Authentication and Access Control in Multi-agent Systems

Pongsin Poosankam
School of Computer Science
Carnegie Mellon University
ppoosank@andrew.cmu.edu

Advisor: Prof. David Garlan
School of Computer Science
Carnegie Mellon University
garlan@cs.cmu.edu

Abstract

In a multi-agent system dedicated to personal task management, information and resources within the system are usually sensitive and should be accessible by a limited set of people. Some unique properties of such systems raise new engineering challenges for the design and implementation of security and access control mechanisms: interpretations of information by different agents may have different levels of granularity and information can flow through nodes belonging to different entities. We propose two design principles for determining when and how access control is enforced: (1) perform access control as early as possible; and (2) define policies controlling access at the information level. In this research, we focus on the RADAR Project as an example of such multi-agent systems. RADAR (Reflective Agents with Distributed Adaptive Reasoning) is a software-based cognitive personal assistant that helps people manage their routine tasks such as answering emails, scheduling meetings, and updating websites. In order to complete the tasks, agents in RADAR communicate with each other to obtain task-related information. In this paper, we describe how we apply the proposed design principles by implementing two levels of information access control policies in RADAR. The policies are configurable and can be applied efficiently in any multi-agent systems.

1. Introduction

Agents are entities in the environment, including humans, applications, and services. A multi-agent system, such as CMU's RADAR project, is a distributed system that consists of more than one agent where the agents communicate and collaborate to complete their tasks. Tasks are well-defined, repeatable computer-supported activities that a user carries out over time. In most cases, an agent cannot solve a task alone and needs to communicate with other agents to complete the task.

Nodes in a multi-agent system are administered by different entities. Without any access restriction, an entity in the system would be able to access other entities' data regardless of how confidential the data is. On the other hand, as some tasks require collaborative actions from different entities, the system's capability in solving such tasks will be lessened if entities cannot access other entities' data at all. As a result, an access control mechanism is required for the system to achieve its highest capability and privacy protection.

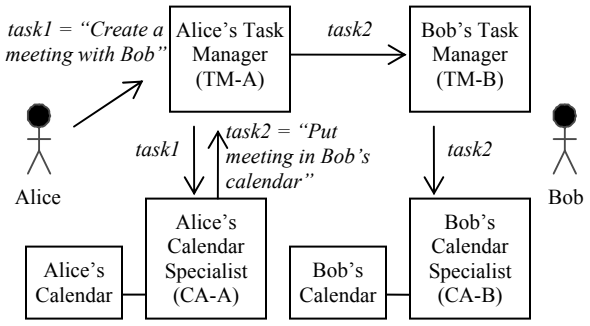


Figure 1: *Scheduling a meeting. When Alice wants to create a meeting with Bob, her request is sent to her calendar specialist (CA-A). Then, it makes another request to add the meeting to Bob's calendar. If successful, the meeting with be added to Alice's calendar.*

Let us demonstrate the unique engineering challenges for access control in multi-agent system with an example scenario. Figure 1 shows how users can request a meeting with other users in a multi-agent system. We call an entity that creates, manages, and updates user's tasks *Task Manager,* and entities that perform tasks *Specialists*. *Calendar Specialists* manage users' calendars and perform related tasks such as 'create new meetings' or 'find schedules.' In the process of creating a meeting, Bob's calendar is accessed and modified on behalf of Alice.

Based on Figure 1, we identify the following challenges for performing access control in multi-agent system:

- Tasks in multi-agent system, like *task2*, can be derived from other tasks, like *task1*. The initiators of these tasks can also be different entities: they can be users like Alice, task managers, or specialists. Although *task2* is a sub-task of *task1*, their contents are different. This may lead to different access control decision.

- Because entities in multi-agent system are specialized on different kind of tasks, they interpret tasks at different levels of granularity. B's calendar specialist (CA-B) interprets *task2* as "a request to add a meeting to Bob's calendar if possible." Whereas, B's task manager (TM-B) interprets it as "a calendar-related task that should be forwarded to CA-B." As a result, access control performed by different entities, even with the same set of policies, will give different results.

