Access Control Mechanisms for Inter-Organizational Workflow

Myong H. Kang, Joon S. Park and Judith N. Froscher

Naval Research Laboratory
Information Technology Division
4555 Overlook Ave. Washington, DC 20375
{mkang, jpark, froscher}@itd.nrl.navy.mil

ABSTRACT
As more businesses engage in globalization, inter-organizational collaborative computing grows in importance. Since we cannot expect homogeneous computing environments in participating organizations, heterogeneity and Internet-based technology are prevalent in inter-organizational collaborative computing environments. One technology that provides solutions for data sharing and work coordination at the global level is inter-organizational workflow. In this paper, we investigate the access control requirements for inter-organizational workflow. We then present access control solutions for inter-organizational workflow based on our implementation. Many of the requirements and solutions in this paper address the scalability of existing security solutions, the separation of inter-organizational workflow security from concrete organization level security enforcement, and the enforcement of fine-grained access control for inter-organizational workflow.

Keywords
Access control, Security, Organizational security, Enterprise, Workflow

1. INTRODUCTION
The Internet and business globalization have replaced the separation that was typical of the traditional business paradigm. Unconventional coalitions among businesses and nations are formed to advance common goals. These coalitions then quickly dissolve as individual objectives change. Threats now lie in these essential connections among participating enterprises, which also enable profitable cooperation. To facilitate these alliances, businesses and the military rely on distributed information technology (IT) for most operations. A secure computing infrastructure (e.g., secure network, firewall) is needed to support their missions. In addition to a secure computing infrastructure, the enterprise needs

- Flexible IT resources and infrastructure that allow rapid configuration,
- Secure distributed applications that can be easily constructed across enterprise boundaries, and
- Enterprise-level anomaly detection and recovery.

One technology that tries to provide solutions to the above problems is inter-organizational workflow. A workflow is a distributed application that interacts with uses and other applications to achieve common goals. Even though the above three requirements are equally important, we focus on the second item, especially access control issues. In this paper, we use “enterprise application” and “inter-organizational workflow” interchangeably because an inter-organizational workflow is an instance of enterprise applications. Figure 1 shows two inter-organizational workflows.

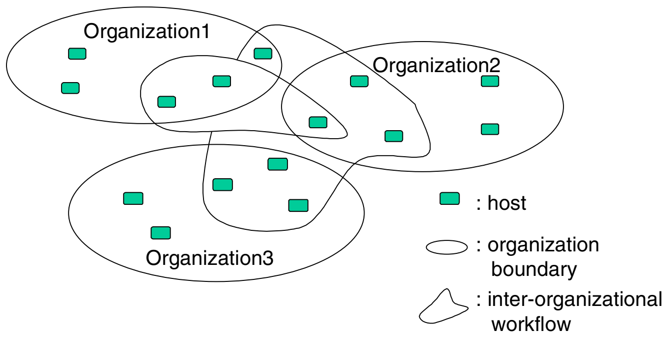

Figure 1. Two inter-organizational workflows

In figure 1, one workflow spans two (physical) organizations while the other workflow spans three (physical) organizations. We can view an inter-organizational workflow as a virtual enterprise that supports a specific mission. Once a workflow is designed, each task [3, Appendix] should be assigned to a specific organization and host (computer). In this example, we assume that hosts are connected via some networking mechanisms (e.g., Internet). Also note that multiple workflow tasks may be assigned to the same host.

In this paper, we study the access control requirements for inter-organizational workflows. We then present access control solutions for inter-organizational workflow based on our implementation. The rest of the paper is organized as follows. We briefly describe a prototype inter-organizational workflow management system (WFMS), SALSA, in section 2. Section 3 presents access control requirements for inter-organizational workflows. We review related access control research in section 4.

In section 5, we present access control mechanisms that have been implemented in SALSA. Section 6 summarizes this paper.

2. AN INTER-ORGANIZATIONAL WORKFLOW MANAGEMENT SYSTEM

An enterprise application that supports global, virtual enterprises may span multiple organizations and legal boundaries. Conventional WFMS cannot support such enterprise applications due to its mostly centralized architecture and functions. In most cases, the autonomy of users and organizations are greatly restricted due to centralized architectural and design considerations. Therefore, we need new inter-organizational WFMSs.

A WFMS, typically, consists of two parts; design-time and runtime tools. We constructed an inter-organizational WFMS, called SALSA [4], by implementing new design-time tools and extending an existing distributed CORBA-compliant workflow runtime engine, OrbWork[6] from University of Georgia. There is no centralized workflow engine in Orbwork. Instead each task contains a portion of the workflow specification that pertains to that task's interaction with other tasks in the workflow. Data resources that the task uses are also known as work-items. Our extension is based on one important requirement of inter-organizational workflows: the autonomy of participating organizations should be honored not only during the design phase but also during the runtime (execution) phase. In other words, different workflow designers may work on different portions of an inter-organizational workflow during the design phase. Also, multiple workflow runtime engines, which are managed by different organizations, may have to work together to accomplish a mission.

