Authorization and Access Control of Application Data in Workflow Systems

Shengli Wu[1]
The University of Strathclyde, Glasgow, Scotland G1 1XH
Email: shengli@cs.strath.ac.uk
Amit Sheth, John Miller
LSDIS Lab, University of Georgia, Athens, GA 30602
Email:{amit,jam}@cs.uga.edu
Zongwei Luo[2]
IBM T.J. Watson Research Center, Yorktown Heights, NY 10598
Email: zongwei@us.ibm.com

Abstract: Workflow Management Systems (WfMSs) are used to support the modeling and coordinated execution of business processes within an organization or across organizational boundaries. Although some research efforts have addressed requirements for authorization and access control for workflow systems, little attention has been paid to the requirements as they apply to application data accessed or managed by WfMSs. In this paper, we discuss key access control requirements for application data in workflow applications using examples from the healthcare domain, introduce a classification of application data used in workflow systems by analyzing their sources, and then propose a comprehensive data authorization and access control mechanism for WfMSs. This involves four aspects: role, task, process instance-based user group, and data content. For implementation, a predicate-based access control method is used. We believe that the proposed model is applicable to workflow applications and WfMSs with diverse access control requirements.

[1, 2] This research was performed while the author was at the LSDIS Lab, University of Georgia (http://lsdis.cs.uga.edu) and was funded in part by NIST-ATP sponsored Healthcare Information Infrastructure Program (70ANB5H1011) and Naval Research Laboratory sponsored "Workflow Management for Advanced DoD Applications" and "Extending METEOR with Workflow Reuse, Adaptation, and Collaboration" projects.

Keywords: Workflow management system, Authorization, Access Control, Predicate-based Access Control, Workflow Process Metadata-data, Security, Workflow Repository

1. Introduction

Workflow management systems (WfMSs) are widely used in numerous application domains. Workflow activities (or tasks) and transitions are components of workflow applications that capture and represent business process and business logic. According to the Workflow Management Coalition's Workflow Reference Model [H94], data used in workflow systems can be divided into three kinds: control data, workflow relevant data, and application data. Workflow control data is maintained by the workflow enactment service to identify the state of individual process or activity instances or other status information. Workflow relevant data is used by the WfMS to determine the state transition of a workflow process instance. Usually, both of these two kinds of data are not accessible, or only accessible in a very limited way, by ordinary users. Hence the access control policy for both of them is relatively simple. While workflow application data is application specific and every user may use it; its access control requirements may become very complicated in many applications.

Many access control requirements [BBFR98] exist for workflow applications and WfMSs. However, the access control and authorization requirements for data involved in applications managed by WfMSs have not received much attention. We focus on authorization and access control of workflow application data in this paper.

Until recently, many WfMSs did not manage application data directly, but left that to the application systems themselves. A case in point is the reference model [H94] of WfMC in which security of application data at the workflow level has not been discussed. Unfortunately, in such a situation, the security protection of application data may be compromised. In the example that we will discuss in Section 3, some requirements (such as Regulations 1 and 3) cannot be met if the WfMS does not provide an access control mechanism for them. Therefore, we believe a workflow-level access control mechanism is necessary for many workflow applications.

The rest of this paper is organized as follows. Section 2 presents a brief background and related work in this area. In Section 3, we discuss access control requirements for application data that should be supported by WfMSs and illustrate them using a healthcare workflow example. Four major aspects for access control are identified. In Section 4, a comprehensive access control model is presented. Section 5 discusses an approach for implementing the access control model. The key idea is to use a predicate-based access control method [BM82, CFMS95] and a repository to manage the metadata of workflow processes. Section 6 concludes the paper.

2. Background and Related work

The next three paragraphs of this section introduce useful terminology and concepts, followed by related work.

