SecureFlow: A Secure Web-enabled Workflow Management System

Wei-Kuang Huang*
Department of Operation and Information Management
University of Connecticut
368 Fairfield Road, Storrs CT 06269
whuang@sba.uconn.edu

Vijayalakshmi Atluri†
MSIS Department and
Center for Information Management,
Integration and Connectivity (CIMIC)
Rutgers University
180 University Avenue, Newark NJ 07102
atluri@andromeda.rutgers.edu

Abstract

The objective of this paper is to present a web-based Workflow Management System (WFMS), called *SecureFlow* that can serve as a framework for specification and enforcement of complex security policies within a workflow, such as separation of duties. The main advantage of SecureFlow is that it uses a simple 4GL language such as SQL to specify authorization constraints, thereby improving flexibility and user-friendliness. Due to the modular nature of the SecureFlow architecture, the security specification and enforcement modules can be layered on top of existing workflow systems that do not provide adequate support for security. SecureFlow relies on the *Workflow Authorization Model* (WAM) recently proposed by Atluri and Huang.

1 Introduction

Since timely services are critical for any business, there is a great need to automate or reengineer the business processes. Typically many organizations achieve this by executing these coordinated activities (tasks) that constitute the business process (workflow) through workflow management systems (WFMS). Today, with the explosion of the Internet technologies, the demand on business process management has raised to a new level. Web and workflow management systems together serve as an ideal combination to integrate the distributed processes that are across or within enterprise boundaries. This is because, first Web is already globally distributed, robust and reliable communication mechanisms are already in place, web browsers are commonplace, and web servers can interact with the databases via CGI programs to store, retrieve, and route data. Second, with the global naming scheme for resources (e.g. URLs), protocols for accessing named resources (e.g., HTTP), HTML, Javascript and some other emerging Web technologies such as eXtensible Markup Language (XML) and Java, all together provide an ideal combination for the development of client-server collaborative workspace.

Such web-enabled workflow systems can serve effectively for more dynamic internet-based business processes over heterogeneous computing platforms. Some examples of such processes include global supply chain management, universal telecommunication service management, and mobile patient care service management.

The Internet-based workflow systems have drawn a lot of attention recently ([7],[15]), as they offer a number of advantages. First, they support better mobility by allowing users to access workflow systems from virtually any computer connected to the Internet with a standard web browser. Also, by using Java applets, the user can acquire the program that performs the task on demand without prior installation on user machine. Second, they are extremely scalable. Due to this, companies are anticipating to use the public Internet as a vehicle to conduct business-to-business transactions such as electronic commerce.

There are a number of prototypes using web technology as a major user interface. Mobile [5] uses web for building user interface and integrating external applications. Panta Rhei [8] primarily uses web as the messaging tool to exchange forms between users. WASA [16], constructed in Java, supports dynamic modification of Workflow specification. Commercial products that fall into this category include IBM FlowMark [10] and HP Changengine [15], etc.

The desired security to ensure the secrecy, correctness and integrity of a business process is specified through a set of security policies. These high level policies state which user is authorized to execute a specific task within the business process (or workflow). To simplify the security administration, they are typically

*The work of W.-K. Huang was supported by the summer grant from the department of Operation and Information Management (OPIM), University of Connecticut.

†The work of V. Atluri was supported in part by the National Science Foundation under grant IRI-9624222.

specified on roles rather than on specific users, and users are in-turn assigned to one or more roles. An example of such a policy may be "the paper submitted to a research conference can be reviewed only by a member of its program committee." This simple role-based access control [14] may not be adequate for expressing many business policies. An example of such policy is "none of the authors of the paper is eligible to review a paper, even though the author is a program committee member." These policies, also known as separation of duties [6, 13] should be specified as additional constraints.

Most of the existing systems provide minimal security features such as user authentication. Although some commercial WFMS such as FlowMark, Notes and Changengine can support role-based access control, they do not provide support to specify and enforce separation of duties constraints. They have to be implemented in an ad-hoc manner through a script type language. Such ad-hoc implementation makes analysis and maintenance of security policies more difficult. In addition, many efforts in WFMS implementation have been placed in protecting data transmitted over the network, little emphasis has been given on providing access control for workflow activities.

Karlapalem and Huang [12] have raised some important security issues in activity management system including role management, support of separation of duties constraints, dynamic assignment of access privileges along with activity execution and inference control of information. Their system, named as CapBasED-AMS, provides a general architecture for incorporating discretionary access control, mandatory access control, event based constraints to the activity management system. However, the system does not provide capability to handle separation of duties constraints nor be able to synchronize the propagation of the authorization with the workflow execution.

Since existing authorization models developed for Database Management Systems (DBMS) are not adequate for WFMS, in [2], Atluri and Huang have proposed an authorization model suitable for workflows, called *Workflow Authorization Model* (WAM). Later, in [3], they have enhanced WAM to incorporate separation of duties constraints. This paper presents a web-based Workflow Management System (WFMS), called *SecureFlow* that can serve as a framework for specification and enforcement of complex security policies within a workflow such as separation of duties. The main advantage of SecureFlow is that it uses a simple 4GL language such as SQL to specify authorization constraints, thereby improving flexibility and user-friendliness. Due to the modular nature of the SecureFlow, the security specification and enforcement modules can be layered on top of existing workflow systems targeted for Internet, Intranet and traditional distributed processing applications that do not provide adequate support for security.

SecureFlow is based on WAM. WAM ensures that the tasks that constitute the workflow are executed only by authorized users or processes (subjects), and makes sure that authorized subjects gain access on the required objects only during the execution of the task. This is achieved by granting and revoking of privileges in synchronization with the progression of the workflow through proper authorization mechanisms. In WAM, authorization specification is done by way of specifying authorization templates. The actual authorization, i.e., the triple (subject,object,privilege) is derived only during the execution of the task to which the authorization template is associated. That is, a subject assigned to a task of a particular workflow instance is carried out by executing an SQL statement since authorization repositories are stored as a simple relational database. This enables us not only to specify complex security policies such as separation of duties and voting-based authorization [11], but also allows to effectively manage large policy bases.