- The agents in the system can be administrated by different users. Thus access control needs to ensure that the tasks and other information flow only through agents authorized to access them.

In the rest of this paper, we elaborate on how these challenges affect the design of an access control architecture deployed in a multi-agent system. In section 2, we discuss the access control requirements of information available in a multi-agent system. In section 3, we present two design principles for such architecture. In section 4, we introduce RADAR as a multi-agent system that deploys the architecture. In section 5-7, we present the design of our access control mechanisms. In section 8, we discuss our experience gained in designing and implementing the access control mechanisms. In section 9, we discuss related work. We propose the future work and conclude the paper in Section 10 and 11.

2. Access Control Requirements

In addition to the engineering challenges discussed above, the access control infrastructure also needs to offer flexible ways for granting users and entities access to information and resources. Moreover, the performance drawback introduced by the access control enforcement should be minimal. The access control requirements for multi-agent system are the following:

- User must be able to specify access control policies on his or her information and resources managed by any specialists based on the task initiator, the task type (scheduling, direct access to information, etc.) and other constraints such as time and location. For example, Bob may give Alice an access to his calendar managed by any of his calendar specialists (Bob may have multiple calendar specialists). He may allow Alice to schedule a meeting with him (Alice as a task initiator) in his office (location constraint) at least two days in advance (time constraint).

- User must be able to configure the access control policies in the future. The access control policies should be easy to edit. The changes should take effect immediately.

- The performance drawback introduced from access control enforcement should be minimal. Obviously unauthorized request should be dropped before a costly operation is performed.

3. Access Control Design Principles

Based on the engineering challenges discussed in Section 1 and the access control requirements in Section 2, we propose two design principles for determining when and how access control is enforced in a multi-agent system:

3.1 Perform Access Control Early

This principle suggests that access control policies should be enforced as soon as possible without altering the access decision. Performing access control early in the process gives multi-agent system an opportunity to improve its performance. For example, if one of Bob's policies is allowing no one other than his colleagues to interact with him via the multi-agent system and Alice is not in his colleague list, her meeting request will obviously be denied. If this design principle is applied, the request should be denied before it reaches Bob's calendar specialist as the policy can be enforced earlier. As a result, useless operation can be ignored, thus improving the overall system performance.

3.2 Users Define Policies at Information Level

This principle suggests that users should be able to issue policies control access to information and resources at the information level instead of issuing them at the level of individual agents. Policies at information level includes policies that are based on nature of information and resources and also the types of task requests (scheduling, direct access to information, etc.). This principle is based on the observation that a multi-agent system consists of many specialists and multiple of them may work on the same kind of tasks or manage the same type of information

and resources. For example, Bob may have two calendar specialists, *CA-B1* and *CA-B2* with the same functionality. He should be able to issue a policy such as "Alice can access my weekday calendar" instead of "Alice can access my weekday calendar on specialist *CA-B1* by submitting a request via her task manager." He should also be able to issue policies based on the type of requests (tasks). The policy "Alice can do anything with my calendar but not my e-mail" will allow any calendar tasks but deny other types of tasks.

4. RADAR

In this research, we focus on the CMU's RADAR project as an example of multi-agent system. RADAR (Reflective Agents with Distributed Adaptive Reasoning) is a research project to build a software-based cognitive personal assistant that helps people manage their routine tasks such as answering emails, scheduling meetings, allocating resources, maintaining websites, and accessing different kinds of information directly.

4.1 Concept of Operation

Tasks in RADAR are well-defined, repeatable computer-supported activities that a user carries out over time. They may be interleaved with other tasks. Although they exhibit certain regularities, tasks are carried out differently for different people, and even for the same person, differently in different contexts [10].

The RADAR project is concerned with assisting users in their computing tasks by exploiting deep knowledge about how they carry out tasks. As a result, RADAR can automate many of the things that users would otherwise have to handle manually. The context for this assistance is a computing environment similar to today's computing platforms and applications, but augmented with special system components, called *specialists*, whose responsibility is to act on behalf of users to simplify and streamline user tasks. An example of *specialist* is one that helps a user manage his calendar, scheduling meeting, handling room reservations, dealing with scheduling conflicts, etc [10]. Different specialists are designed to perform different tasks. Currently, we have four specialists working (i.e. email, calendar, allocation, and web specialists). Nevertheless, RADAR architecture is designed to be extensible and new specialists can be added at any time.

