed to take application related contexts into consideration in the design of access control policies. The enforcement algorithm is proven to be more efficient than the conventional access control enforcement algorithm in terms of time complexity.

We envision that secure access is a comprehensive process for access control and service invocation procedure. For instance, even if a user is granted the access right to a service, she will not be able to access the service if the service required encryption mechanism is not currently available on the user side. To remedy this situation, we introduce an adaptive function invocation module. We have applied the proposed adaptive secure access mechanism to a real-world computer-assisted surgery application called UbiCAS [17]. Compared with existing solutions, this paper has the following three contributions: (1) *Adaptive Access Control* – We propose a general model to integrate application-oriented contexts into the design; (2) *Adaptive Function Invocation* –

*This research was supported in part by the Michigan Life Science Corridor Grant and National Science Foundation CAREER grant CCF-0643521.

978-0-7695-3283-7/08 $25.00 © 2008 IEEE
DOI 10.1109/SCC.2008.12

Besides adaptive access control, an adaptive function invocation module is also proposed. Users' diverse contexts sometimes mismatch the requirement of functions, and this in turn degrades the performance of functions or fails to execute the function at all; and (3) *Implement secure access for UbiCAS system* – We have successfully implemented our models in a distributed computer assisted surgery system called UbiCAS.

The rest of the paper is organized as follows. The system structure is introduced in Section 2. Then we present the adaptive access control module and adaptive function invocation module in Section 3 and Section 4, respectively. Section 5 depicts the details of implementation and evaluation. Finally, related work and concluding remarks are listed in Section 6 and Section 7, respectively.

2 System Structure

In a typical service-oriented computing, the server defines some functions including the implementation, the parameters, and the interfaces. If a user wants to use the remote service, she calls one of the functions by following the function interface. After the server finishes the task of the function, the result will be sent back to the user. In the following context, we use function and service interchangeably. Given this context, this section presents the system structure of the proposed adaptive secure access mechanism, which consists of two major components, *an adaptive access control module* and *an adaptive function invocation module*, as shown in Figure 1.

2.1 Adaptive Access Control Module

The adaptive access control module enforces the access control policy. It constrains what a user can do, as well as what programs executing on behalf of the users are allowed to do. In this way the access control module seeks to prevent the activity that could lead to breach of security.

The adaptive access control module has two parts, *the access control point (ACP)* and *access policy database*, as shown in Figure 1. When a client wants to access remote functions, first, he needs to provide his context information, such as role, time, location, to ACP to acquire the access right to the desired functions. With the support of access control policy database, ACP is able to do the access control enforcement using some algorithms which will be discussed in Section 3. Access control policies are stored in the access policy database in the form of an ordered two-dimensional directed access control policy graph (ACPG), which will be explained also in Section 3.

2.2 Adaptive Function Invocation Module

After a user request passes the adaptive access control module, the adaptive function invocation module starts the procedure to adapt the invocation according to the user's dynamic contextual information. The motivation is that the same function call could experience totally different performances under different scenarios. As an example, calling an image download function could take intolerably long through a low bandwidth network, like dial-up, while being fast via a T1 connection. Therefore, the priority of the adaptive function invocation module is to select appropriate adaptive components to augment the function call in different scenarios. Furthermore, if the selected components require user side deployment, these components will be delivered to the user side and be plugged into the running space of the user side program.

Adaptive function invocation module includes *function invocation point (FIP)*, *function pool*, and *adaptive component database* as shown in the lower part of Figure 1. FIP receives user context information and acquires the adaptive function invocation graph, which will be described in Section 4, from the function pool. Then FIP interacts with the user to decide the mandatory components for the function invocation and delivers the components from the database to the user if necessary.

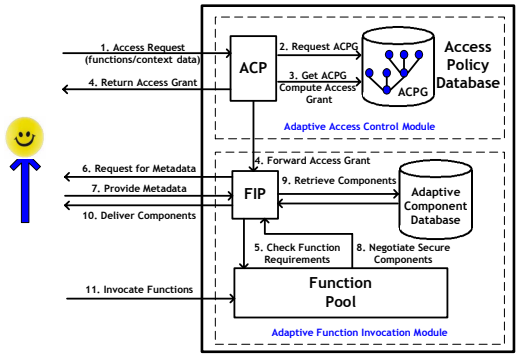


Figure 1. An overview of the adaptive secure access mechanism.