To support such autonomy among participating organizations, we extended OrbWork with *cooperative processes* [3] that allow multiple independent autonomous workflows to cooperate based on the contract among them. To make the contract among participating organizations rigorous, we introduce a *workflow domain* in our design tool. A workflow domain is a generic concept that can be used to represent each organization or even security domain [3]. A workflow in a workflow domain corresponds to an independent workflow during runtime. In other words, when an inter-organizational workflow is designed, it is a single workflow across multiple organizations. However, this single inter-organizational workflow design is split into multiple autonomous workflows, and they are deployed to participating organizations for execution. The number of independent workflows that will be produced is the same as the number of participating organizations or workflow domains [3]. In this way, we address the autonomy of organizations and their concern for mutual protection. In other words, each organization maintains and executes its portion of workflow, and any data to and from other organizations for receiving and sending should be exchanged according to the organization's security policies [3]. Any communication among workflow domains should follow the contract governing cooperation among independent workflows or organizations. The contract may specify what kinds of requests or data can be passed from one entity to another, when a response is expected. Since several portions of workflow design may be assembled to accomplish an enterprise level mission, it is important to validate that the overall design is consistent and sound. We provide translators for converting an inter-organizational workflow design into inputs to an existing Petri-net based analysis tool Woflan [11], and a model checking tool Spin [2], so that the consistency of the inter-organizational workflow design can be validated. The overall structure of SALSA is as follows:

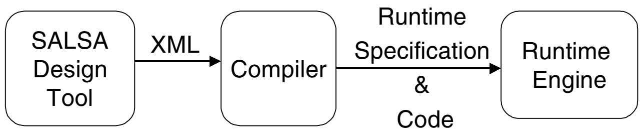

Figure 2. Internal structure of SALSA

The SALSA design tool allows application designers to specify mission/application logic, and the contract governing interactions among participating organizations. In other words, the designer can specify the follows:

– workflow domains that may represent organizations
– task specifications (e.g., inputs, outputs, invocation method for the underlying component) in each workflow domain, and
– control logic and data flow among tasks.

The SALSA design tool also allows application designers to hide complexity by providing a way to group related tasks into a more abstract higher-level task (i.e., the level of abstraction) [3, see Appendix]. The SALSA design tool saves the design specification in XML. When the workflow design is completed, the compiler reads the XML representation of the design, and generates Woflan or Spin inputs for design analysis and validation. Finally it generates runtime specification and code. Currently, we are using modified version of OrbWork as our runtime engine. OrbWork does not have a central scheduler; rather the scheduler is distributed with each task containing the code pertaining to it. Each scheduler only knows its predecessors and successors. Each OrbWork scheduler reads a task specification that was generated by the compiler and executes its role in the overall mission.

Briefly, OrbWork consists of the following CORBA servers: task servers, worklist server, data servers, and a monitor server. Each task server, which is a process from the operating systems' point of view, may contain more than one task [Appendix], where each is a separate thread in a task server. The worklist server maintains the lists of pending work for human tasks. Data servers act as a repository for data that is needed by tasks. A monitor server maintains the history of execution and answers queries from other servers and monitor clients. Since they are CORBA servers, they communicate with each other through CORBA's IIOP.

The task server and worklist server are not only CORBA servers, but also HTTP servers. When a human operator has to interact with worklist server (e.g., human task), he can do so through the HTTP protocol. Also when a human workflow manager needs to intervene for some reasons, he can do so through the HTTP protocol. Figure 3 shows a simplified view of OrbWork.

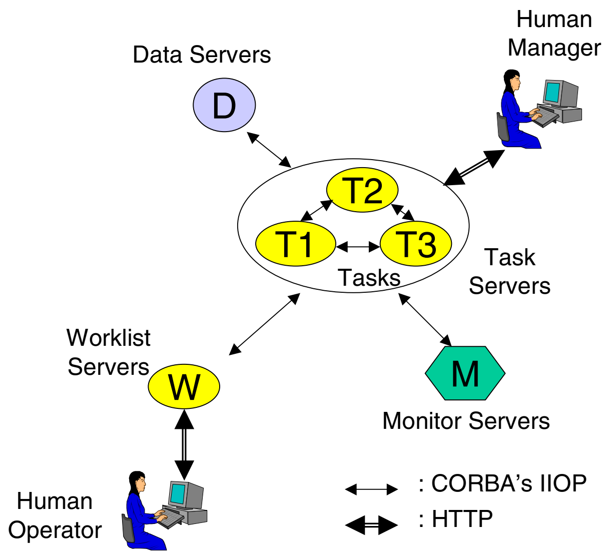

Figure 3. A simplified view of OrbWork

3. ACCESS CONTROL REQUIREMENTS FOR INTER-ORGANIZATIONAL WORKFLOW

There are many security requirements for inter-organizational workflows such as establishing secure communication among servers, and providing different views of the workflow based on users' needs-to-know and their affiliated organization. However, in this paper, we focus on access control requirements for inter-organizational workflows.

3.1. Separation of Application-level (Workflow) Security Infrastructure from Organization-level Security Infrastructure

Because there are several organizations that support an inter-organizational workflow, the participants may change during the life cycle of an inter-organizational workflow. For example, a new organization may replace an old organization or there may be a merger or separation among organizations. Since each organization may support several inter-organizational workflows, it is not realistic for each organization to restructure its security infrastructure for inter-organizational workflows. Therefore, inter-organizational workflows need to be insulated from organization level changes so that workflows can continuously operate without changing workflow specifications including security specifications.