A business process (or process in short) is a set of one or more linked tasks or activities that collectively realize a business objective or policy goal, normally within the context of an organizational structure defining functional roles and relationships. Usually, a workflow process is developed in two phases by using corresponding services of a WfMS. They are workflow creation (designing or building) and workflow enactment (execution or runtime). During the creation phase, process definitions are specified. During the enactment phase, instances of process definitions are executed.

One concept relevant to process enactment is process instance-based user group. There may exist many instances of the same process definition running concurrently under the control of the workflow enactment service. A process instance may involve several human users, who form the user group for that process instance. Because every process instance is created and executed at run time, the corresponding user group that is involved in a particular process instance only exists at run time when the process instance is being executed. What is more, usually we cannot know all the members of any group until all the tasks of a process instance have been started. Two major reasons are as follows: Firstly, when starting to execute a new task in a process instance, the WfMS has to select a suitable user to execute it. The decision of such a selection process can be affected by many factors: eligible users/roles to execute this task which are specified at build time, the users available when the selection process is carried out, the selection

policy of the WfMS, and so on. Secondly, each process instance may need to execute a different subset of all tasks defined in the workflow process definition, or may execute in different sequences, which depends on various factors as well.

Typically, the whole workflow process is very complicated and it is necessary to define it as a collection of tasks. There are data and control dependency relationships among these tasks. WfMC [WfMC96] considers two kinds of tasks, automated and manual. While in some commercial products or research systems, more kinds of tasks are introduced. For example, METEOR [SKMW96, SK99], a WfMS developed at the Large Scale Distributed Information Systems (LSDIS) Lab, University of Georgia, supports some more kinds of tasks such as transactional, non-transactional, compound (network), two-phase commit, manual and so on. A process can be composed hierarchically through compound tasks to effectively manage complexity. That is, a compound task itself is a subworkflow. Therefore, tasks can be organized in the form of complex hierarchical structure.

Now we discuss some of the related work. Access control is one of the important aspects that provide the foundation for information and system security [SS96]. Several kinds of access control models have been proposed for operating systems, database systems, and various kinds of information systems. Among them, role-based access control (RBAC) model is both important and popular. The concept of RBAC began with multi-user and multi-application on-line systems pioneered in the 1970s [SCFY96]. Now it has been used extensively in many kinds of information systems and with diversified enhancements in functionality. Another kind of access control model that will be used in this paper is predicate-based access control [CFMS95, BM82].

For workflow security, previous research has been done mainly on several aspects, which include task assignment constraints, inter-workflow security, and multilevel secure workflow systems [B01]. Using task assignment constraints, assignment methods for the workflow systems are specified in terms of constraints on the permissible assignments of users to tasks and roles. Because the role-based model is a natural choice for implementing security in workflow systems, most of the discussions are based on that. Bertino et al. [BFA99] proposed a formal logical authorization model for assigning users and roles to tasks, with both static and dynamic authorization

constraints. Castano et al. [CCF01] proposed a rule-based model. In their model, the constraints can apply to a specific instance or all instances of a process definition. Time constraints are also supported. Atluri and Huang's work in [AH96a] focused on using Petri Nets to present an authorization model.

Inter-workflow security is concerned with the security of the communication and cooperation of autonomous workflow systems, running at different units of the same organizations or at different organizations. Some related work can be found in [WfMC98], [MFWA99], and [VA98].

Multi-level secure workflow is a part of inter-workflow security. This rich research area is explored, among others by the Workflow Management Coalition [WfMC98], Miller et. al. [MFWA99], Kang et al. [KFSK99], and Atluri et al. [AHB97, AHB01], Atluri and Huang [AH96b].

Castano et. al. [CCF01] present a summary of security adopted in commercial workflow systems, which including IBM MQSeries, Staffware2000, InConcert, and Cosa. All these products provide certain kinds of security mechanisms for supporting task assignment constraints. IBM MQSeries, Staffware2000, and Cosa allow duty constraint binding, while InConcert allows external applications that are invoked at task assignment time to determine the role-to-task assignment.