A *Personal RADAR Space* is a collection of task specialists that acts like a good secretary for a user. A RADAR space contains a Task Manager that all specialists in the RADAR space connect to. *Task Manager* is a special component working as a mediator in a RADAR space. It provides central policy management, serves as an observer of the system, and dispatches tasks to appropriate specialists.

Figure 2 illustrates Personal RADAR Space architecture.

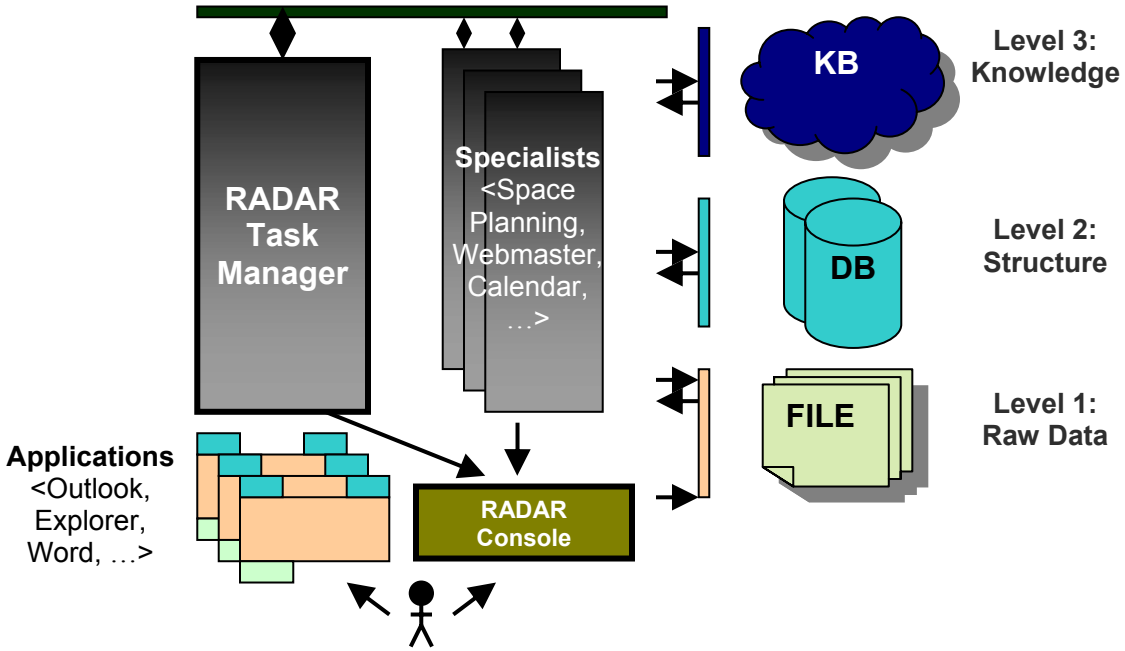


Figure 2: *Personal RADAR Space Architecture Overview*

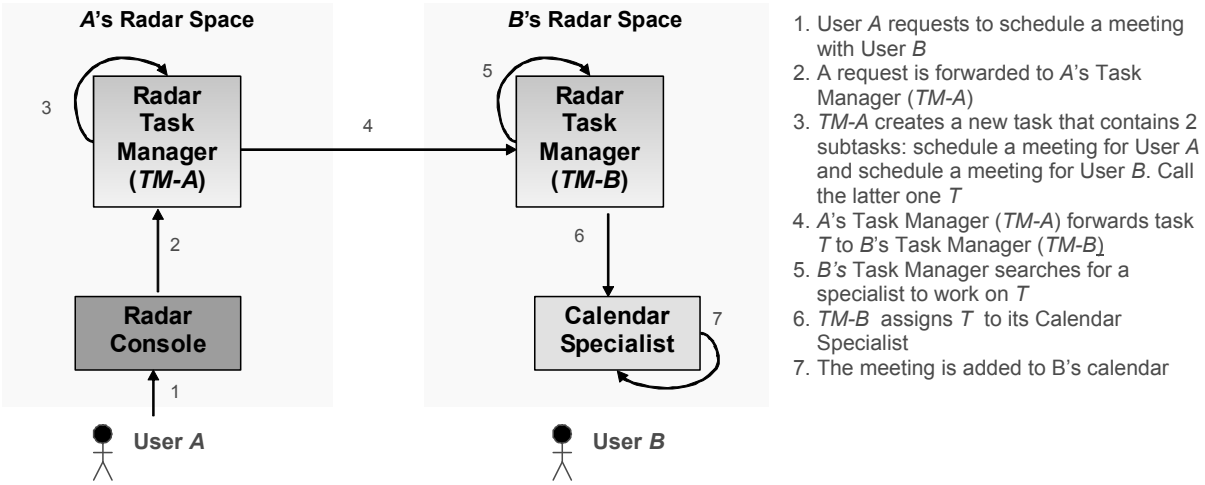


Figure 3: *Information/Resource Access Scenario. The diagram illustrates the task flow when a user's request involves accessing another user's resource. A problem arises when User B does not want User A to access his or her calendar. In that case, access control needs to be done at some point in order to protect User B's privacy.*

4.2 Task's Life Cycle

In RADAR, tasks are owned by the user of the RADAR space, but its lifecycle is controlled by the Task Manager. Specialists can request a change in the task state, although the request may be denied if the Task Manager does not authorize it.

A task is created when the Task Manager accepts the task creation request. The request can be done in many different ways:

- A user manually makes a request through the user interface of legacy applications or RADAR applications.
- A specialist requests a task creation as a subtask of another task.

Depending on the nature of the task, the Task Manager may decide to assign it to a specialist, extract more information from it, or forward it to another RADAR space, etc.

4.3 Access Control in RADAR

As one of multi-agent systems, a personal RADAR space requires collaboration between its specialists in order to complete complicate tasks. Specialists should be able to forward its task to other specialists when needed. However, without any restriction, RADAR specialist would be able to assign malicious tasks to other specialists. In addition, RADAR user would be able to access, via inter-personal RADAR space communication, other users' information and resources regardless of how confidential they are. As a result, an access control mechanism is required for RADAR to achieve its highest capability and privacy protection.