3.2. Fine-grained and Context-based Access Control

Traditionally, an access control decision is made based on subjects and objects. The subjects may be users or processes acting on behalf of users. The objects are data or resources in the system; for example, objects may be files in the file system. Conventionally, a process, which may be an application executing on behalf of a user, is the finest grained subject for which an access control decision can be made by the operating system.

Inter-organizational workflows tend to be large scale and consist of many workflow tasks [Appendix], which can be threads within a process. Hence, conventional access control may be too coarse for workflows, in general. What we need is a fine-grained access control that is based on a user's working context. Workflow tasks provide users' working context. Even the same user may have different data access needs and requirements based on the tasks the user is working on.

3.3. Supporting Dynamic Constraints

Dynamic constraints are required in many inter-organizational workflows. Dynamic constraints may be based on the users of a specific task. For example, if a user performs a task, T1, then that person may not be allowed to perform another task, T2 (i.e., separation of duty [8]) in the same workflow instance. If there is one centralized WFMS, then it is not too difficult to enforce such constraints. However, inter-organizational workflow may consist of several autonomous workflows that work together to achieve an enterprise-level mission. Therefore, inter-organizational work-flows need some framework for sharing relevant execution history among participating workflows.

The above access control requirements are not only requirements for inter-organizational workflows but also those of enterprise applications that have to be executed across multiple organizations. In the following, we present access control related research.

4. RELATED RESERCH

Traditionally, an access control decision is made based on subjects and objects (see figure 4). The subjects may be users or processes acting on behalf of users. Conventionally, a process is viewed as a subject; however, for workflows, a process may include several workflow tasks, which can act on behalf of different users. The objects are data or resources in the system. For example, objects may be files in the file system. Permission is a set of authorized interactions that a subject can have with one or more objects in the system. Permission may have a variety of interpretations in various access control models. The basic idea is to control "*who can access which resources.*"

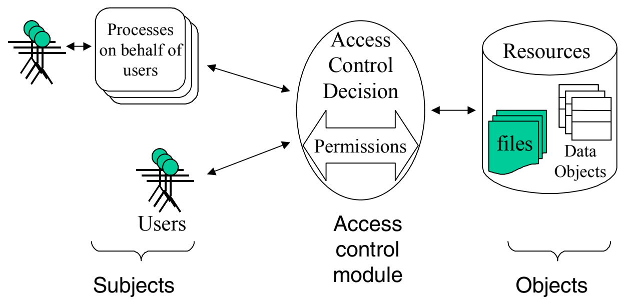

Figure 4. A traditional access control model

4.1. Role-based Access Control (RBAC)

Role-based access control (RBAC, [7]) has rapidly emerged in the 1990s as a technology for managing and enforcing security in large-scale systems. The basic notion of RBAC is that permissions are associated with roles, and users are assigned to appropriate roles. RBAC ensures that only authorized users are given access to certain data or resources. This simplifies security management and we can see that RBAC focuses on the management of subjects in figure 4 using users' roles instead of identities.

In RBAC, a role is a semantic construct forming the basis for access control policy. System administrators can create roles, grant permissions to those roles, and then assign users to the roles on the basis of their specific job responsibilities and policy. Therefore, role-permission relationships can be predefined, making it simple to assign users to the predefined roles. RBAC helps (especially, in a large enterprise system) to determine efficiently which permissions have been authorized for what users.

Constraints (e.g., separation of duties, [8]) can apply to relations and functions in an RBAC model. This is an effective mechanism for establishing higher-level organizational policy. Constraints are predicates, which are applied to the RBAC relations and functions and return a value of *acceptable* or *not acceptable*.

4.2. Task-based Authorization Controls (TBAC)

Task-based Authorization Controls (TBAC, [10]) is a task-oriented model for access control and authorization. It is an active security model that is well suited for distributed computing and dynamic information processing activities, such as workflow management and agent-based distributed computing.

TBAC focuses on security modeling and enforcement from the application and enterprise perspective rather than from a system-centric subject-object view. In the subject-object paradigm, the access decision function checks whether a subject has the required permissions for the operation, but it does not care about the contextual information about ongoing activities or tasks. In contrast, in the TBAC paradigm, permissions are checked-in and checked-out in a just-in-time fashion based on activities or tasks.

4.3. Fine-grained Object Approaches

Conventionally, a file or a data object is a unit of objects for which an access control decision can be made by the operating system. To provide fine-grained access control, permissions can be based on DTD[1] [5] or IDL[2] [9] for data objects. In other words, permissions can be based on specific fields or methods of data objects. For instance, NAI (Network Associates, Inc.) Lab's OO-DTE (Object Oriented Domain and Type Enforcement, [9]) provides relatively finer-grained access control than typical object oriented approaches. It can provide access control based on individual fields or methods of an object in CORBA-based systems. These approaches mainly focus on providing fine-grained access control to objects (figure 4). We refer to such approaches as fine-grained object approaches in this paper.

5. ACCESS CONTROL MECHANISMS FOR INTER-ORGANIZATIONAL WORKFLOWS

Inter-organizational workflows have to be executed on existing computing resources in participating organizations. They may be Windows-based, UNIX-based, LINUX-based, etc. Existing systems have their own security mechanisms. It is not realistic to expect participating organizations to change their computing resources or security mechanisms to support inter-organizational workflows. What is needed are access control mechanisms for workflow that can work with existing systems and security mechanisms. There are many ways to provide access control mechanisms for inter-organizational workflows. The access control mechanisms that we reviewed in section 4 are all applicable to inter-organizational workflows. The challenge is to satisfy access control requirements that we specified in section 3 without changing the existing security mechanisms and infrastructure of participating organizations.

