A Prototype Secure Workflow Server

Douglas L. Long, Julie Baker, and Francis Fung
Odyssey Research Associates
33 Thornwood Dr., Suite 500
Ithaca, NY
{dougl,julie,fung}@oracorp.com

Abstract

Workflow systems provide automated support that enables organizations to efficiently and reliably move important data through their routine business processes. For some organizations, the information processed by their workflow systems is highly valued and in need of protection from disclosure or corruption. Current workflow systems do not help organizations to adequately protect this important data. We describe a prototype secure workflow system that allows users to develop high-level workflow security policies and to automatically execute these policies within the workflow system. These workflow policies can use the workflow context to provide fine-grained, dynamic access control and other security services that enhance the organization's ability to control the information contained in its workflow system. In this paper, we will explain these security policy goals, our prototype policy editor, our prototype workflow server, and our underlying Java-based implementation.

1 Overview

Odyssey Research Associates is developing secure workflow technology that will help the enterprise to secure and protect its sensitive data as the data moves through the enterprise's day-to-day activities. ORA approaches workflow security from a policy-based perspective. In this approach, high-level security policy defines the protection requirements for all types of enterprise data and the workflow infrastructure uses this policy to automatically determine the security requirements for properly handling data as it moves through the workflow system. The chief benefit of this approach is that the workflow system can dynamically control access to data as it moves from one workflow activity to another, providing access control that is tightly coupled to the activities that are being performed. This tight coupling ensures that workflow users have access to the data they need, but only when they are performing activities that require this access.

The next three sections of this paper describe the three primary goals for our secure workflow system: dynamic access control, fine-grained data security, and active security. The following two sections describe our approach to secure workflow and the Java-based infrastructure that we have developed to implement it.

This work is funded under a Phase II SBIR, "Security Policy Modeling and Enforcement Tools for Clinical Workflows," sponsored by DARPA.

2 Dynamic Access Control

One of the primary motivations of our work is the need to provide dynamic access control for certain types of enterprise data. The need for an enterprise user to access a particular piece of enterprise data may change over time due to changes in duties, changes in assignments, or for other reasons. A valid reason for access today may not be valid tomorrow. Dynamic access control automatically reacts to these changes, allowing access control decisions to take into consideration factors other than the usual subject, object, and permissions typical of standard access control models. Dynamic access control adds new dimensions to access decisions, considering not just who and what, but why.

There is another reason for using dynamic access control. The greater the sensitivity of an item of information, the more tightly access to that item needs to be controlled. Dynamic access control provides a finer granularity for access control decisions that provides more precise control over who can access the data and why. Dynamic access control provides tighter control over who can access which data, which increases the security for sensitive data.

It is important to note that dynamic access control does not replace other access control models and mechanisms. It instead allows these methods to be used more

effectively. Dynamic access control provides automatic policy-driven use of access control that allows the implementation of complex policies that cannot be effectively implemented manually.

ORA uses workflow as the basis for dynamic access control. Workflow defines processes and facilitates the assignment of the activities necessary to complete these processes. Each activity represents a distinct step in a process. The workflow system assigns specific activities to users based on the previously completed activities, the outcomes of those activities, the roles and responsibilities of the user, the process definition, and the state of the process instance. In other words, users perform work in the context of the workflow environment; this context provides much more information about why a user is performing a particular activity than is typically available. Since any data that a user needs to access to perform an activity is also accessed in the context of the workflow instance, this richer context is available for use in making access control decisions for this data. For example, users can be granted access to certain data while they are performing an activity that requires the access and denied access to the data when they are not performing a relevant activity. In this way, we can use the workflow context for making dynamic access control decisions.