4.4 Information/Resource Access Scenario

Figure 3 illustrates a scenario that requires access control in RADAR: User *A* requests to schedule a meeting with User *B*. A problem arises when User *B* does not want User *A* to access his or her calendar. In such a case, access control enforcement needs to be done at some point in *B*'s RADAR space in order to preserve *B*'s privacy.

5. RADAR Authentication

In this section, we describe the importance and design of RADAR authentication (login) mechanism. RADAR authentication is a login-logout process required for all RADAR users before and after each use of their Personal RADAR Space. Each RADAR user has his or her unique username to be used at time of authentication. Usernames are also used to distinguish RADAR users from one another.

5.1 Role of Authentication in Access Control

When a user or specialists belonging to the user try to access some information or resource, RADAR consults with access control policies to determine whether the user is authorized to access the information/resources or not. Access control enforcement would not work if a user could trick the system and pretend to be other users with privileges. As a result, in order to prevent identity forgery, user authentication must be done before running access control mechanisms.

In addition to user identity assurance, RADAR authentication module also provides a basis for applying message encryption-decryption within a RADAR space. By performing mutual-authentication between two entities in a RADAR space, both entities obtain each other's credentials (e.g., public keys) used to encrypt and decrypt communications between the pair of entities based on a supported encryption mechanism (e.g., public-private keys encryption).

5.2 Choices of Authentication Mechanisms

- **Simple Username and Password.** By using this authentication mechanism, RADAR server will keep a list of its users along with their usernames and their encrypted passwords. When users login, the server verifies whether their usernames and passwords match with those stored. This mechanism is unfavorable because an attacker can sniff users' passwords at the time of authentication.