From an organizational-level access control point of view, a workflow is an ordinary application program acting on behalf of users. For example, figure 5 shows an inter-organizational workflow that consists of two autonomous workflows. In this example, each autonomous workflow is just another application programs. Hence, each autonomous workflow should follow its organization's security policy. This implies that if we carefully hide additional access control mechanisms within the WFMS, it does not affect the existing access control mechanisms that were deployed by participating organizations. In this section, we highlight access control mechanisms that are incorporated in our WFMS, SALSA. Since these security mechanisms are managed by SALSA, existing organizations' security mechanisms are not affected.

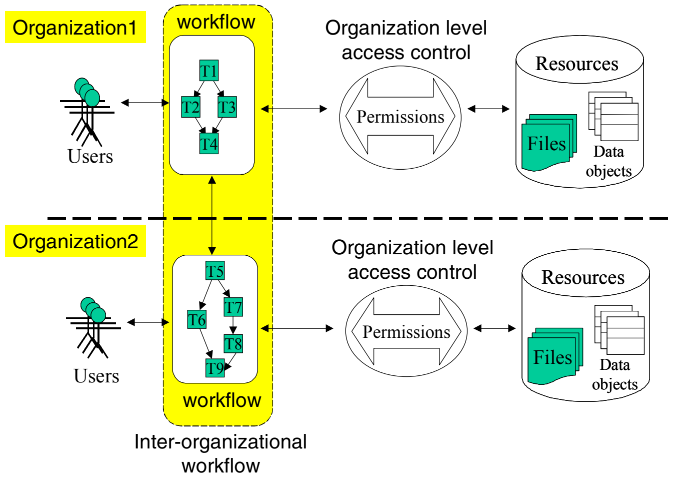

Figure 5. Inter-organizational workflow from conventional access control point of view.

5.1. Decoupling Workflow and Organization Security Infrastructures

In SALSA, there are two types of tasks: human and automatic tasks. Human operators accomplish human tasks and automatic tasks are accomplished by underlying components (e.g., database, executables). Hence, no human operators are needed for automatic tasks. RBAC is a convenient way for a system administrator to create roles, grant permissions to the roles, and assign users to the roles on the basis of their job responsibilities and the system policy. Therefore, in the SALSA implementation, we use RBAC for human tasks.

There are many organizations that can support an inter-organizational workflow. Hence, an inter-organizational workflow may have to interact with the security infrastructures of several participating organizations. If there is a change in participation, a part of the inter-organizational workflow has to be moved to other

[1]Document Type Definition that is used in Extensible Markup Language (XML).

[2]Interface Definition Language.

organizations, and the inter-organizational workflow specification (especially security specification) may have to be changed. We want to avoid changing inter-organizational workflow specifications, including security specifications (e.g., who can access a task). To avoid such disruptions, we need to decouple the workflow-specific security infrastructure from an organization's security infrastructures.

Generally speaking, an organization's role server contains organization-specific role structures that specify available roles, role hierarchy, and user-role assignments in the organization. If a workflow accesses the organization's role server directly, we cannot achieve this decoupling between the workflow security infrastructure and organization security infrastructure. To achieve this decoupling, we introduce a *role domain,* which is a role structure interface for workflow. Just like regular RBAC, each role domain contains specific roles and the relationships among them. Any organization that needs to participate in the cooperation should map its role structure to the role domain for the workflow. Because of this indirect interaction between a workflow and organizations' role structure, the changes in the participating organizations do not affect the workflow security infrastructure. Instead, these changes are confined to modification of the mapping from an organization's role structure to the role domain. In this sense, the relationship between a role domain and the role structures of organizations is similar to the relationship between an interface and a server implementation in client-server interactions. Role domains are interfaces for workflows, and each organization provides a mapping between the role domain and the organization's specific role structure. SALSA provides a role editor [4] for a workflow designer to define a role domain, roles in the domain, and the relationship among the roles.

When a workflow is designed, a workflow designer uses workflow domains, which were introduced in section 2, instead of organizations. The designer also specifies access control requirements in terms of role domains instead of organization-specific role structures. For example, an application designer may specify required roles for each task in the following way,

<Task *n,* Role>: [{roleDomain, (roles)}, {RD_1, (A, B, C, …)}, {RD_2, (X,Y, …)} …]

<Task *n,* Role> declares that this is a required role assignment for task *n.* In this example, RD_1 is a specific role domain and A,B,C,D are specific roles in RD_1. If a user belongs to one of the role domains in the required role set (expressed in […]) and has one of the roles in that role domain or more privilege than one of the roles in the required role set, he is allowed to perform the task.

The role assignment to each task during design time is turned into a security policy for each task that has to be enforced during runtime. When a user accesses a task during runtime, he presents a certificate[3] that reveals his identity and role in his organization. The OrbWork's Worklist server looks up the mapping between the role structure of the user's organization and that of the role domain. If the user has the proper role, he can execute the task and thus access the necessary resources.