Data access control is another issue that needs to be considered for WfMS security. However, in many cases, this issue has not been mentioned [B00]. To our knowledge, [AH96a] is the only one that has considered this issue. In their paper, Atluri and Huang proposed an authorization model for workflow systems. Two aspects, role and task, are considered. However, their work does not distinguish between the types of data involved. This paper is focused on introducing an adequate access control mechanism for workflow systems to protect application data. Besides role and task, other aspects such as process instance-based user group and content need to be considered for access control of application data. Furthermore, some implementation-related issues need to be considered as well. For example, in workflow systems, application data could be managed either by the WfMS or by other application systems. How to address this problem has to be considered. Understanding real-world application requirements through efforts to apply technology in partnership with industry is critical in developing practical and useful

approaches and techniques. At the LSDIS lab we have had significant partnerships involving technology development, prototyping, trialing and commercialization involving the METEOR project and the healthcare, defense and telecommunications industries [SKMW96, SWK+97, KSM99, IC].

In the rest of this paper, we will discuss access control requirements identifiable in a healthcare workflow application, and then follow up with a comprehensive access control model to meet these requirements.

3. Access control requirements in WfMSs

In this section, we discuss some access control requirements in the context of an information intensive healthcare workflow application [RT97]. Medical records contain a great deal of data about people, such as height and weight, blood pressure, and notes about bouts with the flu, cuts, or broken bones. These records may also contain some sensitive information of people such as fertility and abortions, emotional problems and psychiatric care, sexually transmitted diseases, HIV status, substance abuse, physical abuse, genetic predisposition to diseases, and so on. Access to this information must be carefully controlled [RT97]. Now let us discuss some application requirements to illustrate what the new challenges are for access control mechanisms in workflow systems.

The top-level tasks of a healthcare workflow process are shown in Figure 1 using the METEOR workflow builder tool [SWK+97]. Among all top-level tasks, *Check*, *Diagnosis*, and *Payment* are compound tasks, and all other tasks are manual tasks. Various roles and their dominance relationships are given in Figure 2. If role $\mathbf{r_i}$ dominates role $\mathbf{r_j}$, then there is an arc from $\mathbf{r_j}$ to $\mathbf{r_i}$. In the role hierarchy, a *Physician* can be an *Internist*, a *Surgeon*, a *Pediatrician*, a *Gynecologist*, a *Psychiatrist*, or a *Venerologist*. Surgeons can be further divided into several subtypes. The role-task assignment relationship is stipulated in the following way: A *Receptionist* can undertake both tasks of *Appointment* and *Register*; *Check* is required to be undertaken by *Nurse*; *Physician* is specified to execute *Diagnosis* and *ReferToSpecialist*; both *MedicineConsulting* and *MedicineDispensing* are executed by *Pharmacist*; *X_ray* and *Ultrasound* are undertaken by related specialists, respectively.

A hospital has many departments. Each type of physicians belongs to the corresponding department, e.g., all Surgeons (General) belong to the General Surgery Department.