Due to the modular structure of the SecureFlow architecture, the workflow authorization module, called the workflow authorization server (WAS), can be separated from the entire workflow system. WAS is responsible for the authentication of subjects as well as to specify and administer authorization constraints. The workflow specification module and the workflow execution server allow remote users from various sites to specify the workflow processes and execute the workflow from their local web browser, respectively. Automatic workflow execution where examination of the preconditions and triggering subsequent tasks can all be incorporated in the workflow execution server. Thus, the use of web enables execution and security administration of workflows running in a loosely coupled heterogeneous autonomous distributed environment through these centralized servers.

This paper is organized as follows. In section 2, we review the workflow authorization model proposed by Atluri and Huang in [2, 3]. In section 3, we present the architecture of SecureFlow. In section 4, we show how complex security policies in workflow systems can be specified and enforced using simple SQL statements. In section 5, we present the implementation details of SecureFlow and report its current status. Finally, in section 6, we present the conclusions.

2 Workflow Authorization Model

In this section, we review the *Workflow Authorization Model* (WAM) proposed in [2, 3]. A workflow deals with coordinated execution of tasks that involve processing of relevant objects by subjects (either humans or

programs). To execute a task, relevant privileges on required objects have to be granted to appropriate subjects. WAM *dynamically* delegates authorizations to support workflow activities in a way that the time interval associated with the required authorization to perform a task changes according to the time during which the task actually executes. WAM uses the notion of an *Authorization Template* (AT) which specifies the static parameters of the authorization that can be defined during the design of the workflow. ATs are attached to tasks. When the task starts execution, its AT(s) is used to derive the actual authorization. When the task finishes, the authorization is revoked. In the following, we briefly review WAM.

Let $S = \{s_1, s_2 \ldots\}$ denote the set of subjects, $O = \{o_1, o_2 \ldots\}$ the set of objects, $\Gamma = \{\gamma_1, \gamma_2 \ldots\}$ the set of objects types and $R = \{r_1, r_2 \ldots\}$ the set of roles. The function $F : O \rightarrow \Gamma$. That is, if $F(o_i) = \gamma_j$, then o_i is of type γ_j. $G : S \rightarrow R$. I.e., if $G(s_i) = r_j$, then s_i is of role r_j. Let PR denote a finite set of privileges. S_{r_i} denotes the set of subjects that belong to role r_i, and O_{γ_i} the set of objects of type γ_i.

Definition 2.1 An authorization is a 4-tuple $A = (s, o, pr, [\tau_b, \tau_e])$, where subject s is granted access on object o with privilege pr at time τ_b and is revoked at time τ_e. □

An authorization base $AB = \{A_1, A_2 \ldots\}$ is a finite set of authorizations. As workflow execution progresses, all authorizations that have been generated along with the execution are added to the set AB.

Definition 2.2 Given a task tw_i, an authorization template $AT(tw_i)$ is defined as a 4-tuple $AT(tw_i) = ((r_i, -), (\gamma_i, -), pr_i, [\tau_{l_i}, \tau_{u_i}])$ where
(i) $(r_i, -)$ is a *subject hole* which can be filled by a subject s_i where $G(s_i) = r_i$,
(ii) $(\gamma_i, -)$ is an *object hole* which can be filled by an object o_i where $F(o_i) = \gamma_i$,
(iii) pr_i is the privilege to be granted to s_i on object o_i.
(iv) $[\tau_{li}, \tau_{ui}]$ is the time interval during which the task must be executed. □

In the definition for $AT(tw_i)$ (i) says that only subject belonging to role r_i or dominated by r_i is allowed to execute task tw_i thus the subject hole $(r_i, -)$ allows only subjects that belong to role r_i, (ii) dictates that only objects of type γ_i or subtype of γ_i can be processed by tw_i thus the object hole $(\gamma_i, -)$ allows objects of only type γ_i to be filled in, (iii) says that a subject requires a privilege pr_i on the objects that arrive at tw_i for processing and (iv) says the default interval for the authorization template will be the valid time interval for the task.

Authorization templates are attached to the tasks in a workflow. A new authorization is granted to a subject on the specified object if the object's type is same as that specified in the template. A task may have more than one authorization template associated with it. More ATs are required in cases where there is more than one type of object to be processed, or more than one subject is required to perform the processing.

An authorization template enables one to specify rules such as "Only a clerk is allowed to perform check preparation during time 10 and 50." These can actually be stated during the design process by the workflow designer. However, before workflow starts, no authorization is derived. The actual authorization is granted to a particular clerk only when the preparation of check actually starts. When the task finishes, the authorization will expire.

WAM with Separation of Duty Constraints

Separation of duties can be expressed as constraints or rules. In [4], Bertino et al. have identified several types of authorization constraints, including separation of duties. For the sake of simplicity, in this paper, we assume each constraint is a logical expression of the form: $q \leftarrow p$ where q is a single literal since most separation of durties are of this form. We identify the following two basic groups of constraints: assertive and exclusive.

Definition 2.3 Given an authorization template $AT(tw_i) = ((r_i, -), (\gamma_i, -), pr_i, [\tau_{l_i}, \tau_{u_i}])$, we define a set of *potential authorizations*, PA_i, representing all possible authorizations that can be potentially derived from $AT(tw_i)$. Each potential authorization pa in PA_i is a triple (s_i, o_i, pr_i) such that $s_i \in S_{r_i}, o_i \in O_{\gamma_i}$. □

Definition 2.4 Given an authorization $A = (s, o, pr, [\tau_b, \tau_e])$ in AB, we define a *non-temporal n projection* A_{NT} of A as $A_{NT} = (s, o, pr)$. The non-temporal projection of AB, $AB_{NT} = \{A_{NT_1}, A_{NT_2} \ldots\}$. □

In our formalism, each constraint c_i is a logical expression of the form: $q \leftarrow p$ where p is any logical expression consisting of A_{NT} as literals and q is a single literal which is either pa or $\sim pa$ such that $pa \in PA_i$ of some tw_i. $s(p)$ (or $s(q)$) denotes the set of subjects that are specified in $pa \in PA_j$ (or $pa \in PA_i$).

Assertive and exclusive type constraints are expressed as the following rules [9, 3].