2.3 Secure Access Procedure

Figure 1 also shows the whole adaptive secure access procedure. After ACP receives the access request from a user in step 1, it will request and receive the ACPG from the access policy database (steps 2 and 3). Then ACP will enforce the access control policy based on the user provided context information. Next, the access control result will be forwarded to FIP and returned to the user by ACP (step 4). If the access is permitted, FIP will retrieve the adaptive function invocation graph from the function pool (step 5). In steps 6 and 7, FIP requests and receives the user metadata which contains the contextual information of the user environment, such as network bandwidth, CPU type, and so on. In step 8, FIP negotiates security-related components with

the user based on the user *metadata* and the adaptive function invocation graph. If the user does not have the components, FIP will retrieve them from the database and deliver to the user side (step 9, 10). Finally the user application can dynamically deploy the components into the running space and start the function call (step 11). It is worth noting that in step 8, the negotiation procedure is general and not limited to security-related components, and components such as latency-related components can also be negotiated. Since the focus of this paper is secure access, we consider only security-related components.

3 Adaptive Access Control Module

Grimm *et al.* argue that *embracing contextual change* is the key to expose the dynamic changing context [10] to the application so that the adaptation can be conducted accordingly. Therefore, instead of using static access control policies, we introduce a general context in the design of the access control module. The context data represents all parameters related to the access control policy that an application defines, e.g., roles, locations, time, and so on. This design has two benefits. First, any context parameter that a real application required can be regulated and enforced in the access control model. Second, the general context definition is evolvable to the extended access control requirement in the future. Next, we introduce the access control policy definitions and the policy graph, then present the enforcement algorithms.

3.1 Access Control Policy Definitions

DEFINITION 1. *(Context Term). A Context Term is a tuple CT = (name, range, order) where name is the name of the context, range is the value set of the context and order is the method to order the values in the set.*

CT describes one context type from three aspects. First, CT^{name} is the name of the context, for example, the age of users. Second, CT^{range} is the range of the context possible values, for example, from 10 years old to 70 years old, [10, 70]. Third, CT^{order} is the method for ordering the context values in the range. For instance, the ascending order of age as [10, 11, 12, ... 68, 69, 70].

DEFINITION 2. *(Context Instance). A Context Instance of Context Term i, CT_i, is a couple CI = (name, value) where $name = CT_i^{name}$ and $value \in CT_i^{range}$.*

This definition shows one value of the context term. If the context term is represented as a vector like the x axis in two dimension coordinate space, then the context instance is a dot on the axis.

DEFINITION 3. *(Function). A Function in the remote service is defined as F = (name, $(input_i, i \in [1,n])$, $(output_i, i \in [1,m])$, $(com_i, i \in [1,k])$), where name is the signature of the function, $input_i$ is the i^th input parameter of the function, $output_i$ is the i^th output result of the function and com_i is the i^th necessary adaptive component for the function invocation. The number of inputs, outputs, and adaptive components are n, m, and k, respectively.*

This definition for function is similar as the traditional function definition except that we add the extra components which are necessary for the adaptive function invocation.

DEFINITION 4. *(Service). A Service is the collection of related functions.* $Service = \bigcup Function_i, i \in [1,n]$.

DEFINITION 5. *(Context Space). The Context Space is defined as* $CS = \bigcup CT_i, i \in [1,n]$. *For each function f, a Context Sub Space (CSS) is* $CSS_f \subseteq CS$.

Context Space is the whole space of the sum of all context terms. For each function, some of the context terms will be used to do the adaptive access control enforcement. These context terms form a subset of the context space, the context sub space (CSS). Later in this section, we will present the data structure built upon CSS to represent the access control policies.

DEFINITION 6. *(Context Instance Node). A Context Instance Node is a data structure CIN = struct{CI_i; Ptr(CI_j); Ptr(CI_k); ...} where CI_i is a context instance of context term i and Ptr(CI_j) is a pointer directed to a context instance of context term j, given $i \neq j \neq k$.*

DEFINITION 7. *(Context Instance Root Node). A Context Instance Root Node is a CIN. ϕ denotes NULL. CIRN = $\{CIN : \phi \rightarrow CIN\}$. Therefore, $CIRN \subseteq CIN$.*

Context Instance Node is built upon a context instance. It has one or more pointers to point to other context instances of different context terms. For an access control policy, CIN marks each controlled value of each context term and connects them together to form the policy graph. CIRN is a special CIN which is not pointed by any other CIN. For example, in Figure 2 the inverted triangle shapes are CIRNs, and rectangular shapes are CINs. Given these definitions, we can now present the structure of the access control policy graph (ACPG).