- **Kerberos with GSS-API.** This was our choice in the early implementation. Kerberos is a highly secure authentication mechanism in which users' passwords, even the encrypted ones, are not sent out to the RADAR server. With this authentication mechanism, RADAR username and password will be Kerberos username and password which are not stored in the RADAR server. To perform authentication using Kerberos, RADAR user and the RADAR server need to know nothing about each other, but both must trust a third party. This third party is called an authentication server (AS). In addition to Kerberos, GSS-API (Generic Security Service API) provides an interface for accessing Kerberos. Although Kerberos is very secure, Kerberos infrastructure including the AS is required to use it. As a result, this mechanism is preferred in the environment with existing Kerberos infrastructure. On the other hand, setting up an AS requires a lot of configuration and it is not recommended for small RADAR systems such as ones running on a local machine.

- **SASL.** We use SASL (Simple Authentication and Security Layer protocol) as our current authentication mechanism. It defines how authentication data is exchanged between a RADAR client's machine and the RADAR server. In addition, it is an interface of different authentication mechanisms such as Kerberos and simple username/password authentication. There are a number of standard SASL mechanisms for various levels of security and deployment scenarios. These range from no security (e.g., anonymous authentication) to high security (e.g., Kerberos) and levels in between. RADAR authentication module currently supports four SASL mechanisms including anonymous authentication and Kerberos5. The choice of mechanism is decided at time of authentication and it can be changed by the RADAR administrator. SASL is flexible and can be used in any computing environments.

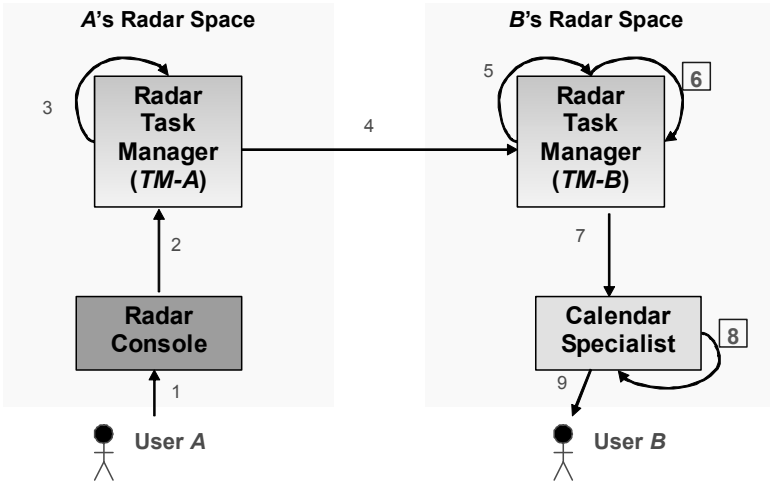
6. RADAR Access Control

By observing the sample information/resource access scenario in Section 4, we found that a user's information/resources access control mechanism must be placed in his or her personal RADAR space in order to ensure policy enforcements. If the mechanism is placed somewhere else, the user and his or her agents will have no control over it. An attacker can then easily alter the policies and attain more privileges in the user's information and resources.

6.1 Choices of Enforcements

For designing RADAR access control mechanism, we propose two levels of access control enforcements:

- **Coarse-grained Access Control in RADAR Task Manager.** We propose to implement a mechanism that accepts a general access control policies which can be applied to any tasks and any specialists. Although, unlike specialists, the Task Manager cannot interpret content of tasks in detail, it knows basic properties of tasks such as task initiator, task receiver, and type of task. As a result, it can perform access control based on the relationship between the task initiator and the task receiver. The policies will be in the form: "Specialist *A* on behalf of user *X* can submit a task of type *T* to specialist *B* working for user *Y*" where *A, X, T, B,* and *Y* may be replace by "any." Although this kind of policies cannot be performed alone because it is too coarse-grained and insufficient, it helps reduce specialists' burden and improve the system performance in the cases of obviously unauthorized access.