- *Exclusive type*: In this type, q is always of the form $\sim pa$ where $pa \in PA_j$ for some tw_j.
- *Assertive type*: In this type, q is always of the form pa where $pa \in PA_j$ for some tw_j.

The set of eligible users for each task changes dynamically based on the current state of the authorization base. For example, based on whether a constraint is

either assertive or exclusive, certain users in the role of clerks are not eligible to execute the task of issuing a check. Moreover, the eligible users vary from one task-instance to another.

Furthermore, only certain constraints play a role in deciding the eligibility of subject to execute a task. Therefore we determine the set of relevant constraints for each task, denoted as C_{tw_i}.

Definition 2.5 In the case of multiple constraints on a task, we define C_{tw_i} as follows: $C_{tw_i} = \{c_j \mid c_j \in C$ which is of the form $q_i \leftarrow p_j$ and $q_i \in PA_i\}$. □

For each task tw_i, we define a set of eligible subjects, denoted as $S_i^e(o)$ with respect to object o.

Definition 2.6 Given an authorization template $AT(tw_i)((r_i, -), (\gamma_i, -), pr_i, [\tau_{l_i}, \tau_{u_i}])$, we define the set of eligible subjects $S_i^e(o)$ as follows:

1. $S_i^e(o) = S_{r_i}$, if $C_{tw_i} = \emptyset$;
2. $S_i^e(o) = S_1 \cap S_2 \cap \ldots S_n$, where each $S_k = S_{r_i} - s(q_i)$, if $c_k : q_i \leftarrow p_j \in C_{tw_i}$ is an exclusive constraint and p_j is true with respect to AB_{NT}, and $S_k = s(q_i)$, if $c_k : q_i \leftarrow p_j \in C_{tw_i}$ is an assertive constraint and p_j is true with respect to AB_{NT}. □

The above definition says that if the constraint specifying the separation of duties is of exclusive type, the set of eligible subjects is obtained by subtracting the disallowed subject from the set of subjects playing the role assigned to execute the task. On the other hand, if the constraint is of assertive type, the set of eligible subjects is simply the set of subjects specified in $pa \in q$. If no constraints affect the task, then the set of eligible subjects is same as the set of subjects playing the role. In the case where multiple constraints are imposed on a task, the net Eligible Subject Set (ESS) is the intersection of all the eligible subject sets derived from each constraint c_i.

An appropriate authorization must be generated from the authorization template at run time in such a way that the subject to execute the task must be chosen from the set of eligible subjects and only when the task receives an object with type specified in the authorization template. The following authorization derivation rule ensures this.

Definition 2.7 [Authorization Derivation Rule for extended WAM] Given an authorization template $AT(tw_i)$ $= ((r_i, -)(\gamma_i, -), pr_i, [\tau_{li}, \tau_{ui}])$ of task tw_i, an authorization $A_i = (s_i, o_i, pr_i, [\tau_{b_i}, \tau_{e_i}])$ is derived as follows:

Grant Rule: Suppose object x is sent to subject y at τ_{a_i} to start tw_i.

If $x \in O_{\gamma i}$ and $y \in S_i^e(x)$ and $\tau_{a_i} \leq \tau_{u_i}$,
$s_i \leftarrow y, o_i \leftarrow x, pr_i \leftarrow pr(AT)$;
$\tau_{e_i} \leftarrow \tau_{u_i}$,
if $\tau_{a_i} \leq \tau_{l_i}$,
$\tau_{b_i} \leftarrow \tau_{l_i}$;
otherwise $\tau_{b_i} \leftarrow \tau_{a_i}$.

Revoke Rule: Suppose w_i ends at τ_{f_i} at which point o_i leaves tw_i.

If $\tau_{f_i} \leq \tau_{u_i}$,
$\tau_{e_i} \leftarrow \tau_{f_i}$. □

Without the separation of duties constraints, a subject is chosen from the set of subjects playing the specified role, whereas with separation of duties constraints, a subject is chosen from the set of *eligible* subjects which may be a subset of the previous set.

More details on WAM can be found in [3]. In the following, we explain the process of deriving authorizations by taking an example.

Example 2.1 Consider a workflow that deals with check and purchasing processing which consists of four tasks including preparation a *request* (tw_1), approval of a *request* (tw_2), issuing a *request* (tw_3) and voiding a *request* (tw_4). Assuming there is time specification to be associated with each task indicating the valid time interval during which the task can be performed, say [10,50] for tw_1, [20,60] for tw_2 and [40,80] for tw_3 and tw_4. Two object types are processed in the workflow: check and purchase request. For brevity, we use *request* to represent any of the two types. Suppose the security policies specify that any individual belonging to the role *clerk* is authorized to prepare, issue and void a *request* and only a *manager* can approve a *request*. To initiate the workflow, a clerk is given the privilege to prepare a *request*. The prepared *request* is then forwarded to a manager. The manager is given the privilege to approve or disapprove the *request*. When the approval task finishes, the decision as to whether the *request* will be issued or voided is based on the result of the approval. i.e., check/purchase order will be issued if it is approved, otherwise it will be forward to the request void task. Suppose there are another two constraints in business policy restricts that no any single individual is allowed to both prepare and issue a check and a purchase order has to be approved by two managers belonging to different departments. Obviously, the former is an exclusive type constraint whereas the latter is an assertive one. In the following, we show the required authorization templates and constraints associated with each task:

$AT_1(tw_1)$ = ((clerk,-), (check,-), prepare,[10,50])
$AT_2(tw_1)$ = ((clerk,-), (purchase_request,-), prepare, [10,50])
$AT_1(tw_2)$= ((manager,-), (check,-), approve, [20,60])
$AT_2(tw_2)$= ((manager,-), (purchase_request,-), approve, [20,60])
$AT_1(tw_3)$= ((clerk,-), (check,-), issue, [40,80])

$AT_2(tw_3)$= ((clerk,-), (purchase_request,-), issue, [40,80])
$AT_1(tw_4)$= ((clerk,-), (check,-), void, [40,80])
$AT_2(tw_4)$= ((clerk,-), (purchase_request,-), void, [40, 80])
$C_{tw_3} = \{c_1\}$ and $C_{tw_2} = \{c_2\}$ where
c_1: $(\forall x \in S_{clerk}, y \in O_{check}) \cdot (\sim (x, y, issue) \leftarrow (x, y, prepare))$
c_2: $(\forall x1, x2 \in S_{manager}, y \in O_{purchase\_request}) \cdot ((x2, y, approve) \leftarrow (x1, y, approve) \cdot x1.department <> x2.department)$

□

3 SecureFlow Architecture

The architecture of SecureFlow is comprised of four major components, namely, workflow design/specification module, workflow execution server, workflow authorization server and workflow client. Figure 1 depicts the architecture of SecureFlow.