[3] In our implementation, X.509 certificate is used to provide user ID and role/organization information. We use Phaos' JCA to generate certificates and Phaos' SSLava for SSL connection between Web browser and OrbWork's Worklist servers.

5.2. Fine-grained and Context-based Access Control

Consider a workflow that consists of 4 tasks: T1, T2, T3, and T4. Further, assume that all four tasks are in one process, and task T1 needs permission P1, task T2 needs permission P2, task T3 needs permission P3, and task T4 needs permission P4. Using traditional access control, even if a user needs only permission P1 to execute task T1, he will get P1, P2, P3, and P4 because conventional access control mechanisms cannot distinguish different tasks within a process.

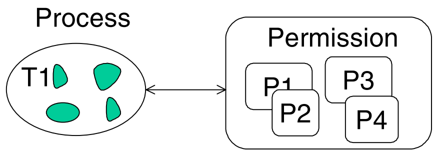

Figure 6. Traditional access control model

To support fine-grained access control in a workflow, we introduce task-specific access control modules (TACM). The purpose of the TACM is to provide fine-grained access control for both subject and object (see figure 2) in the following ways:

1. Divide a process (subject in figure 4) into many tasks,
2. Divide the data set (object in figure 4) that a workflow needs to access into many subsets. This is possible because the resources that each task needs to access may be a subset of the resources that the whole workflow needs to access, and
3. Provide access control between divided, thus smaller, subject and divided object.

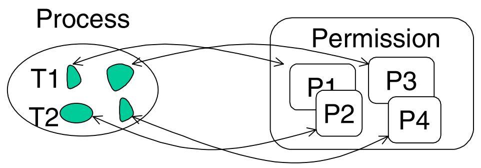

Figure 7. Fine-grained access control model

Thus, we make use of fine-grained object approaches that were discussed in section 4.3 at the task level rather than the process level. We provide a mechanism for a workflow designer to provide the task-data access specification that describes which fields of a data object can be accessed by a specific task. We provide a tool for a workflow designer to specify a data access policy for each task. In other words, each task has its associated data access policy that has a series of the following triples,

<Task *n,* Data>: [{Data object, field name, permission}, {…}, …]

<Task *n,* Data> declares that this is a data access control assignment for task *n*. The permission in this statement can either be read-only, full-control, no-access, etc. The task data access specification is also translated into a runtime specification for OrbWork to enforce. Any violation of the specification causes OrbWork to throw a data access exception.

In this approach, if a user has the correct required role, the user is allowed to access the task. However, data access by the user is further restricted by the task's context. Consider a workflow that consists of two tasks, Task1 and Task2. Assume that the following access control policy has been set:

<Task1, Role>:	[{GM, (Accountant)}, {Ford, (Accountant)}]
<Task1, Data>:	[{DataObj1, field1, read-only}, {DataObj2, field3, full-control}]
<Task2, Role>:	[{GM, (Manager)}, {Ford, (Project_leader)}]
<Task2, Data>:	[{DataObj2, field3, read-only}, {DataObj3, field2, read-only}, {DataObj3, field3, full-control}]

This means that only a user who has the Accountant role in GM or Ford role domains can execute Task1. When the user executes Task1, he has read-only permission on field1 in DataObj1 and full-control permission on field3 in DataObj2. Except field1 in DataObj1 and field3 in DataObj2, the user cannot read nor modify any other fields or data object.

In this sense, SALSA uses capability-based security. Figure 8 show that each task maintains data access capabilities. For example, task T2 can read fields f1 and f3 of data object D2, and write to field f2 of D2. In case of human tasks, a human operator must have a required role to access a task. Once the access to the task is granted, the operator can access only the portion of data that is in the task's capability list.

	D1			D2			D3		
	f1	**f2**	**f3**	**f1**	**f2**	**f3**	**f1**	**f2**	**f3**
T1	r		w				r	r	w
T2				r	w	r			
T3	r	w					r	r	r
T3				r	r	w			

capability

Figure 8. Fine-grained security for all tasks

It is interesting to compare the fine-grained and context-based access control that we introduced in this section to TBAC [10]. As we reviewed in section 4.2, TBAC activates and deactivates permissions in a just-in-time fashion based on the context associated with progressing tasks. If the TBAC is implemented in a centralized fashion (i.e., permissions are managed by a central access control module), it could introduce unnecessary constraints (e.g., race conditions) across workflows. In our SALSA implementation, permissions are managed in a distributed fashion; hence, it does not introduce unnecessary constraints across workflows (i.e., SALSA can enforce TBAC-like policies in a distributed fashion).

For example, if a user wants to execute a human task, the user must have the required role for the task. This is enforced by Access Control List (ACL) based security (see section 5.1 and step1 of figure 9). Note that subjects S1, S2, … are {roleDomain, role} pairs and T1, T2, … are tasks in step1 of figure 9.

Once a user is granted to access a task, the user's access to data objects is further restricted by the capability of the task that the user is accessing (step2 of figure 9). Hence, the task provides a context for the user's data access. Since the data access is restricted by the capability of each task, there is no need for TBAC style activation and deactivation of permissions in a just-in-time fashion based on the context associated with progressing tasks.