Figure 2 illustrates the access control policies for a function. There are $n+1$ types of context terms involved, from CT_0 to CT_n, which are ordered based on some defined standard. For example, using alphabet ascendent order for *Role, Time, Location, and Speed* context terms, CT_0=*Location*, CT_1=*Role*, CT_2=*Speed*, and CT_3=*Time*. In the context space, the ordered context terms will help

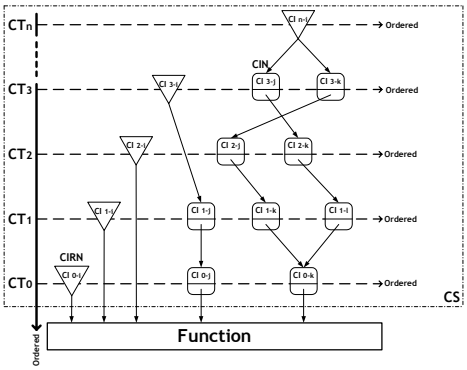


Figure 2. The adaptive access control policy graph.

us locate a specific context term faster than the sequential search. The context space consists of not only the ordered context terms but also the ordered context instances of every context term. Each dotted horizontal arrow line represents the ordered context instances for the corresponding context term. For instance, CI_{0_i}, CI_{0_j}, and CI_{0_k} are three ordered context instances of context term CT_0. The context instance and its possible pointers form a context instance node (CIN), denoted as rectangular (CINs) and inverted triangle (CIRNs) shapes in the figure. A linked path from a start CIRN to the function (lower part in the figure) is an access path, formally defined as follows.

DEFINITION 8. *(Access Path). A access path AP is an ordered sequence of CINs,* $CIN_i, i \in [1, n]$. *Symbol* $\rightarrow$ *means "point to",* ϕ *means "none". In an access path,* $CIN_n.Ptr \rightarrow Function$ *and* $\phi \rightarrow CIN_1$. *In other words,* CIN_1 *is a CIRN,* $CIN_i.Ptr \rightarrow CIN_{i+1}$.

From Definition 8, we know that the access path is defined as a directed link list of CINs starting from a CIRN. For example, $CI_{3-i} \rightarrow CI_{1-j} \rightarrow CI_{0-j}$ is an access path, which means that if a user's context instances of context terms CT_3, CT_1, and CT_0 match the values of CI_{3-i}, CI_{1-j}, and CI_{0-j} respectively, this user is granted the access right to the function. The set of all the access paths form the access control policy of the function. For instance, in Figure 2 the access control policy of the function has six access paths.

In order to support evolvability, we define an abstract policy class, which acts as an interface to specifying the context terms required by the application access control policy. For a specific access control requirement, we need to materialize a concrete policy class that inherit the abstract class and extend the context terms. Based on these policy classes, the system will build an ACPG. A detailed example is given in Section 5. This evolvable design enables system administrators to easily and flexibly add/update access control policies. In the next section, we describe the access control enforcement algorithms.

3.2 Access Control Policy Enforcement

ACPG is predefined and saved in the access policy database as shown in Figure 1. Several related algorithms are proposed to perform the access control policy enforcement. First, when the access control point (ACP) in Figure 1 receives the context metadata from a user and retrieves ACPG from the access policy database, the CINs corresponding to the context instances of the metadata are located by the algorithm `FindCIN`.

For each input context instance, `FindCIN` locates the corresponding context term. Since the terms are sorted, the computational complexity is $\mathcal{O}(1)$. Then `FindCIN` will find the input context instance (CI) in this CT's ordered value space. It also takes $\mathcal{O}(1)$ steps. If the CI has a CIRN or CIN in it, the CIRN or CIN will be returned. If n is the total number of context terms and i is the number of input context instances, then the total complexity of `FindCIN` is $\mathcal{O}(n)$. Next the ACP will use another algorithm called `AnyAccessPath`. This algorithm is used to check if there is any access path existing among the CINs returned by the `FindCIN` algorithm.

For each CIRN found by the `FindCIN` algorithm, `AnyAccessPath` searches if an access path started from that CIRN exists. If one access path is found, `AnyAccessPath` returns with true value, which means the access permission is granted to the user. Otherwise, the access is denied. Note that the core of `AnyAccessPath` is `HavePath`, which calls itself recursively to check the existence of a path starting from a CIN. Before we present the computational complexity of the algorithm, several facts are observed. Let k be the total number of CIRNs returned by the `AnyAccessPath` algorithm. Let j be the total number of other CINs returned by `AnyAccessPath`. First, because a user only provides one context instance for each context term, therefore, $k + j = n$, where n is the total number of context terms. Second, making decision whether that *CIN.Ptr* is one of the CIN set $\{ CIN_1, CIN_2, ...CIN_j \}$ is $\mathcal{O}(1)$ (line 9 of function `HavePath`) because the set is an ordered array. Let us assume each CIN has m pointers and the height of each access path is h. Then the best case of `AnyAccessPath` is $\mathcal{O}(n)$ while the worst case is $\mathcal{O}(mn^2)$.