3.1 System Components

Workflow Design/Specification Module (WDSM): This module provides an interface for specifying the workflow including tasks and dependencies. To specify a workflow, the user logins to the workflow design and specification module and selects the workflow to be defined. For defining a task, the user is required to input the name of the task, the interval during which it can be executed, and the location of the client program that is invoked for executing the task. A task ID is automatically generated at the completion of task specification. For specifying the dependencies among tasks, the child (activated) task and the parent (activating) task need to be specified. The system supports a number of dependency types, namely begin on commit, abort, begin on abort, begin, exclusive, etc. (Refer to [1] for a definition of various types of task dependencies.) This information is maintained in three relations, WORKFLOW, TASK and DEPENDENCY. (The corresponding database schema can be found in section 3.3.)

Workflow Execution Server (WES): This module schedules the submission of execution request of the task based on the dependency requirement. The Workflow Execution Interface provides workflow users access to workflow tasks. To execute a task, the user first logins to the workflow execution server and selects the task to be performed. WES then consults with WAS (described below) and determines whether the user possesses appropriate authorization to execute the task. WES enables only those tasks that the user is authorized to execute. As the workflow progresses, the status of the tasks in the workflow is updated. Workflow authorization is generated when the task is activated. Each instance of an authorization has an unique ID and is recorded in the AB.

Workflow Authorization Server (WAS): WAS is the core part governing the security administration in SecureFlow. WAS interacts with execution server and provides authorization support for workflow execution. The features of WAS include security policy specification, session management, granting and revocation of authorizations and authentication of users. Workflow Security Administration Interface provides a web interface for workflow related security administration and session management. It allows security officers to inquire, create, update and delete roles, users, AT as well as separation of duties constraints. To specify a security policy, a security officer accesses WAS through the Workflow Security Administration Interface. Since the main focus of this paper is on this module, we provide a detailed description in the next subsection.

Workflow Client: The Workflow client resides on each of the participating hosts and communicates with the workflow execution server. On the client's end, it also provides API to interact with the underlying application programs that perform the task.

A separate administrative authorization base is employed to provide access control to access the workflow specification module, workflow authorization server and the workflow execution server. For example, the policies such as "only workflow managers are allowed to specify new workflows or modify existing workflows," are enforced through the administrative authorization base, which is not shown in the figure 1.

3.2 Workflow Authorization Server

The authorization server consists of an Authorization Specification Module, an Authorization Generation Module and an Authorization Repository.

Authorization Specification Module (ASM): This module allows users to state workflow related access control policies. These specifications are in-turn written to the authorization repository that will be enforced during the workflow execution. There are six submodules in ASM: User Specification, Role Specification, Role-user assignment, Authorization Template Specification (ATS), Object type specification, and Constraint Specification (CS). While specifying a role, a hierarchy code is assigned to indicate the domination relationship of the role in an authorization hierarchy. This information is maintained in USER, ROLE, OBJECT_TYPE and AT relations, respectively. All the constraint information are stored in CONSTRAINT and CT relations, respectively.