Dynamic access control is not needed in every situation, but it is critical to providing adequate access control in some environments. Consider, for example, the clinical environment. A typical clinical environment may define roles, such as doctor, nurse, intern, clerk, etc., and use these roles for controlling access to medical records. While this scheme is sufficient to differentiate which users (roles) can read, write, and update which parts of a patient record, it does not provide the level of control over the patient record that is needed. For example, this scheme will allow any doctor to access any patient record at any time, regardless of whether or not the doctor is involved in the care of that patient. This inadequacy can lead to abuses: users can browse patient records to find personal information on celebrities, VIPs, friends, etc.; users can use this information in inappropriate ways; or users can even adversely affect the quality of patient care. The problem is that the need to access a patient's record is dynamic. Ideally, medical personnel should have no access to records of patients whom they are not treating, full access (appropriate to their role) to records of patients under their care, and limited access to records of patients whom they are no longer treating. Dynamic access control provides the necessary fine-grained access control and prevents the kinds of abuses cited above.

3 Fine-grained Data Security

Another primary motivation of our work is the need to provide fine-grained protection mechanisms for documents. Some parts of a document may be more sensitive than other parts and these parts may require stronger protection mechanisms than the less sensitive parts. On the other hand, the application of strong protection mechanisms to the entire document may unduly restrict access to the less sensitive parts of the document. Our solution to this problem is to provide for different levels of protection for different parts of a document. For example, some parts of the document may have access controls and other parts may be selectively encrypted. This solution enables organizations to balance the need for higher security with the need for access to data. Consider, once again, the clinical environment. An electronic patient record can be viewed as a single document that contains information with a range of sensitivities. Based on their role, different medical personnel require access to different parts of this document; some of these personnel require access to sensitive data, others do not. Our approach treats a patient record as a single integral document, but protects the different parts of the document according to the sensitivity of the data contained there.

For this project, we have developed a security policy editor that assigns security policy to the different parts of a document, based on the structure of the document. Any document (or as we shall see later, data source) that has sufficient structure can have policy assigned to its parts in this manner. The security policy editor uses XML Data Type Definitions (DTD) to describe the structure of a class of documents, allows the user to assign policy to the different elements in this structure, and outputs the resulting policy. This policy is used by the enforcement mechanism described in section 6 to control the security of all documents that are instances of this class. (Note that, for the current prototype, the structure of the documents that we work with is restricted to a particular hierarchical form; we have yet to consider arbitrary DTDs.)

4 Active Security

The final motivation of our work is the need to provide an active security environment that automatically implements the desired security policy with little or no intervention from end users. There are a number of reasons that end users cannot be completely depended on for policy implementation. End users are notorious for not wanting security to get in the way. End users are not always sufficiently educated about security and can often inadvertently bypass or disable security functionality without being aware they have done so. End users do not always remember to do the right thing. Finally, end users need assistance with complicated policies; policies that require users to take frequent manual actions are doomed to failure. Our approach to solving these problems is to

build the policy into the workflow infrastructure. By making the workflow infrastructure responsible for security policy interpretation and implementation, we provide the end user with automated support that relieves the end user of much of the drudgery.

Consider the clinical environment once again. Medical end users require immediate access to medical records. This means that medical end users will not tolerate any delay in access to medical records due to security. For these reasons, security in clinical environments must be almost entirely transparent to the end user. In this environment, active security is essential to protecting patient privacy.

5 Secure Workflow

Odyssey Research Associates has developed a secure workflow system that protects enterprise data according to a workflow security policy. We have also developed a security policy editor that helps security administrators to define the workflow security policy. The secure workflow system consists of two types of servers: a workflow server and a data repository. As currently implemented, multiple data repositories may be used to support a single workflow server. The workflow server is responsible for all workflow-related functions: keeping track of process status, assigning activities, etc. Data repositories are responsible for the storage and retrieval of the data that workflow users require to perform assigned work activities. Workflow clients can only interact with the servers through protected interfaces. These protected interfaces are security aware in the sense that all actions performed using a protected interface must conform to the workflow security policy.

The design of the secure workflow server incorporates an off-the-shelf workflow engine that provides basic workflow services; we augment the basic workflow services by wrapping them with protected interfaces as described above. In order to use an off-the-shelf workflow engine effectively to build our secure workflow server, we require a standard object model for workflow services that we can use in conjunction with any off-the-shelf workflow engine. Fortunately, the Object Management Group (OMG) and the Workflow Management Coalition (WFMC) have joined forces to provide just such a standard model [5]. The OMG, supported by the WFMC, has adopted a workflow management facility. The OMG standard, originally known as the jFlow Workflow Management Facility, is co-authored by representatives of 20 companies and supported by 20 others, which altogether represent the workflow industry's major players. By adopting the jFlow standard, enhancing it with our policy enforcement mechanisms, and using a commercial workflow engine as a back-end server, we have been able to develop a secure workflow server with a significantly smaller investment of time and resources than would otherwise be possible.