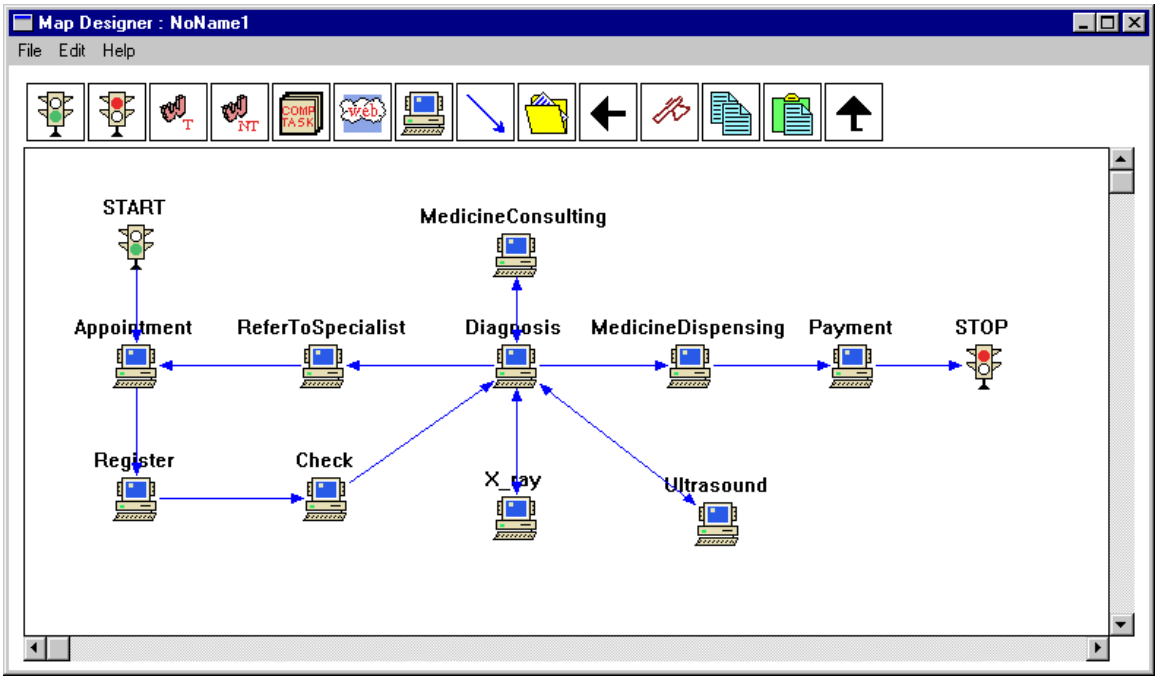

Figure 1: Task and their relationships in healthcare workflow process

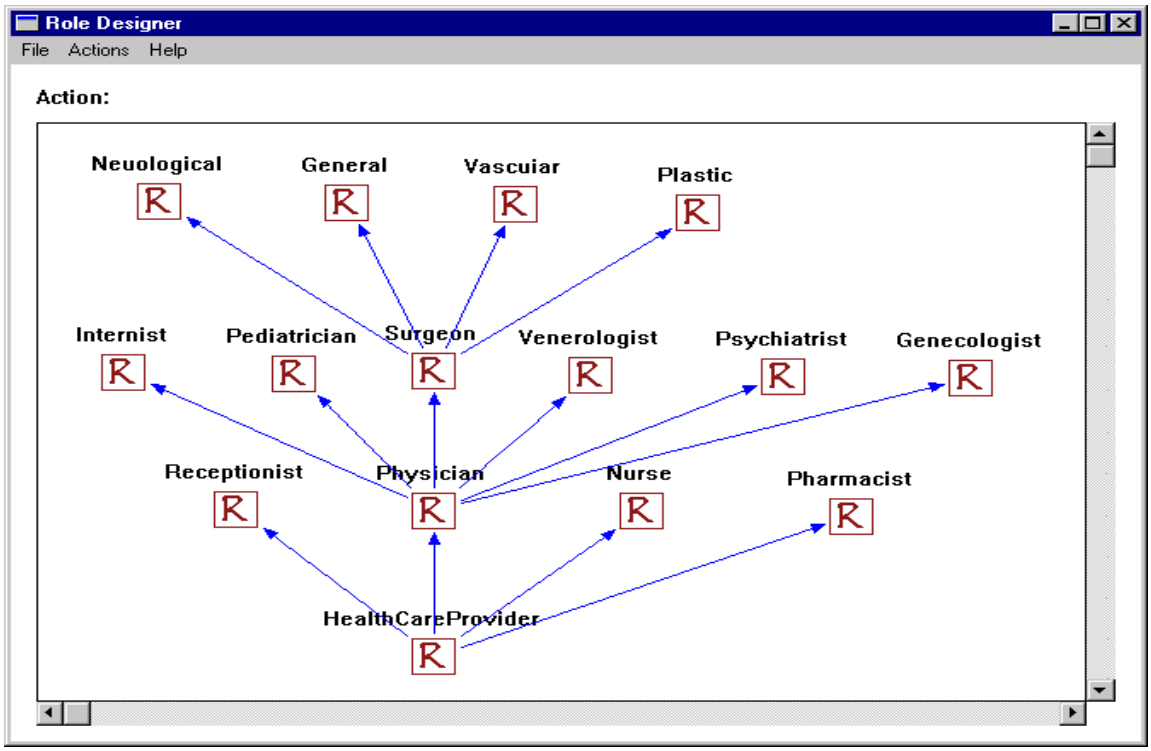

Figure 2: Role hierarchy for healthcare workflow process

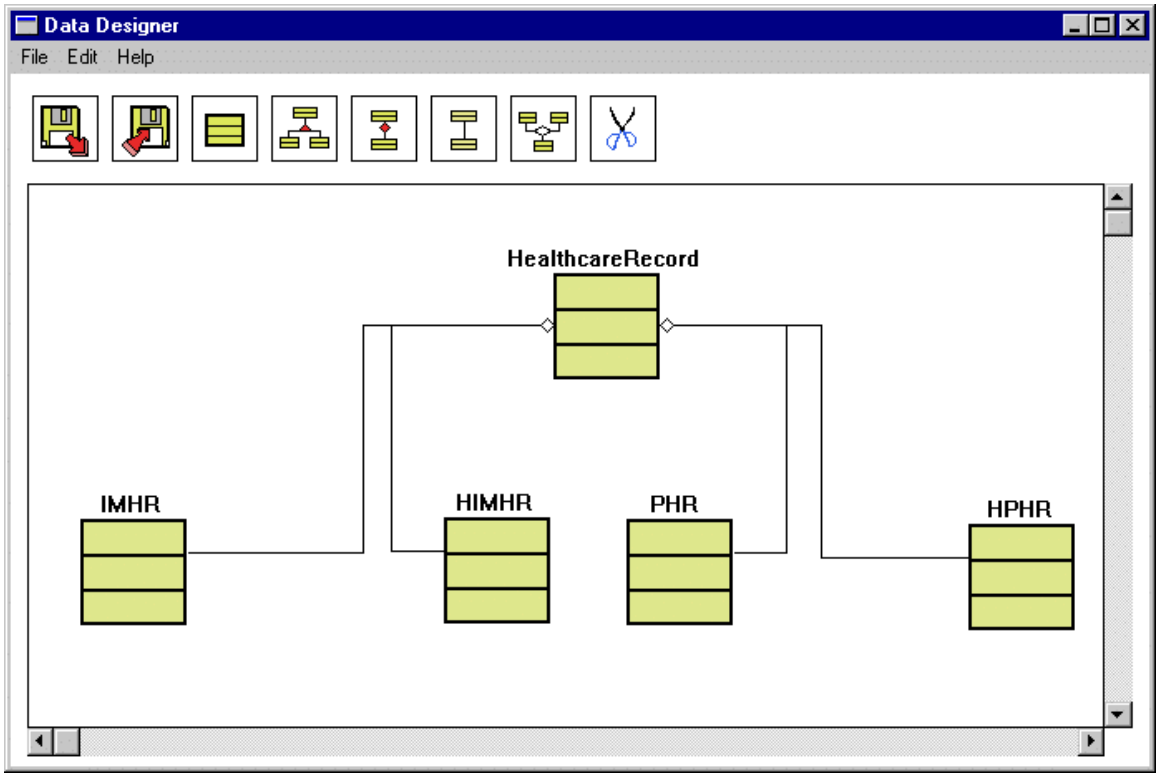

Figure 3. Data structure in healthcare workflow process

The hospital maintains a hospital-wide *HealthCareRecord* table (also called Master Patient Index, [SKMW96]) that records some basic information of all patients who have been treated or are being treating by the hospital. Every department has its own healthcare record tables. Upon every visit of a patient, a new "encounter" record is inserted into the table corresponding to the department of the physician who treats the patient. When a patient checks in, the receptionist checks if the patient's records are in the HealthCareRecord table. If the patient does not have a record in the HealthCareRecord table (that means this is his/her first visit to the hospital), the receptionist first creates a new record in the HealthCareRecord table for that patient, then creates a new record in the department table (such as Internal Medicine or Pediatrics). If the patient does, the receptionist just creates a new record in the corresponding department table for the patient. For those patients who are treated now in the hospital, they have current records in the department tables. For those patients who have finished their treatments, the hospital keeps their medical records for a certain period of time. Such records are called historical records. Current and historical records of patients are kept in separate tables, which are *current healthcare record* and *historical healthcare record*, for every department. A patient may have more than one (historical) medical record in different department tables. Figure 3 shows a general HealthCareRecord table for the whole hospital and four tables used by two departments, Internal Medicine and Psychiatry. The structures of these tables are given in the Appendix. This example will be used throughout the paper.