Finally, the algorithm `GrantAccess` combines `FindCIN` and `AnyAccessPath` to decide the access permission. Obviously, the complexity is between $\mathcal{O}(n) + \mathcal{O}(n) = \mathcal{O}(n)$ and $\mathcal{O}(n) + \mathcal{O}(mn^2) = \mathcal{O}(n(1 + mn))$.

For comparison purpose, we also give the complexity of conventional access control method. Usually, conventional access control methods do not utilize the complex data structure to organize the contexts. For example, it defines the access as:

$$Permission := Clause_1 \bigcup Clause_2 ... \bigcup Clause_i$$
$$Clause := Context_1 \bigcap Context_2 ... \bigcap Context_j$$

Here, the *Clause* is similar to the notion of access path in our approach. The *Context* is like the context term. During the access control checking, the algorithm checks each *Clause* by comparing the *Context* value with the value received from the user. With the same previous parameters as what we have in previous analysis, i.e., n CTs, k CIRNs, m pointers for each CIN, and h steps along an average path, we can easily see that there are $\mathcal{O}(km^h)$ paths, or *Clause*. The complexity of the best case is $\mathcal{O}(n)$ which happens when the first clause satisfied. However, on average it requires to check $\mathcal{O}(nkm^h)$ steps, which is much worse than the worst case of our approach $\mathcal{O}(mn^2)$.

4 Adaptive Function Invocation Module

In most of the previous efforts on access control [5, 8, 24], the procedure to access the relevant resources on the server side is not described or neglected after the access right is granted to a user. In reality, as we have argued in introduction, the same service could have a variety of user-perceived behaviors (performance), depending on user-specific contexts, such as client hardware/software configurations, network connections, and so on. For example, some applications require all Windows clients have service pack 2 installed to access the resource even with valid user name and password. In this case, the user will either not be able to access the service or create some problems on the server side if her machine does not have the service pack 2 installed. To rectify this situation, we propose an adaptive function invocation approach that adjusts the function call according to the user's context. Combined with the adaptive access control module, our approach facilitates adaptive and secure access to remote services and provides the best possible performance/user experience to different users.

Specifically, adaptive components are utilized to adapt the function invocation. As shown in Figure 1, all the adaptive components are stored in the database. According to the specific requirements of a function, the function invocation point (FIP) will request the metadata from the user. Then FIP will decide which components are necessary for the function invocation. Finally, FIP retrieves those components from the database and delivers them to the user. Subsequently, user application can dynamically link the components into the running space and start the function invocation. Several techniques, such as mobile code [12] and dynamic class loading, can be used to plug in components on the fly.

Invocation requirements of a function are described as an function invocation graph, as shown in Figure 3. Usually a function has different requirements based on the context of access launched from a user, for example secure and communication optimization requirements. Given the requirements, one or more components are provided to adapt the invocation according to the diverse user configurations. For instance, if one function needs content encryption, for each user hardware configuration, a specific encryption algorithm should be employed, since different encryption algorithms have various performances on different platforms [19]. The adaptive function invocation module leverages the adaptive function invocation graph (AFIG) in Figure 3, which is similar as the access control graph, to define a directed path that connects user's multiple metadata to access control components.

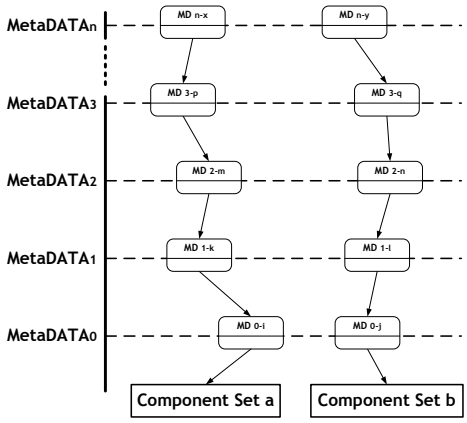


Figure 3. The adaptive function invocation graph (AFIG).

In Figure 3, each metadata line represents one kind of user configuration. Many modalities could influence the function invocation, such as processor type, network bandwidth, screen resolution, and so on. For one type, there are multiple values. For instance, if *MetaData 0* represents processor type, then *MD 0-i* could refer to PocketPC processor and *MD 0-j* could refer to Pentium Duo Core CPU. One linked list containing the different *MetaData* values directs to a set of components. The graph will guide FIP to find the component set for a specific user. We can see that the graph has similar structure as the access control policy graph in Figure 2, so previously formalized definitions and algorithms can be easily applied here. Therefore, details about the graph definitions and search algorithms will not be repeated. Next we will present an example of adaptive function invocation.