1. User *A* requests to schedule a meeting with User *B*
2. A request is forwarded to *A*'s Task Manager (*TM-A*)
3. *TM-A* creates a new task that contains 2 subtasks: schedule a meeting for User *A* and schedule a meeting for User *B*. Call the latter one *T*
4. *A*'s Task Manager (*TM-A*) forwards task *T* to *B*'s Task Manager (*TM-B*)
5. *B*'s Task Manager (*TM-B*) searches for a specialist to work on *T*
6. *TM-B* performs coarse-grained access control on *T*
7. If allow, *T* is forwarded to the Calendar Specialist
8. The Calendar Specialist performs fine-grained access control based on detailed content of *T*
9. May request a confirmation from User *B*

Figure 4: *Two levels of access control enforcement. (6) and (8) are when Task Manager and specialist enforce policies.*

- **Fine-grained Access Control in specialists.** We propose to implement a mechanism that enforces highly detailed access control policies which may be applied to some specific types of tasks and to some specific specialists. This kind of policies can be enforced in specialists because they are specialized to perform specific kinds of tasks and to understand the tasks' specific details. The policies will be in the form: "Specialist *A* on behalf of user *X* can <read/write> information *D* belonging to specialist *B* working for user *Y* based on constraint *C*" where *A, X, D, B, Y,* and *C* may be replaced with "any." Although this kind of policies is fine-grained and sufficient, enforcing it alone may lead to large performance drawback, as all tasks are forwarded to appropriate specialists and are interpreted in detail before really being enforced.

6.2 Combination of Two-Level Enforcements

We decided to implement two levels of access controls discussed above; a coarse-grained one in Task Manager and a fine-grained one in specialists. Although the fine-grained policy enforcement alone is sufficient, we place the coarse-grained policy enforcement in order to improve the performance.

Figure 4 illustrates the same scenario in figure 3 with two levels of access control policies being enforced. When a task reaches the Task Manager, it determines which specialists are responsible for the task. Based on the knowledge of responsible specialists, the task initiator, and task type, the Task Manager performs access control on the policies of the form: "Specialist *A* on behalf of user *X* can submit a task of type *T* to specialist *B* working for user *Y*" where *A, X, T, B,* and *Y* may be replace by "any."

If the task violates the policies on the Task Manager, a refuse message will be replied to the task initiator. Otherwise, the task will be forwarded to the appropriate specialists. The specialist can extract the task content in detail. It can also perform information access control on any information needed in order to complete the task. The access control policies at this level will be in the form: "Specialist *A* on behalf of user *X* can <read/write> information *D* belonging to specialist *B* working for user *Y* based on constraint *C*" where *A, X, D, B, Y,* and *C* may be replaced with "any."

7. Implementation

Similar to other components of RADAR, its authentication module has been implemented in JAVA. RADAR authentication module consists of two parts: Login Interface and Administrative Interface. RADAR access control module will consist of three parts: coarse-grained access control in Task Manager, a framework for fine-grained access control in specialists, and the interface of user-defined policies.

7.1 RADAR Login Interface

A login user interface (Figure 5b) verifies users' usernames and passwords and obtain users' pending tasks and their personal preferences from the database. We implement a login user interface to RADAR on top of SASL mechanism. In order to give a user more convenience, the login interface is compatible with existing authentication tools such as KClient and Leash. For example, after a user log-in using Leash, he or she

can use the obtained Kerberos ticket for logging in to RADAR without reentering his or her username and password again.

When launched, RADAR login interface establishes an RMI connection with the RADAR Task Manager and obtains the name of SASL mechanism to be used for authentication process. Possible SASL mechanisms are:

- *CRAM-MD5:* Support a hashed username/password authentication
- *DIGEST-MD5:* Support HTTP digest authentication
- *GSS-API:* Support GSS-API Kerberos5 auth.
- *NONE:* No authentication is required

RADAR Login application will display a GUI to collect username and password from a user. It will then perform SASL mutual authentication process with the RADAR Task Manager server to verify that the given username and password are correct and that the user is authorized to access his or her personal RADAR space at the time. If the user is authorized, a RADAR console will be created for the user. RADAR console and Task Manager will communicate with each other using SASL-established secure JMS connection. The user can use RADAR console to manage his or her personal RADAR space until logging out.