Authorization Generation Module (AGM): This

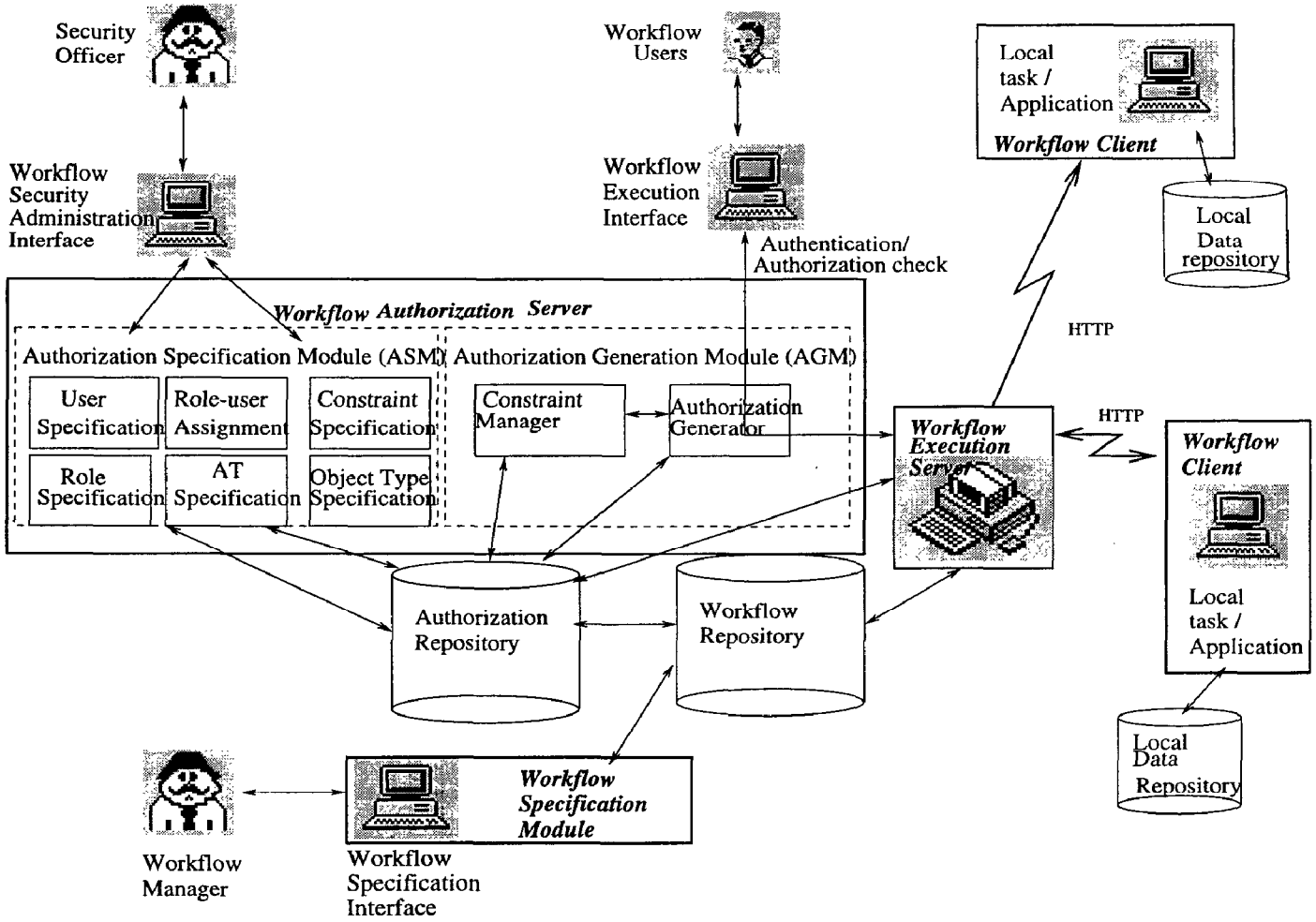

Figure 1: System Architecture for SecureFlow

module is responsible for computing ESS for each task, and generating the appropriate authorization. AGM comprises of two components: Constraint Manager (CM) and Authorization Generator (AG). CM assembles all the security constraints relevant to a task, executes the constraints and generates an ESS that is used to authorize the users. Since the ESS is computed based on the authorization history, the current instance and workflow status, the ESS is generated at run time. The AG then takes the given request, consult the authorization template and ESS to generate the required authorization. Part of the AG serves as an authentication/Authorization check.

The authorization generation process is described in the following steps:
(1) Based on the task information from the execution request, CM first fetches the AT(s) that are associated with the task.
(2) For each AT, CM assembles the constraints table for the associated constraints for the given task.
(3) It then executes the SQL constraints and produces the ESS for the particular execution request.
(4) AG authenticates the subject against the ESS and verifies the object from the desired object type. If the authorization is verified, the actual authorization is generated and archived at AB. Otherwise, the execution request is rejected.

Authorization repository: It contains workflow related security information such as security policies, ATs, and authorization base. Information of authorization repository is stored in a relational database structure. As workflow execution progresses, all authorizations that have been generated along with the execution are stored in the authorization repository. It requires only a limited size as the content of authorization base will be purged periodically.

Figure 7 in appendix B shows the entity relationship diagram of the authorization repository. Some workflow-related entities such as WORKFLOW, TASK, DEPENDENCY and WORKFLOW STATUS are stored in a separate workflow repository and are related to other security-related entities in the authorization repository.

3.3 Database Schema

CT(CTID,TemplateType,Template)
CONSTRAINT(ConstraintID,Constraint,taskID)
ESS(ESSID,TaskID,ObjectID,Subject)
AB(Subject,ObjectID,Privilege,StartTime,EndTime)
AT(ATID,TaskID,Role,ObjectType,Privilege,
StartTime,EndTime)
OBJECT_TYPE(ObjectType,Description)
OBJECT_INSTANCE(ObjectID,ObjectType,Status,
Value)
SUBJECT(Subject,Department,Position,Role)
TO_DO(TaskID,ObjectID)
ROLE(Role,HierarchyCode)
WORKFLOW(WorkflowID,WorkflowDescription)
TASK(TaskID,WorkflowID,TaskDescription,
StartTime,EndTime)
DEPENDENCY(DependencyID,WorkflowID,
ParentTask,ChildTask,Dependency)
WORKFLOW_STATUS(WorkflowID,ObjectID,
CurrentState,Subject)

4 Security Policy Specification and Enforcement

Since role and object types are partially ordered, we allow authorizations to be inherited from child roles to their parents and from parent object types to their children. Consider a general authorization template $AT = \{(R,-),(O,-),pr,[\tau_l,\tau_u]\}$. This says that all roles that dominate role R and any sub-type objects of object type O are qualified for this template.

Security constraints can be realized through a three-phase process.

4.1 Constraint Specification Phase

During this phase, the security officer specifies constraints with the help of built-in constraint templates. We show below two examples of constraint templates, one for exclusive type and the other for assertive type. The system allows creation of new constraint templates. The corresponding database schema is in section 3.3.

For example, the two constraint templates shown in figure 2 can be used to specify constraints c_1 and c_2 in example 2.1. That is the variables $selected_task, $selected_object are instantiated by the appropriate values such as check preparation, check, respectively. Notice that the constraints are bound only at the task level but not at the instance level during this phase. In other words, they are not yet applied to a specific instance of the workflow.

Some of the Workflow-wise variables:

- $selected_object[ ] refers to the current selected item(s) from the list.
- $selected_task refers to the task currently selected by the user.
- $current_role refers to the role currently selected by the user.
- $current_subject[ ] refers to the subject(s) who currently login to the WES and submit the execution request.
- $selected_workflow[ ] refers to the current selected workflow by the user.

During the specification of a constraint, the task on which the constraint is imposed will also be indicated. At the end of constraint entry, constraints are stored in CONSTRAINT table. Figure 3 shows the two actual constraints that can be generated for c_1 and c_2 in example 2.1.

4.2 Constraint Binding and Execution Phase

As the workflow progresses, the status of workflow instances are monitored and recorded with a timestamp in TO_DO and OBJECT_INSTANCE tables.

When a task is triggered automatically by a dependency or is activated by a human subject, he or she is authenticated against the role in AT. A list of object instances in TO_DO list with respect to the task are retrieved and displayed to ask for execution. A user can determine which objects, from the list of objects, to be used to perform the task. The selected object is then assigned to the variable $selected_object. Alternatively, it can also be set to refer to the earliest item in the list as a default. Constraints related to the task are then retrieved from the CONSTRAINT database along with the value assigned to the variables and from the applicable SQL statements. Since the binding of objects and tasks are done at run time, the separation of duties constraints are enforced at instance level. In the case where multiple constraints are specified on a task, all relevant SQL statements will be executed to derive the result. For example, if a user selects ck_5 for constraint c_1 and $purchase\_request_2$ for c_2 as his choice, they will result in the queries in figure 4, respectively.