4.1 Adaptive Data Encryption in Function Invocation

Several functions in UbiCAS [17], such as image load function, segmentation function, demand content encryption for patient information security/privacy specified by the HIPAA standard [13]. Although many symmetric or asymmetric encryption algorithms have been proposed, it is difficult, if not impossible, to build one single encryption protocol which performs well in such dynamic an environment

as Internet. The only way to support effective secure function invocation is to provide a flexible encryption adaptation mechanism.

In [18], we found that different encryption algorithms have different performance on various platforms. Therefore, we choose AES and RC4 as the candidate encryption algorithms to adapt the function invocation. In the implementation, FIP will choose one encryption algorithm for a specific function invocation according to the operation system types, as shown in Figure 4. Note that usually Windows XP runs on laptops or desktops which normally has enough computing power to execute AES. However, for Smartphone- and PocketPC-like devices running Windows CE operating system, previous data confirms that AES algorithm is too heavy for them. Thus, RC4 might be a good choice for data encryption on this kind of platforms. The adaptation focuses on how to choose different algorithms in the context of symmetric encryption. The procedure to set up the symmetric key(s) is beyond the scope of this paper. It is very easy to set up the symmetric keys using the Diffie-Hellman [6] key exchange or certificate based authentication.

Note that this example is fairly straightforward, however, we think it is enough to illustrate the idea of adaptive function invocation. The performance result of encryption adaptation for function invocation will be presented in the next section.

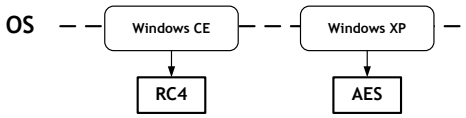


Figure 4. Adaptive encryption function invocation graph.

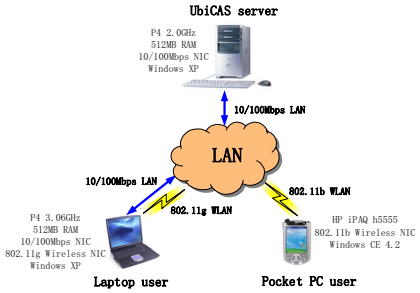


Figure 5. A simplified UbiCAS system deployment and configuration.

5 Implementation and Evaluation

An adaptive secure access system is built in UbiCAS [17], which is a distributed Computer-Assisted Surgery (CAS) system. Two parts, the adaptive access control module and the adaptive function invocation module are implemented using Java 1.5 SDK platform. In this section, we first present a brief introduction about the UbiCAS system is presented. Then we show the implementation of the adaptive access control and function invocation modules. Finally, performance evaluation results are presented.

5.1 The UbiCAS System

Computer-Assisted Surgery has broad applicability to human health. Traditional CAS systems are isolated solutions located in the operating room. Therefore, all the surgery data preparation, registration, segmentation, planning, and related operations are restricted to one physically fixed machine which reduces the potential for telepresence and telesurgery in CAS systems. UbiCAS extends the stand-alone CAS system into distributed environments. UbiCAS allows surgeons to retrieve, review and interpret multimodal medical images, and to perform some critical neurosurgical procedures on heterogeneous devices from anywhere at anytime. It has to handle several typical challenges in the mobile computing environment, such as security and privacy, multi-modalities of diverse network connections and data formats, surgery-related function implementation and conciliation on heterogeneous devices, especially on resource-constrained devices like PocketPCs and smartphones and so on.

The adaptive secure access to the UbiCAS server deals with the above issues. The adaptive access control module provides the controlled access to the functions. Then the adaptive function invocation module yields secure UbiCAS function invocation. A simplified system deployment and configuration is shown in Figure 5. The UbiCAS server, a laptop user and a PocketPC user connect together in one local area network. The laptop user has two network interfaces, Ethernet and 802.11g wireless. The PocketPC user has 802.11b wireless connection. The access control and function invocation modules are implemented in the UbiCAS server. They controls the secure access to two functions, the DICOM image load function and the image segmentation function.

5.2 Adaptive Access Control Implementation

Based on the adaptive access control module design, each function has its own access control policy graph. The class *Policy* is employed to describe the context terms in a policy as shown in the left side of Figure 6. It defines four context terms as *Role*, *Location*, *Time*, and *OS* (Operating System types). the `Policy` class provides a template to define policy for specific roles. An example policy class for doctor is shown in the right side of Figure 6.