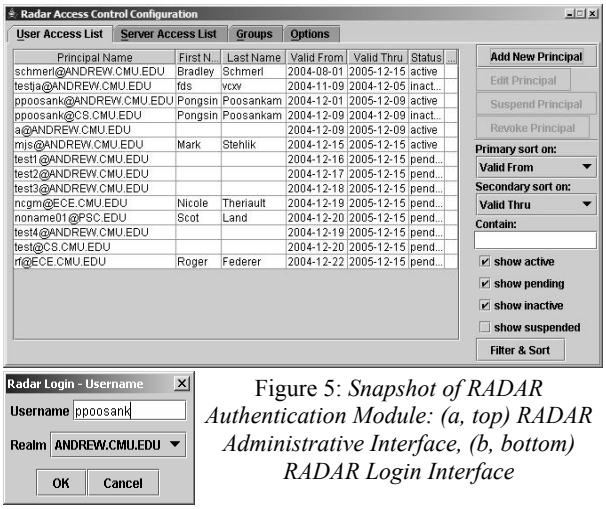


Figure 5: *Snapshot of RADAR Authentication Module: (a, top) RADAR Administrative Interface, (b, bottom) RADAR Login Interface*

7.2 RADAR Administrative Interface

RADAR Administrative Interface (Figure 5a) is an interface for user account management. RADAR Administrator can use this tool to register new RADAR users, revoke users, and update their information. This tool is also used to specify SASL access control mechanisms to be applied in user authentication process.

7.3 An Interface for User-Defined Policies

As stated in an access control requirement in Section 2 and in a design principle in Section 3.2, we propose an interface for user to define access control policies that can be converted to the two-level access control policies explained in Section 6.2. The interface will be in the form of "wizard" tool that consists of five steps or more:

1. Specify the task initiators to apply policy
2. Specify the task type to apply the policy
3. Give the specific information to apply the policy
4. Give constraints based on the task type
5. Specify a receiver specialist to apply policy

Each step in the wizard can be skipped which will mean "any." For example, skipping the step 2 means "any type of task will apply the policy." When the wizard is completed, the user-defined policies will be converted to coarse-grained and fine-grained policies and will be added to the Task Manager and the applied specialists, respectively.

8. Discussion

In section 3.2, we argue that enforcing access control policies early will improve the overall performance of a multi-agent system. Although this argument sounds reasonable, it is not always correct. In the case that earlier enforcement steps yield significantly more expensive operations, applying our design principle to a system will lead to a performance drawback. However, in most multi-agent system architectures, interpretation of received information (including tasks and requests) usually grows larger in its granularity as the information flows. As a result, our design principle is valid in most cases, including in RADAR.

By implementing access control upon receiving tasks in Task Manager and specialists, we ensure that the access control enforcement is done as early as possible. Although we perform redundant policy checking (because only fine-grained policies are sufficient already, the coarse-grained policies are then redundant), we only check the grant of access at most twice: first in the Task Manager and later in one of the specialists. Moreover, the access control in Task Manager is relatively simpler and can be done in more efficient way than the ones in specialists. As a result, the performance drawback from redundant policy checking on the Task Manager is small. Conversely, performance gain, resulting from replacing expensive policy checking in specialists with the cheaper one in the Task Manager, is comparatively large. As a result, in the

system with a considerable rate of unauthorized requests, its overall performance will be improved if applying our design principle. On the other hand, if the system has a negligible rate of policy violation, its overall performance will be worsened.

During the refinement of our access control design, we found our design principles to be contradicted with each other. Namely, to follow the first principle: "perform access control early," we need to place an enforcement mechanism in the Task Manager; however, the second principle: "users define policy at information level" prevents us from placing an enforcement mechanism in the Task Manager because the Task Manager does not have enough knowledge of information that is related to the task it received. At the end, we introduce a wizard tool to resolve the problem. By using the tool, user can specify policies at information level and they will also be automatically converted to policies that are enforceable by the Task Manager.