Assuming the process shown in Figure 1 is used by all departments in the hospital[3], we will discuss some regulations that the hospital may stipulate and means by which the workflow system could support them effectively.

3.1 Process instance-based user group

Regulation 1. For every patient's visit there is a group of healthcare professionals who are involved in the treatment of that patient. Such a group may consist of one (or several) physician, one (or several) nurse and some other personnel. This situation is much easier to observe especially for inpatients. To maintain as much privacy as possible, it is

[3] In practice, it is possible that special workflow processes are needed for some departments due to their particularities. However, it is a better choice for most departments to share the same process.

stipulated that only a member of the group treating that patient is allowed to check the patient's current medical record when performing the tasks assigned to him/her.

Discussion: This regulation is based on process instance-based user group. A physician, a nurse or a receptionist may be involved in the treatments of several patients, and therefore, take part in several medical groups at the same time. Role-based or discretionary access control does not suffice in such situations. So it is necessary for the workflow system to support the concept of *process instance-based user group*, or *user group* for short.

3.2 Data content

Regulation 2. When performing the *Diagnosis* task, a physician *d* can browse all historical healthcare records of

- the patient *p* that *d* is treating (may include those records of *p* that were not treated by *d*; some more restrictions are discussed in Regulation 5 below), and
- those cases of other patients besides *p* the physician *d* treated before.

Discussion: A major difficulty with this requirement is due to the large amount of historical healthcare records. Usually, in a hospital there are thousands of patient healthcare records. Using access control tabulation with every healthcare record as a single controlled object is not a good solution. Because in such a way, we have to keep a huge set of tuples for the access control tabulation, even more tuples than that of the healthcare records themselves.

An effective way to solve this problem is using predicate-based access control [CFMS 95, BM82]. We put some related information such as names and employee numbers of physicians along with the patient's records. We use a predicate that specifies "a physician can query those healthcare records in which the physician's ID (an attribute of healthcare record) equals his/her ID". In such a way, just one rule can substitute for many records in the access control tabulation. Thus, we need the access control mechanism to support predicate-based access control.

3.3 Task

Regulation 3. A pharmacist can perform two tasks. One is to dispense medicine for a patient according to the prescription of a physician. In this task, he/she cannot read any

medical record of that patient. The other is to provide medicine consultation for physicians. For an effective consultation, he/she needs to know some related information about that patient. So when performing that task, he/she is allowed to read those records of the patient that the physician who consults him/her can read.

Discussion: When performing different tasks, a pharmacist needs to be given different privileges. Therefore, task should be a factor in the access control mechanism.