After all policy classes are defined and implemented, the access control policy graph can be formalized as shown in

```
class Policy {                    class Policy-example extends Policy {
//Context term definitions        //Context term definitions
Role     role;                    Role     Doctor;
Location     location;            Location     Office;
Time     time;                    Time     0-24;
OS     os;                        OS     Any;
// methods                        }
....
}
```

Figure 6. The `Policy` class and an example.

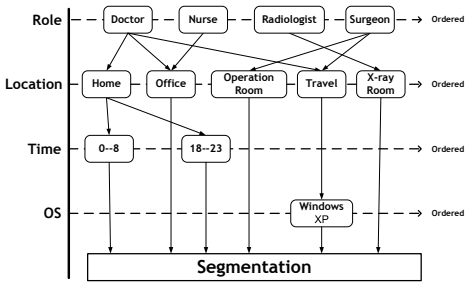


Figure 7. Adaptive access control policy graph in UbiCAS.

Figure 7, which illustrates the overall access control policy graph for the segmentation function. The figure can be easily read as follows. For example, a *doctor* at *home* can only access the *segmentation* function from 0AM to 8AM and from 6PM to 11PM. Following the graph, the access control enforcement algorithm can enforce the control based on each user's context data.

In the implementation we also instrument the user name and password authentication mechanism to work together with the access control module. Context term instances like role, location, user name, and password are provided by the user input. Operating system type is acquired by probing the system API without input from the user. Note that the location can be easily determined by other approaches, such as GPS or even 802.11 wireless network. Several attempts [2, 11, 23] address the location discovery for both indoor and outdoor scenarios. In the prototype implementation, we assume that the user provides the location automatically, however, it is trivial to leverage existing location discovery systems in a production system.

5.3 Adaptive Function Invocation Implementation

As mentioned in Section 4, diverse encryption algorithms have different performances on various platforms. It inspires us to adapt the encryption algorithms in the implementation of the function invocation module according to the adaptation policy shown in Figure 4. It is reasonable to make the assumption that Windows XP is running on desktop or laptop with sufficient resource for the AES encryption algorithm. Windows CE represents the PocketPC- and smartphone-like devices that have less power to run AES. Hence, a lightweight encryption algorithm, RC4, should be chosen instead.

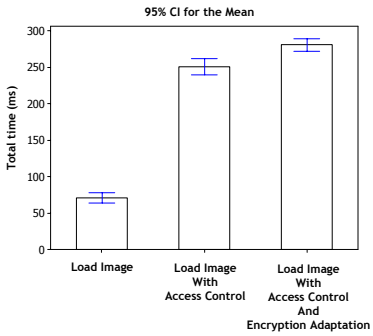
5.4 Performance Evaluation

We have conducted extensive evaluation the access overhead of two functions, *image loading* and *image segmentation*. Due to the space limit, we show the results of image load function only.

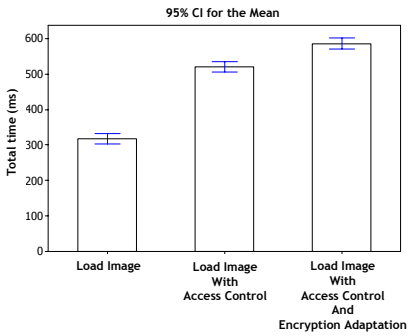
5.4.1 Image Load Function

In the secure access to the image load function, a user request will go through the two modules, adaptive access control and function invocation, before accessing the function. For demonstration purpose, we reduce the number of DICOM images to four in one patient case. Each function invocation will download the four images to the user side and load these images to the GUI interface running on the client side. Figure 8(a) shows three different function access scenarios for laptop users in LAN. In the first case, the image load function is called without access control and function invocation modules. In the second case the access control module is added. Finally, both the access control and adaptive function invocation modules are called before the image load function is called. In each case, the same experiment is repeated 150 times.

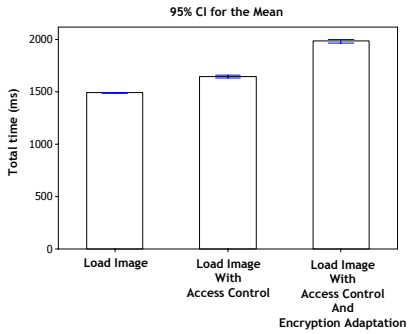
In Figure 8(a), where a laptop user accesses the function in a LAN, we can see that the invocation with access control yields more overhead, an increase of about 200 milliseconds, compared with the invocation without the access control. Since each invocation only triggers the access control module once, we think the overhead is acceptable. The total time increment due to encryption adaptation is not significant as shown in the rightmost bar. In Figure 8(b), where a laptop user access the function in a wireless 802.11b LAN, we can see that with the decreasing of the network bandwidth, the difference between the basic invocation and the invocation with access control is diminishing. The difference is closer in Figure 8(c), where a PocketPC user accesses the same function in 802.11b WLAN. To understand the reason of the diminishing difference, we illustrate the total time breakdown in Figure 8(d). We know that the image load function is a communication-intensive function. In low bandwidth networks, the transmission time dominates the total time, as shown by the second part of bars (Figure 8(d)). The access control times are approximately the same in the three scenarios. Server side encryption time is too small to be shown in the figure. The decryption time of the PocketPC user is larger than that of laptop user due to the limited computing resource of PocketPCs.