Step1:

	T1	T2	T3	T4
S1	x		x	
S2		x		
S3				x
S4		x	x	

Access to human tasks are enforced by access control list (ACL) based security

Step2:

	D1			D2			D3		
	f1	**f2**	**f3**	**f1**	**f2**	**f3**	**f1**	**f2**	**f3**
T1	r		w				r	r	w
T2				r	w	r			
T3	r	w					r	r	r
T3				r	r	w			

All tasks have capabilities (Capability-based security)

Figure 9. Two-step process for SALSA security in terms of access control matrices

5.3. Supporting Dynamic Constraints

Workflows sometimes require dynamic constraints such as dynamic separation of duty [8] (e.g., 2-man rule). Consider the following example: a simplified employee expense reimbursement scenario (see figure 10). This example consists of five tasks; four human tasks and one automatic task. We assume that a required role is associated with each human task. For simplicity, we also assume that all roles are from the same role domain. Any human operator who has a role that is in the required role set or has more privilege than any of the roles in the required role set can execute

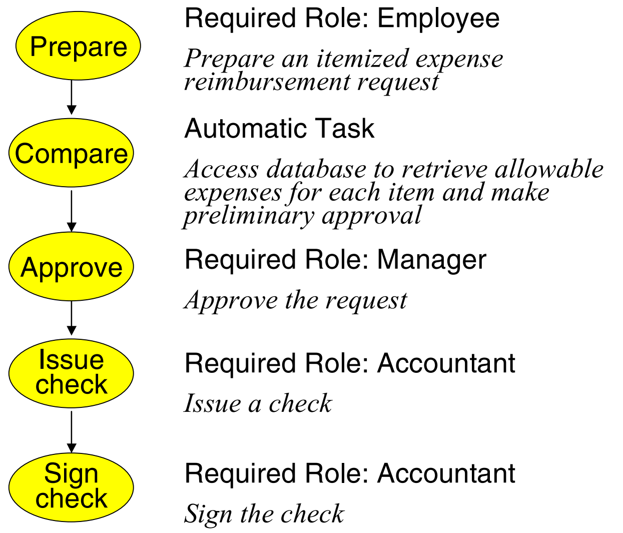
Prepare
Required Role: Employee
Prepare an itemized expense reimbursement request
Compare
Automatic Task
Access database to retrieve allowable expenses for each item and make preliminary approval
Approve
Required Role: Manager
Approve the request
Issue check
Required Role: Accountant
Issue a check
Sign check
Required Role: Accountant
Sign the check

Figure 10. An example of a simplified employee expense reimbursement process

the task. Note that the Issue_check and Sign_check tasks require the same role in this example.

Consider a scenario where an employee prepares an expense reimbursement request. An organization may want to enforce a security policy that specifies that the employee who prepared an expense reimbursement request should not approve the request. This is a general application of the traditional 2-man rule (i.e., separation of duty) that can be applied to two different tasks, Prepare and Approve, with two different required roles. In other words, if the employee, who initiates the reimbursement process, happens to be a manager, then the manager should not approve the expense reimbursement request that he initiated even though he has both Employee and Manager roles. We can apply the 2-man rule to the two other tasks, Issue_check and Sign_check that have the same required role, Accountant. In this case, the 2-man rule says that a person who issues a check should not sign the check. The Accountant role can be split into two roles, Accountant1 and Accountant2, and assign Accountant1 to Issue_check task and Accountant2 to Sign_check task with static separation of duty (i.e., users cannot be assigned to both Accountant1 and Accountant2). However, that is a solution that reduces the number of people who can perform the task.

To overcome these difficulties, we propose to use history-based access control. We have introduced the workflow monitor server in section 2. Since the monitor server keeps a log of execution history (e.g., who performs task A in workflow instance 5), a task that requires execution history to make access control decision can query the monitor server.

To implement this mechanism, we had to extend SALSA. The first aspect is design-time support. We introduce constraints for each task. Suppose the following constraint must be enforced: "task1 and task3 should not be executed by the same person for the same workflow instance." A workflow designer can specify dynamic constraints on task1 as !Performer(task3), which means that a user who performed task3 cannot perform this task, and !Performer(task1) for task3. If there is a dependency between task1 and task3, then only the task that is executed later may be constrained. This will generate a runtime specification that allows runtime tasks to query execution history from the monitor server.

The second aspect is runtime support. Inter-organizational workflows consist of several autonomous workflows. Hence, there may be many monitor servers. In SALSA, there is a monitor server per runtime engine. Therefore, we need some communication mechanisms that exchange relevant information among monitor servers. Each monitor server has its own database so that it can record events from OrbWork and answer any query from OrbWork or monitor clients. Monitor clients can register their topics of interests to monitor servers. For example, one monitor client may be interested in all events in a specific workflow while another monitor client may be interested in only events that have to do with a specific task. The monitor server records clients' interests and dispatches only those events that each client is interested in. The monitor server is not only a server but also a client, so that it can register its topics of interests to other monitor servers and receive interesting events from other monitor servers (figure 11).