The design of the data repository is also based on industry standards. We use XML DTDs to define the structure of the repository data. The interfaces that clients use to access repository data are based on the Document Object Model (DOM), which is an object model developed by the World Wide Web Consortium for representing XML documents and data [4]. As is the case with the workflow server, we have wrapped the DOM interfaces with protected interfaces that ensure that clients perform all data storage and retrieval actions in accordance with the workflow security policy. By using XML as the basis for the data repository, we can easily place any data that can be generated in an XML format into the data repository.

We also use XML DTDs to define the structure of the workflow data for the security policy editor. This allows us to tie the security policy to specific element types, which, in turn, allows a fine-grained specification of the security policy.

Figure 1 shows how all of these components fit together to provide a secure workflow system.

Note that policy creation, class generation, and placing of the initial set of repository data onto the data repository does not require access by workflow users and can therefore be done in a controlled environment. Once an administrator has initialized the repository through this administrative process, workflow users are able to access the data through the protected interfaces.

6 Active Security Infrastructure

To implement the protected interfaces we have developed a distributed, object-oriented, policy enforcement infrastructure. We also have developed tools that help implementers to build applications based on this infrastructure.

The distributed policy enforcement mechanism includes not only the component or components that store and adjudicate the security policy, but also those components that connect the security policy components with the rest of the system and ensure that the appropriate security policies are properly enforced. We take an object-oriented approach to the design of the security services and policy enforcement mechanisms. We provide the tools for developing a set of policy-aware objects that are capable of enforcing security policies developed using the security policy editor. We have developed an approach that makes it possible to add our policy enforcement mechanisms to existing sets of interfaces, with minimal impact on these interfaces. This

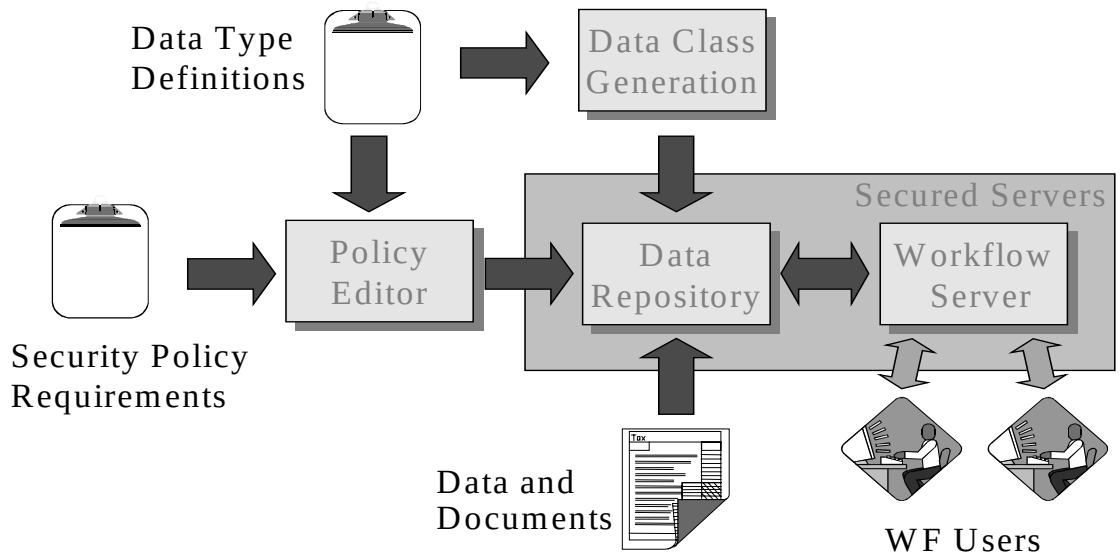

Figure 1. Secure Workflow and Data Repository

in turn allows us to take advantage of object models and standards that industry groups and other standards bodies are currently developing

The distributed policy enforcement mechanism includes not only the component or components that store and adjudicate the security policy, but also those components that connect the security policy components with the rest of the system and ensure that the appropriate security policies are properly enforced. We take an object-oriented approach to the design of the security services and policy enforcement mechanisms. We provide the tools for developing a set of policy-aware objects that are capable of enforcing security policies developed using the security policy editor. We have developed an approach that makes it possible to add our policy enforcement mechanisms to existing sets of interfaces, with minimal impact on these interfaces. This in turn allows us to take advantage of object models and standards that industry groups and other standards bodies are currently developing.