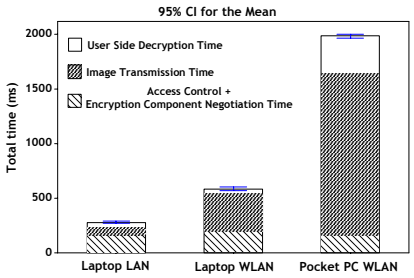
(a) Load Function for laptop in LAN



(b) Load function for laptop in 802.11b WLAN



(c) Load Function for PocketPC in 802.11b WLAN



(d) Breakdown of the image load function execution

Figure 8. Performance of adaptive secure access to load function.

Overall, secure access incurs 300%, 84%, and 33% more overhead compared with the direct image loading in three different scenarios, laptop in LAN, laptop in WLAN and PocketPC in WLAN in terms of the total time. The overhead diminishes as the network bandwidth reduces. In summary, the two proposed modules do not jeopardize the performance of secure access to the image load function.

6 Related Work

Our work shares its goal with several recent efforts that attempt to enforce access control for various objects in distributed environments and to inject adaptive functionality into application. We list the most relevant ones here.

In [1], Lampson *et al.* propose the concepts, protocols, and algorithms for access control in distributed systems, from a logical perspective. It also provides a logical language for access control lists and theories that decide whether requests should be granted. Sandhu *et al.* [24] introduce the role-based access control model, which efficiently associates permissions with roles rather than users to greatly simplify security management for administrators. Distributed Role-Based Access Control (dRBAC) [8] is a scalable, decentralized trust management and access control mechanism for systems that span multiple administrative domains. Temporal-RBAC (TRBAC) [5] is an extension of the RBAC model. TRBAC supports periodic role enabling and disabling and temporal dependencies among such actions, expressed by means of role triggers, which related to a different delay time. Generalized role-based access control (GRBAC) [20] leverages and extends the power of traditional RBAC by incorporating subject roles, object roles, and environment roles into access control decisions. In [14], a dynamic, context-aware security infrastructure is proposed to provide flexible, on-demand authentication, extensible context-aware access control to healthcare applications. [26] presents a delegation framework that can be used within the security framework of healthcare applications. Wilikens *et al.* discuss how to apply CBAC to healthcare applications in [25]. C-TMAC [9] extends TMAC by using general contextual information. Such contextual information can include the time of access, the location from which access is requested, the location where the object to be accessed resides, transaction-specific values that dictate special access policies and so on. [16] extends the RBAC by introducing the notions of role context and context filters to make RBAC sensitive to the context of an attempted operation. Edjlali *et al.* propose the history-based access control for mobile code [7]. The key idea is there is to maintain a selective history of the access requests made by individual programs and to use the history to improve the security differentiation. This approach provides a nice means for adaptation, and complements the

proposed adaptive secure access very well.

Our work is different from above mentioned previous efforts in the following ways. First, we do not introduce one specific context, like location of time into the access control model. Actually we systematically propose a general secure access mechanism to integrate any context that the application possibly needs. Finally, combining access control and adaptive function invocation is novel, and it is one of the early efforts enabling secure access of remote services in the coming service-oriented computing era.

7 Conclusions

In this paper, we propose an adaptive secure access mechanism for accessing remote services. Compared with previous efforts which handle predefined static contexts for access control, our approach is able to integrate application-oriented access control contexts into the system and to dynamically evolve the access control policy to handle future system requirements. Besides access control we also raise another important issue of adaptive function invocation which has not been addressed in any previous work. We have successfully implemented our models in a distributed computer assisted surgery system called UbiCAS. The performance evaluation on different configurations shows that our approach provides efficient secure access to remote services with an acceptable overhead In future, we plan to design an access control policy description language to work with our proposed enforcement algorithm. Enriching the function invocation module with more intelligently reactive adaptation is also an interesting research direction.

References

[1] M. Abadi, M. Burrows, B. Lampson, and G. Plotkin. A calculus for access control in distributed systems. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, p. 706C734, September 1993.