9. Related Work

Urs Hengartner and Peter Steenkiste also proposed to define policies controlling access to people location at information level [3]. In their project, information flows through multiple nodes from its source to the destination. Unlike in RADAR, the source and the destination usually have at most two nodes in between (in the case that information flows from one specialist to another specialist via two Task Managers) [2]. The information in their scope may also changes its granularity before reaching the destination.

Another related work introduces a human trust management model and framework that facilitates the construction of trust-aware mobile systems and applications [1]. The model support: reasoning about trust, dissemination of trust information in the network, and derivation of new trust relationship from previously formed ones. Each node in the system has a set of credentials that are used to prove its trustworthiness to other nodes. In other words, this work attempts to create a trust computing environment in mobile system with cognitive trust relationship. Although this work does not directly related to our current work, some of its goals are similar to those of our future work discussed in the next section. Namely, it attempts to construct a cognitively secure computing environment.

10. Future Work

The implementation of RADAR Access Control Module will be continued in May 2005. Coarse-grained access control enforcement will be placed in Task Manager and a framework for specialists' fine-grained access control enforcement will be available for specialist developers. Wizard tool for user to define access control policies will be implemented based on the proposal in Section 7.3.

In addition to application-based RADAR login and access control modules, we planned to implement web-based version of the modules as well. They will be done using JSP under CMU infrastructure. Another possible future work is to extend our work to deploy cognitive learning as one of RADAR's overall goals.

11. Conclusions

In this paper, we examined engineering challenges and access control requirements for information and resources available in a multi-agent system. We then presented two design principles that we applied in our access control mechanism design for CMU's RADAR project. The mechanism consists of two parts: coarse-grained and fine-grained access control enforcements. As a basis for access control policy enforcement, we implemented an authentication module on top of SASL. Currently, we are implementing the interface for users to define access control policies on their personal information and resources.

Acknowledgements

We would like to thank Dr. David Garlan, the advisor for this research, and Dr. Bradley Schmerl, a systems scientist working on the task manager architecture and backend implementation, for their ideas, support, and comments on the project.

We would also like to thank Mark Stehlik for organizing the CS Senior Thesis program, as well as for all of his advices concerning this project.

References

[1] L. Capra. Engineering Human Trust in Mobile System Collaborations. *SIGSOFT FSE*, pages 107-116, Oct 2004.

[2] U. Hengartner and P. Steenkiste. Access Control to Information in Pervasive Computing Environments. In *Proceedings of 9th Workshop on*

Hot Topics in Operation Systems (HotOS IX), Lihue, HI, pages 157-162, May 2003.

[3] U. Hengartner and P. Steenkiste. Implementing Access Control to People Location Information. In *Proceedings of 9th Symposium on Access Control Models and Technologies (SACMAT 2004), Yorktown Heights, NY, June 2004*, pages 11-20, June 2004.

[4] U. Hengartner and P. Steenkiste. Exploiting Hierarchical Identity-Based Encryption for Access Control to Pervasive Computing Information. *Technical Report CMU-CS-04-172,* October 2004.

[5] J. Kohl. The Kerberos Network Authentication Service (V5). RFC 1510. September 1993.

[6] J. H. Lee and A. Prakash. Malleable Shared Workspaces to Support Multiple Usage Paradigms. *Technical Report CSE-TR-370-98,* 1998.

[7] J. Linn. The Kerberos Version 5 GSS-API Mechanism. RFC 1964. June 1996.

[8] A. C. Long, C. Moskowitz, and G. Ganger. A Prototype User Interface for Coarse-Grained Desktop Access Control. *Technical Report CMU-CS-03-200,* November 2003.

[9] J. Myers. Simple Authentication and Security Layer (SASL). RFC 2222, October 1997.

[10] Radar Architecture Group. *The Radar System Architecture,* November 2004.

[11] Y. Wang and D. Garlan. TaskPort: A Task Management Interface in an Intelligent Cognitive Assistant System, May 2004.