4.3 Authorization Generation Phase

During this phase, the eligible subject set is computed and if the user who submits a request is among this set, authorization is granted. The ESS_i($selected_object) is implemented as a hash array where $selected_object serves as a key to uniquely identify the element in ESS_i. The following is used to determine whether a subject is authorized to perform the task on a certain object ($selected_object).

if $current_subject $\in ESS_i$($selected_object) then grant authorization; otherwise reject access

CTID	Template Type	Template
ct1	Exclusive	select distinct subject from SUBJECT, AT where SUBJECT.role=AT.role and AT.taskid=$selected_task and SUBJECT.subject not in (select subject from AB, AT where AB.objectid=$selected_object and AB.subject=SUBJECT.subject and AB.privilege=AT.privilege and AT.taskid=$selected_source_task);
ct2	Assertive	select distinct subject from SUBJECT where SUBJECT.role =$selected_role and SUBJECT.department not in (select department from SUBJECT S2, AB where S2.subject=AB.subject and S2.role=$selected_role and AB.privilege=$selected_privilege and AB.objectid=$selected_object);

Figure 2: Constraint Templates

ConstraintID	Constraint	TaskID
c1	select distinct subject from SUBJECT, AT where SUBJECT.role=AT.role and AT.taskid='tw3' and SUBJECT.subject not in (select subject from AB, AT where AB.objectid=$select_object and AB.subject=SUBJECT.subject and AB.privilege=AT.privilege and AT.taskid='tw1');	tw3
c2	select distinct subject from SUBJECT where SUBJECT.role ='manager' and SUBJECT.department not in (select department from SUBJECT S2, AB where S2.subject=AB.subject and S2.role='manager' and AB.privilege='approve' and AB.objectid=$selected_object);	tw2

Figure 3: Generated Constraints

```
select distinct subject from SUBJECT, AT
where SUBJECT.role=AT.role and
AT.taskid='tw3' and
SUBJECT.subject not in  (
select subject from AB, AT where
AB.objectid='ck5' and
AB.subject=SUBJECT.subject and
AB.privilege=AT.privilege and
AT.taskid='tw1');

select distinct subject from SUBJECT where
SUBJECT.role ='manager' and
SUBJECT.department not in (
select department from SUBJECT S2, AB where
S2.subject=AB.subject and
S2.role='manager' and
AB.privilege='approve' and
AB.objectid='purchase\_request2');
```

Figure 4: Queries

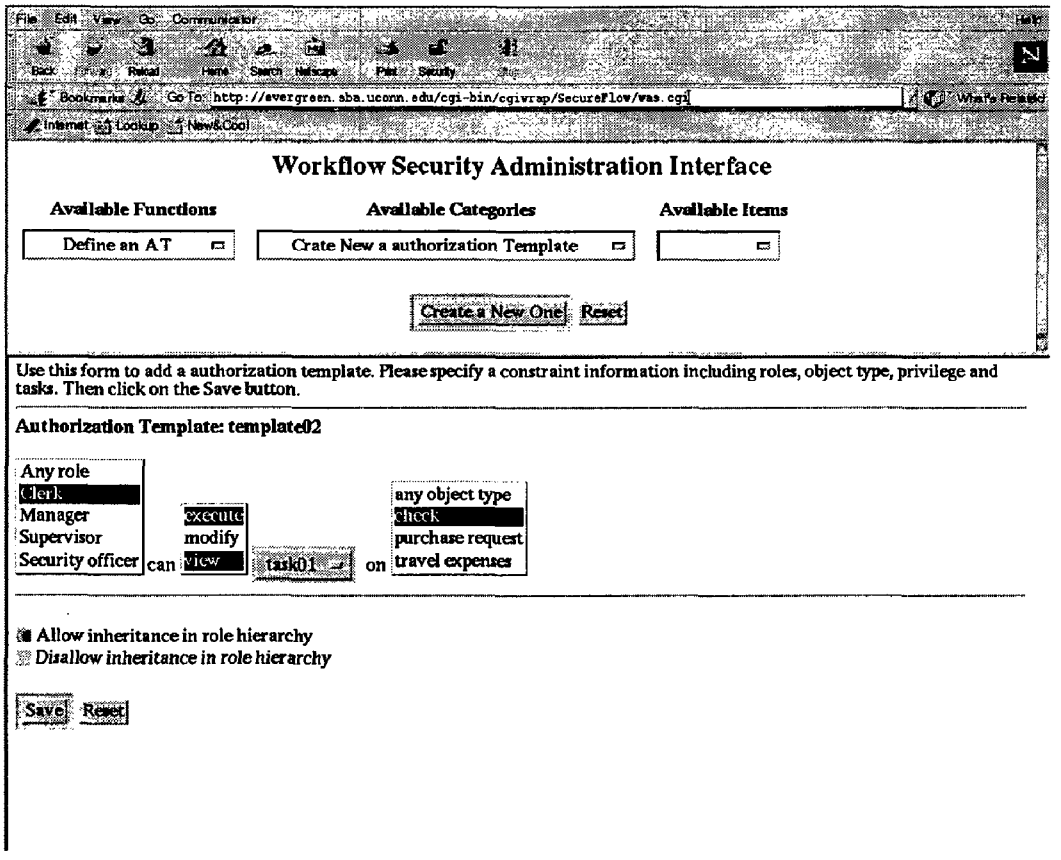

Figure 5: Specification of Authorization Templates

For example, if the result from the query in the second phase is $ESS_3(ck_5)$ = {John, Mary} and $current_subject is Mary, an authorization {Mary, ck_5, Issue, [40,80]} will be generated.