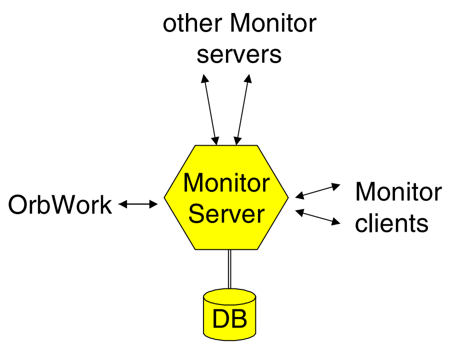

Figure 11. The structure of SALSA monitor server

Since each monitor server can act as a client we can arrange monitor servers in a hierarchical fashion. Consider a scenario where organization A is collaborating with organizations B, C, and D, and organization A is acting as a coordinator. Assume that each organization maintains its own workflow and monitor servers due to security and autonomy issues. Even though they manage their own workflows, organization A may need to know the status of the work in progress in organizations B, C, and D. In this case we can configure monitor servers so that organization A can receive specific events from organizations B, C, and D.

6. SUMMARY

In this paper, we described workflow-specific access control requirements such as dynamic constraints, fine-grained and context-based access control, and the need to insulate inter-organizational workflows from organization level changes. We presented a way to satisfy the above requirements. We have introduced the role domain as an interface between workflows and organization-specific security infrastructure. We also have introduced history-based access control for dynamic constraints, and fine-grained and context-based access control. Even though we introduced the access control mechanisms in the context of inter-organization workflow, they can be applied in other contexts, such as applications within a system or an organization. In our implementation of these mechanisms for SALSA, we carefully engineered the mechanisms so that existing security infrastructures of participating organizations are not affected and the autonomy of each organization is honored.

We can summarize the SALSA security architecture and its interaction with organizations' security mechanisms as follows (see figure 12). There are two kinds of access control modules in the overall security architecture:

– An organization-specific access control module (OACM) that is controlled by each organization and enforces a security policy that was set by each organization. The access control module in Figure 4 is an example of an organization-specific access control module.

– The task-specific access control module (TACM) that is controlled by each workflow and enforces task-specific security policies. Only a person with intimate knowledge of the workflow can set the security policy of each task because, in general, a task-specific security policy depends on the semantics of the workflow. This module enforces access control mechanisms that were introduced in section 5.2.

Since TACMs are distributed and autonomous, they cannot enforce constraints that depend on the activities of other tasks. To accommodate the needs for coordinating access control decision among tasks, we expand the capabilities of the workflow monitor. The monitor records workflow-specific events during runtime and responds to queries from the task-specific access control modules of the workflow.

TACMs need to support changes in participants because organizations that support a specific mission may be changed even before the mission is over. We can achieve this goal by reducing the dependency of task-specific access control modules on participating organizations' security infrastructure. The workflow security server (WSS) is a tool for achieving this goal. The workflow security server provides workflow-specific security infrastructure information (e.g., workflow-specific role domain) and the mapping information between the workflow-specific security infrastructure and the security infrastructures of participating organizations. When the participants change, we need to update the mapping between the workflow-specific security infrastructure and that of participating organizations.

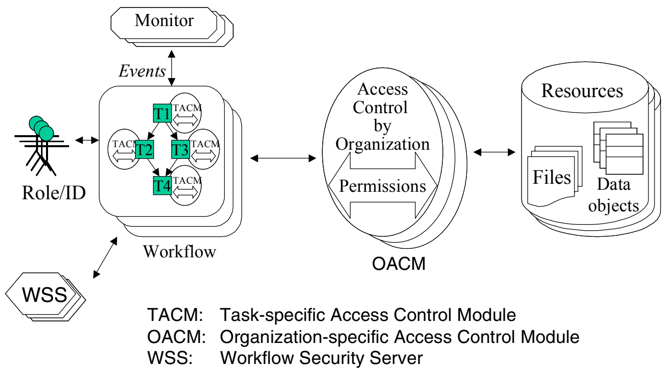
Monitor
Events
T1
TACM
T2
TACM
T3
TACM
T4
TACM
Role/ID
Workflow
WSS
Access Control by Organization
Permissions
OACM
Resources
Files
Data objects
TACM: Task-specific Access Control Module
OACM: Organization-specific Access Control Module
WSS: Workflow Security Server

Figure 12. The SALSA security architecture

Currently, we have completed implementation of 90% of all the mechanisms that we described in this paper. The remaining work is to modify OrbWork to enforce a history-based access control for dynamic constraints. We plan to release the SALSA design tools to the public at the first quarter of CY2001.

7. REFERENCES

[1] "Extensible Markup Language (XML) 1.0," World-wide-Web Consortium, http://www.w3.org/TR/1998/REC-xml-19980210.html.

[2] G.J. Holzmann, *The model checker Spin,* IEEE T/SE, Vol. 23, No. 5, May 97, pp. 279-295. See also http://cm.bell-labs.com/cm/cs/what/spin/

[3] M. H. Kang, J. N. Froscher, A. P. Sheth, K. J. Kochut, and J. A. Miller, "A Multilevel Secure Workflow Management System," Proceedings of the 11th Conference on Advanced Information Systems Engineering, Heidelberg, Germany (1999).

[4] M. H. Kang, B. J. Eppinger, and J. N. Froscher, "Tools to Support Secure Enterprise Computing," In Proceedings of 15th Annual Computer Security Applications Conference, Phoenix, Arizona, December 1999.

[5] D. L. Long, J. Baker, and F. Fung, "A Prototype Secure Workflow Server," In Proceedings of 15th Annual Computer Security Applications Conference, Phoenix, Arizona, December 1999.