3.4 Privilege propagation

Also, in Regulation 3 (refer to Regulation 1 as well), another issue is that different physicians may consult the same pharmacist; thus, a pharmacist needs to have different privileges in different situations, even while performing the same task, *MedicineConsulting*. Such privileges cannot be decided statically at build time as usual[4]. Hence, we need a privilege propagation function from one role to another role in certain circumstances.

3.5 Role

Regulation 4. When performing the *Check* task, a nurse can read the current record of the patient. Besides, she can add some related items, such as the reading of pulse rate or blood pressure of the patient into the record as well, but she cannot read any other records except the one she is working with. Similar restrictions apply to *Receptionist* when he/she performs task *Register*.

Discussion: From Regulations 1, 2 and 4, we can see that on the one hand, even in the same process instance group, different roles such as Nurse and Physician may have different privileges. On the other hand, different users who play the same role, but are involved in different process instances will need different privileges. Therefore, role and process instance-based user group are two distinct concepts. Neither of them can act as a substitute for the other.

[4] A decomposition of the related task is a solution without introducing new functions for access control mechanism. For example, we can decompose *MedicineConsulting* into *InternalMedicineConsulting*, *SurgeryMedicineConsulting* and so on. But in this example, if the hospital has 20 departments, we must add 19 extra tasks.

3.6 Dynamic authorization

Regulation 5. The historical medical records of most departments (except *Gynecology*, *Venereology*, and *Psychiatry*) can be shared by all physicians to some extent. Specifically, when a physician treats a patient, he/she can read the historical medical records of that patient in most departments besides his/her own. However, the medical records of *Gynecology*, *Venereology*, and *Psychiatry* cannot be read unless the patient agrees to this.

Regulation 6. In the treatment of a patient, some members of his/her medical group may be changed from one healthcare provider to another. The successor must get the same privilege as his/her predecessor.

Discussion: Typically, authorizations are defined at build time, so a workflow designer decides the exact relationships among users, roles, process instance-based user groups, tasks, and objects. The above two regulations show that such designing and changing functions are needed at run time as well. In Regulation 5, it is necessary to define new access control rules, and in Regulation 6, it is necessary to change user/role assignment at run time. For supporting these regulations, all related information must be kept in such a way that dynamic changes are allowable at run time. Consistency is the key concern for making changes when many process instances are running. As adaptive and dynamic workflow is an important aspect in workflow systems [SK99, SO99], dynamic authorization is surely a part of it that needs to be supported. However, in this paper we do not intend to discuss the detailed solution to this problem because it is rather implementation-oriented.

We conclude this section with some discussion of general requirements for the access control mechanism over application data in workflow systems. From the healthcare workflow application we discovered quite a few access control requirements. All regulations discussed in this section have to be enforced at the same time. This cannot be achieved by any current workflow system, nor by any access control model proposed so far. We expect that many of these access control requirements not only exist in the healthcare application, but also in many other workflow application fields.

In some situations, e.g., a database system or a file management system is used by the workflow system to store its application data, some of the access control requirements at the workflow level can be met by enforcing adequate access control rules in the database system or file management system. However, some of the requirements cannot be met without a workflow level security mechanism. In the above example, some regulations (such as Regulations 1 and 3) cannot be met if the WfMS does not provide an access control mechanism for them. So a workflow-level access control mechanism is necessary for managing application data.

Role is an important concept for organizational and security models and is extensively used in many kinds of software systems. This is also the case for workflow systems.

Process instance-based user groups are definitely needed to support cooperation among a group of people to complete a job (process instance). Besides the healthcare field, many other kinds of applications such as banking, insurance, legal & law enforcement, immigration, have similar access control requirements. Investment processes in a bank, case investigation for damage claims in an insurance company, case judgement in a court proceeding, and immigration application processing in a nation's immigration service, typically need more than one person's efforts. Letting just the party concerned, but not anyone else, to access the information they need is a general security principle. So process instance-based user group makes excellent sense.