5 Implementation

SecureFlow can be implemented on any platform as long as it supports multi-threading and messaging. However, the popularity of WWW, the standardization of common protocols such as HTTP, Open Database Connectivity (ODBC) and Simple Workflow Access Protocol (SWAP) as well as JAVA enbedded browsers present a readily available platform for developing workflow processes across heterogeneous platforms. Currently, we are in the process of implementing the components of SecureFlow with HTML, Javascript, Java and Perl CGI programming. Figures 5 and 6 show the interface of the authorization specification module.

As shown in figure 5, to specify an AT, a task is first selected. The associated role(s), object type(s) and the privileges can be specified from pull-down lists to assist the input.

As shown in figure 6, the authorization specification interface facilitates the specification of security constraints imposed in the workflow activities via the visual interface and facilitates to input SQL statements. It allows user to specify a constraint from a constraint template list. The selected template is displayed as a form prompting users for the value of the variables. User can also create a new constraint template or a constraint by expressing the constraint in SQL with workflow-wise variables.

6 Conclusions

The objective of the paper is to present a web-based Workflow Management System, called *SecureFlow* that serves as a framework for specification and enforcement of complex security policies within a workflow such as separation of duties. The main advantage of SecureFlow is that it uses a simple 4GL language such as SQL, thereby improving flexibility and user-friendliness in specifying authorization constraints. Due to the modular structure of the SecureFlow architecture, workflow authorization module, called the workflow authorization server, can be separated from the entire workflow system. Thus, the security specification and enforcement modules can be layered on top of existing workflow systems that do not provide adequate support for security.

References

[1] Nabil R. Adam, Vijayalakshmi Atluri, and Wei-Kuang Huang. Modeling and Analysis of Workflows Using Petri Nets. *Journal of Intelligent Information Systems*, 10(2), 1998.

[2] Vijayalakshmi Atluri and Wei-Kuang Huang. An Authorization Model for Workflows. In *Proceedings of the Fifth European Symposium on Research in Computer Security, in Lecture Notes in Computer Science, No.1146, Springer-Verlag*, September 1996.

[3] Vijayalakshmi Atluri and Wei-Kuang Huang. A Petri Net Based Safety Analysis of Workflow Authorization Models. *Journal of Computer Security, to appear*, 1999.

[4] Elisa Bertino, Elena Ferrari, and Vijayalakshmi Atluri. A Flexible Model Supporting the Specification and Enforcement of Role-based Authorizations in Workflow Management Systems. In *Proc. of the 2nd ACM Workshop on Role-based Access Control*, November 1997.

[5] C. Bussler, S. Jablonski, and H.Schuster. A new generation of workflow management systems: Beyond taylorism with mobile. *ACM SIGOIS Bulletin*, 17(1):17–20, 1996.

[6] David D. Clark and David R. Wilson. A comparison of commercial and military computer security policies. In *Proc. IEEE Symposium on Security and Privacy*, pages 184–194, Oakland, California, April 1987.

[7] Umeshwar Dayal, Qiming Chen, and Tak W. Yan. Workflow technologies meet the internet. In Asuman Dogac, Leonid Kalinichenko, M. Tamer Ozsu, and Amit Sheth, editors, *Advances in Workflow Management Systems and interoperability*, pages 343–358. NATO Advanced Study Institute, 1997.

[8] Johann Eder, Herbert Groiss, and Walter Liebhart. The workflow management system panta rhei. In Asuman Dogac, Leonid Kalinichenko, M. Tamer Ozsu, and Amit Sheth, editors, *Advances in Workflow Management Systems and interoperability*, pages 129–144. NATO Advanced Study Institute, 1997.

[9] Wei-Kuang Huang and Vijayalakshmi Atluri. Analyzing the Safety of Workflow Authorization Models. In *Proc. of the 12th IFIP WG 11.3 Workshop on Database Security*, July 1998.

[10] IBM. Ibm flowmark: Modeling workflow, version 2 release 2. In *Publication No. SH-19-8241-01*, 1996.

[11] S. Kandala and R. Sandhu. Extending the BFA Workflow Authorization Model to Express

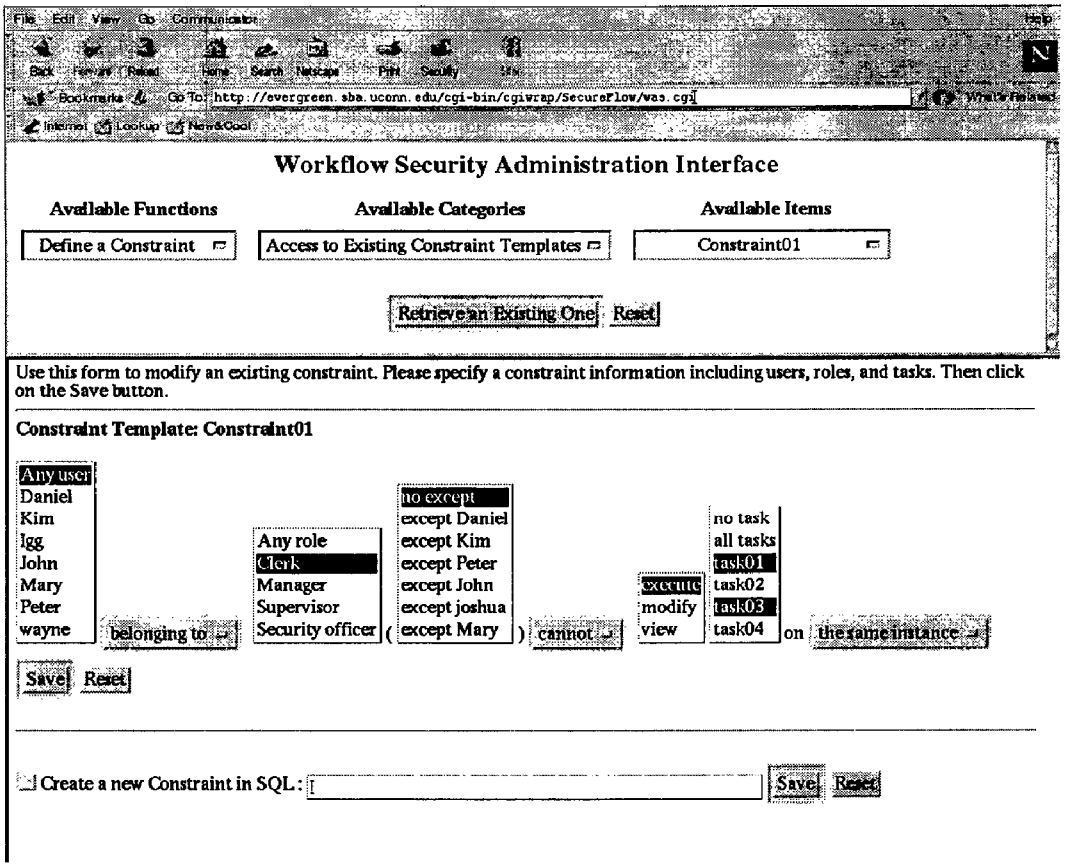

Figure 6: Specification of Authorization Constraints

Weighted Voting. In *13th IFIP WG11.3 Working Conference on Database Security*, page to appear, 1999.