We considered several options, including Microsoft's COM, OMG's CORBA, and Java Remote Method Invocation (RMI) for implementing the secure workflow server. We ultimately decided to use Java RMI technology. We rejected COM because of the restrictions of COM technology to the Windows/Intel architecture. At the other extreme, we decided that we did not need the full power (and expense) of CORBA and its ability to integrate heterogeneous programming environments, languages, and legacy systems. The Java environment and Java RMI provide an elegant development environment (along with cross-platform portability) without the expense and overhead of a CORBA environment. However, we have purposely kept our object models and overall design very CORBA-like to maintain the potential for CORBA integration at some time in the future.

In our prototype workflow system, we protect objects by placing their implementations on secured servers and providing protected interfaces that provide and enforce secure access to server objects. Clients access server objects through proxy objects whose purpose is to enforce client-side security policy. A corresponding object on the server side implements the secure interface to the server object and enforces server-side security policy. Our security policy enforcement mechanism is layered on top of Java RMI in a way that transparently provides server objects with the security protection the policy requires. With our design, we can automatically generate all the code necessary for security policy enforcement directly from the specification of the Java interface to the server object implementation. The server object implementation does not require any modification in order to be secured. At runtime,

the security policy is instantiated on the server and securely downloaded to the client proxy.

Part of our approach is to automatically generate all the code necessary for security policy enforcement directly from a Java interface's specification. From a Java interface definition, we generate a set of security-aware Java interfaces and classes. The generated interfaces and classes transform the original Java interface into a secure, remote interface capable of enforcing the security policy defined for the interface. Security policy enforcement is largely transparent to clients that use the transformed interface and to servers that provide implementations of the interfaces. We are using Spectangle, a tool developed by ORA as part of the CORBA/THETA project for the NSA [3], for the code generation. Spectangle provides a powerful, flexible code generation facility that is ideal for this project and allows us to automatically generate the security code for a Java interface in a cost-effective manner.

7 Conclusion

ORA has developed a prototype secure workflow server that automatically implements an enterprise security policy for workflow users and activities. We have developed a security policy editor that security administrators can use to develop appropriate workflow policies and we have developed a secure workflow server and secure data repository that enforce these policies for all interactions with workflow client software. In addition, we have implemented the servers using a distributed object-oriented policy enforcement infrastructure. This project has demonstrated how high-level security policies can be implemented in a workflow system, providing enhanced security services that protect sensitive data as it is moved through the system.

References

[1]. Roshan Thomas, "Security in Workflow Processes: Dynamic Security Models for Clinical Workflows," Odyssey Research Associates, December 4th, 1996.

[2] Douglas Long, Peter Samsel, "Security Policy Modeling and Enforcement Tools for Clinical Workflows, Semi-Annual Report," Odyssey Research Associates TM-98-0016, June 15, 1998.

[3] Maureen Stillman, Cheryl Barbasch and Matthew Stillerman, "Design of a CORBA-Compliant THETA," Odyssey Research Associates TM96-0026, March 1997.

[4] W3C, "Document Object Model (DOM) Level 1 Specification," World Wide Web Consortium October 1, 1998.

[5] OMG, "Joint Workflow Management Facility - Revised Submission," Object Management Group bom/98-06-07, July 4, 1998.

[6] Charles F. Goldfarb, Paul Prescod, The XML Handbook. 1998, Upper Saddle River, NJ: Prentice Hall PTR.

[7] Carla Marceau, "Specification-based code generation using literate programming tools," Odyssey Research Associates, December 5, 1996.