[6] K. Kochut, A. Sheth, and J. Miller, "ORBWork: A CORBA-Based Fully Distributed, Scalable and Dynamic Workflow Enactment Service for METEOR," UGA-CS-TR-98-006, Technical Report, Department of Computer Science, University of Georgia, 1998.

[7] R. S. Sandhu, E. J, Coyne, H. Feinstein, and C. Youman, "Role-Based Access Control Models," IEEE Computer, 29(2): 38-47, February 1996.

[8] R. Simon and M. E. Zurko, "Separation of Duty in Role-Based Access Control Environments," In Proceedings of New Security Paradigms Workshop, September 1997.

[9] D. F. Sterne, G. W. Tally, C. D. McDonell, D. L. Sherman, D. L. Sames, and P. X. Pasturel, "Scalable Access Control for Distributed Object Systems," In Proceedings of 8th USENIX Security Symposium, Washington, DC, August 1999.

[10] R. K. Thomas and R. S. Sandhu, "Task-based Authorization Controls (TBAC): A Family of Models for Active and Enterprise-oriented Authorization Management," In Proceedings of the IFIP WG11.3 Workshop on Database Security, August 1997.

[11] H.M.W. Verbeek, T. Basten, and W.M.P. van der Aalst. *Diagnosing Workflow Processes using Woflan*. Computing Science Report 99/02, Eindhoven University of Technology, Eindhoven, 1999.

Appendix: An Inter-organization Workflow Model

In our inter-organization workflow model, a *task* represents an abstraction of an activity. A task can be regarded as a unit of work, which is performed by a variety of processing entities, depending on the nature of the task. There are two kinds of tasks: a network task and a simple task. A simple task can be performed by (*realized by*) a human, or by performing a computerized activity through executing a computer program, a database transaction, etc. A network task is performed by a network of interconnected tasks. Hence, a network task provides one level of abstraction (view) and its realization provides a lower level of abstraction (view). Since the realization of a task may contain many tasks at different levels of abstraction, a task is a recursive reference in the inter-organization workflow model. In this model, each task belongs to a workflow domain which may represents an organization or some other domain (e.g., security domain).

Figure A shows an inter-organization workflow (i.e., Task1 that is a top-level network task) that consists of three levels of abstractions (views). In Figure A, Task1 and Task 5 are network tasks that were realized by a network of tasks. Other tasks are simple tasks that can be realized by other means (e.g., human, database, executable). Transition T_j represents a transition from Task2 to Task3 (i.e., Task2 has been completed and Task3 can make use of results that were produced by Task2).

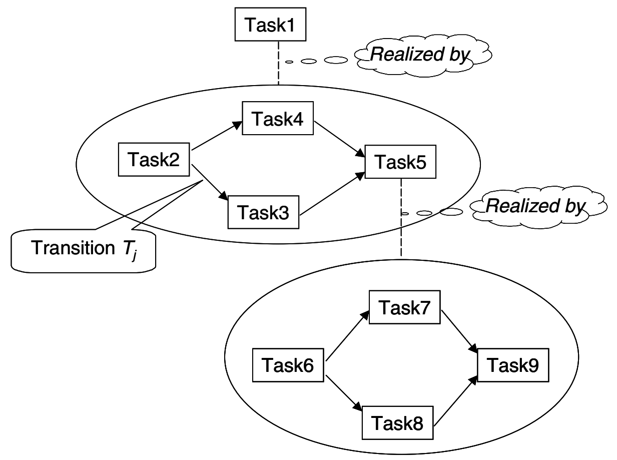

Figure A. An enterprise application model

A task may play the role of a source task or a destination task (e.g., Task2 is the source task and Task3 is the destination task of the transition t_j in Figure A) for a number of transitions. All of the transitions for which a task is the destination task are called the *input transitions* for that task (e.g., transition t_j is an input transition for Task3). Likewise, all of the transitions for which a task is the source task are called its *output transitions* (e.g., transition t_j is an output transition of Task2). A transition may have an associated Boolean condition, called its *guard*. A transition may be activated only if its guard is true.

The classes (i.e., types of objects) that are associated with an input transition to a task are called the task's *input classes*, and those appearing on an output transition are called *output classes* of that task. A task's output class, which is not its input class, is *created* by the task. A task's input class, which is not its output class, is *dropped (consumed)*. Note, that some input classes may be unused by the task. They are simply transferred to the task's successor(s).

A group of input transitions is called an *AND-join* if all of the participating transitions must be activated for the task to be *enabled* for execution. An AND-join is called *enabled* if all of its transitions have been activated. All the input transitions of a task may be partitioned into a number of AND-joins. A group of input transitions is called an *OR-join* if the activation of one of the participating transitions enables the task.

A group of transitions is said to have a *common source* if they have the same source task and all lead either from:

- its success state, or
- its fail state

A group of common source transitions may form either:

1. *AND-split*: Each of the transitions in the group has the condition set to `true`. It means that all of the transitions in the group are activated, once the task completes.
2. *OR-split* (selection): An ordered list of transitions where all but the last transition may have arbitrary conditions (i.e., the last transition on the list has the condition set to `true`). The first transition whose condition is satisfied will be activated.
3. *Loop*: A special case of an OR-split, where the list is composed of exactly two transitions: loopback and continue. Loopback implies branch taken and continue implies branch not taken (i.e., fall through).