[12] Kamalakar Karlapalem and Patrick C. K. Huang. Security Enforcement in Activity Management Systems. In *Advances in Workflow Management Systems and Interoperability*, pages 166–194. NATO Advanced Study Institute, 1997.

[13] Ravi S. Sandhu. Separation of Duties in Computerized Information Systems. In Sushil Jajodia and Carl Landwehr, editors, *Database Security, IV: Status and Prospects*, pages 179–189. North Holland, 1991.

[14] Ravi S. Sandhu et al. Role-based Access Control Models. *IEEE Computer*, pages 38–47, February 1996.

[15] Ming-Chien Shan, Jim Davis, Weimin Du, and ting Huang. Hp workflow research: Past, present, and future. In Asuman Dogac, Leonid Kalinichenko, M. Tamer Ozsu, and Amit Sheth, editors, *Advances in Workflow Management Systems and interoperab ility*, pages 91–105. NATO Advanced Study Institute, 1997.

[16] Gottfried Vossen and Mathias Weske. The wasa approach to workflow management for scientific applications. In Asuman Dogac, Leonid Kalinichenko, M. Tamer Ozsu, and Amit Sheth, editors, *Advances in Workflow Management Systems and interoperability*, pages 145–165. NATO Advanced Study Institute, 1997.

A The E-R Diagram of SecureFlow Data

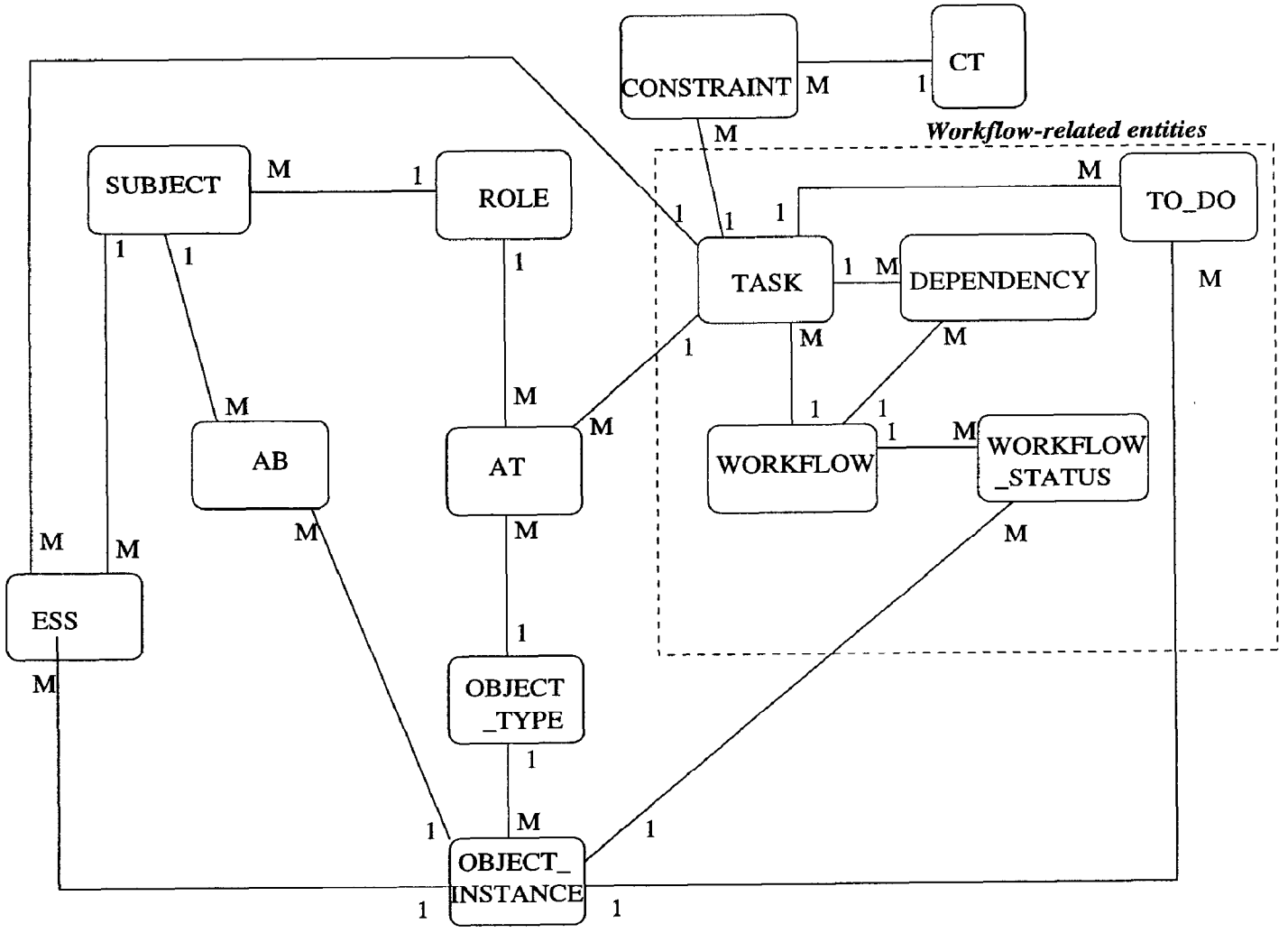

Figure 7: The E-R Diagram of the Authorization Repository