[2] P. Bahl and V. Padmanabhan. Radar: An in-building rf-based user location and tracking system. *In Proceedings of IEEE Infocom 00*, Apr. 2000.

[3] B. Benatallah, F. Casati, and F. Toumani. Web service conversation modeling: A cornerstone for e-business automation. *IEEE Internet Computing*, p. 46C54, 2004.

[4] D. Berardi, D. Calvanese, G. Giacomo, M. Lenzerini, and M. Mecella. Automatic service composition based on behavioral descriptions. *International Journal of Cooperative Information Systems*, p. 333C376, 2005.

[5] E. Bertino and P. Bonatti. Trbac: A temporal role-based access control model. *ACM Transactions on Information and System Security*, p. 191C223, August 2001.

[6] W. Diffie and M. Hellman. Multiuser cryptographic techniques. *IEEE Transactions on Information Theory* 22(1):644–654, Nov. 1976.

[7] G. Edjlali, A. Acharya, and V. Chaudhary. History-based access control for mobile code. *Proc. of ACM Conference on Computer and Communication Security*, pp. 38-48, Nov. 1998.

[8] E. Freudenthal, T. Pesin, L. Port, E. Keenan, and V. Karamcheti. drbac: Distributed role-based access control for dynamic coalition environments. *Proceedings of the 22 nd International Conference on Distributed Computing Systems (ICDCS 2002)*, July 2002.

[9] C. Georgiadis, I. Mavridis, G. Pangalos, and R. Thomas. Flexible team-based access control using contexts. *ACM Symposium on Access Control Models and Technologies*, May 2001.

[10] R. Grimm et al. System support for pervasive applications. *ACM Transactions on Computer Systems*, p. 421C486, November 2004.

[11] A. Haeberlen, E. Flannery, A. Ladd, A. Rudys, D. Wallach, and L. Kavraki. Practical robust localization over large-scale 802.11 wireless networks. *In Proceedings of ACM MobiCom04*, 2004.

[12] D. Halls. *Applying Mobile Code to Distributed Systems*. Ph.D. thesis, Computer Laboratory University of Cambridge, 1997.

[13] the american health insurance portability and accountability act, `http://www.hipaa.org/`.

[14] J. Hu and A. Weaver. Context-aware security infrastructure for distributed healthcare applications. *Pervasive Security, Privacy and Trust (PSPT2004)*, Aug. 2004.

[15] A. D. Joseph, J. A. Tauber, and M. F. Kasshoek. Mobile Computing with the Rover Toolkit. *IEEE Transaction on Computers:Special Issue on Mobile Computing* 46(3):337–352, Mar. 1997.

[16] A. Kumar, N. Karnik, and G. Chafle. Context sensitivity in role-based access control. *ACM SIGOPS Operating Systems Review*, July 2002.

[17] H. Liu, H. Lufei, W. Shi, and V. Chaudhary. Towards ubiquitous access of computer-assisted surgery systems. *Proceedings of the 28th Annual International Conference IEEE Engineering in Medicine and Biology Society*, Aug. 2006.

[18] H. Lufei and W. Shi. An adaptive encryption protocol in mobile computing. *book chapter of Wireless Network Security (Springer 2006)*, 2006.

[19] H. Lufei and W. Shi. Fractal: A mobile code based framework for dynamic application protocol adaptation. *Journal of Parallel and Distributed Computing*, p. 887C906, July 2006.

[20] M. Moyer and M. Ahamad. generalized role-based access control. *Proceedings of the 21st international conference on distributed computing systems*, 2001.

[21] B. D. Noble. *Mobile Data Access*. Ph.D. thesis, School of Computer Science, Carnegie Mellon University, May 1998, `http://mobility.eecs.umich.edu/papers/diss.pdf`.

[22] S. Paurobally and N. Jennings. Protocol engineering for web services conversations. *Engineering Applications of Artificial Intelligence*, 18, 2005.

[23] N. Priyantha, A. Chakraborty, and H. Balakrishnan. The cricket location-support system. *In Proceedings of ACM MobiCom00*, 2000.

[24] R. Sandhu, E. Coyne, H. Feinstein, and C. Youman. Role-based access control models. *IEEE Computer*, p. 38C47, February 1996.

[25] M. Wilikens, S. Feriti, A. Sanna, and M. Masera. A context-related authorization and access control method based on rbac: A case study from the health care domain. *Proceedings of the Seventh ACM Symposium on Access Control Models and Technologies*, June 2002.

[26] L. Zhang, G. Ahn, and B. Chu. A role-based delegation framework for healthcare information systems. *Proceedings of the Seventh ACM Symposium on Access Control Models and Technologies*, June 2002.