Task is another important aspect. Even with roles and process instance-based user groups, authorization cannot be specified in an accurate manner. A user usually may need to access different information when he/she is performing different tasks. So including task designation can make the authorization and access control model in better accord with the *least privilege* principle. On the other hand, just supporting task designation is not enough as well, because different users (e.g., doctors/nurses) may need different privileges when performing the same task (e.g., diagnosis/check). Therefore, we have the following observations about workflow-level access control mechanisms for application data:

- Process instance-based user group, task, and role are three important aspects that the access control mechanism of workflow systems need to support simultaneously.

- Data contents need to be considered when a workflow application includes a large amount of application data, as in the healthcare workflow application.
- The problem of privilege propagation may be encountered in some situations, so it is useful for the access control model to support it.
- Dynamic authorization is an important aspect of adaptive workflow systems that needs further work.

4. A comprehensive access control model

In this section, we will present a comprehensive access control model to meet the requirements (except dynamic authorization) discussed in Section 3. As a preparation, let us introduce some notations and assumptions about WfMS first. Based on that, we will describe all the components used, then present the access control model, and discuss some related authorization rules and specifications. Examples from the healthcare application will be given to explain how the model works.

A process instance is a single enactment of a process definition, including its associated data. We distinguish different process instances by assigning a unique ID to each of them. Every running process instance has a set of relevant variables to represent their enactment state. Many variables are defined by the workflow system automatically. For example, the ID of a process instance, the user group of a particular process instance, the performer of every task in a process instance, and so on. Such environment variables are called system variables. Some examples of environment variables are:

- #ThisInstance.ID – It denotes the current process instance's ID (the process instance that the user is being involved is called current process instance).
- #ThisTask.Name – The current task's name (the task that the user is performing is called the current task).
- #ThisRole.Name – The role name that the user is playing.
- #ThisUser.ID – The ID of the user.

We also assume that a workflow designer can define additional variables in a process definition. For instance, in a healthcare workflow application, we can define a variable

PatientID for it. Such variables are called user-defined variables. Then at any time when a process instance is executed, the user can access all these variables.

4.1 Components

The access control model relies on the following components:

4.1.1 Subject

The first component is subject that includes role, user, and process instance-based user group. We have a set of users **U**. Each user is a member of it. Roles are named collection of privileges and represent organizational agents intending to perform certain job functions within an organization. Roles are hierarchically organized in an organization. We use **R** to indicate a set of roles $\mathbf{r_i}$ ($1 \le i \le n$) and $<_R$ to indicate a role hierarchy. Let $\mathbf{r_i}$, $\mathbf{r_j} \in \mathbf{R}$ be roles. We say that $\mathbf{r_i}$ dominates $\mathbf{r_j}$ in the hierarchy ($\mathbf{r_i} <_R \mathbf{r_j}$), if $\mathbf{r_i}$ precedes $\mathbf{r_j}$ in the ordering. Figure 2 is an example of role hierarchy in the healthcare workflow application. There is a many-to-many relationship between users and roles to indicate which user can play which role.

A process instance-based user group includes all the users who are involved in a particular process instance. It is dynamic because for any process instance, all the user-task assignments are done at run time. So when the process instance keeps running, more and more users will be added to the user group of that process instance. There is a many-to-many relationship between users and process instance-based user groups.

4.1.2 Task

The second component is task. Tasks are organized as a tree in the workflow process. We use a tree **T** to indicate a set of tasks with a relation $<_T$ to indicate task hierarchy. For tasks $\mathbf{t_i}$, $\mathbf{t_j} \in \mathbf{T}$, we say that $\mathbf{t_i}$ includes $\mathbf{t_j}$ in the hierarchy ($\mathbf{t_i} <_T \mathbf{t_j}$), if $\mathbf{t_j}$ is a sub-task of $\mathbf{t_i}$ in the workflow process. An example of task hierarchy in the healthcare workflow process is shown in Figure 4 (with some tasks omitted for brevity).

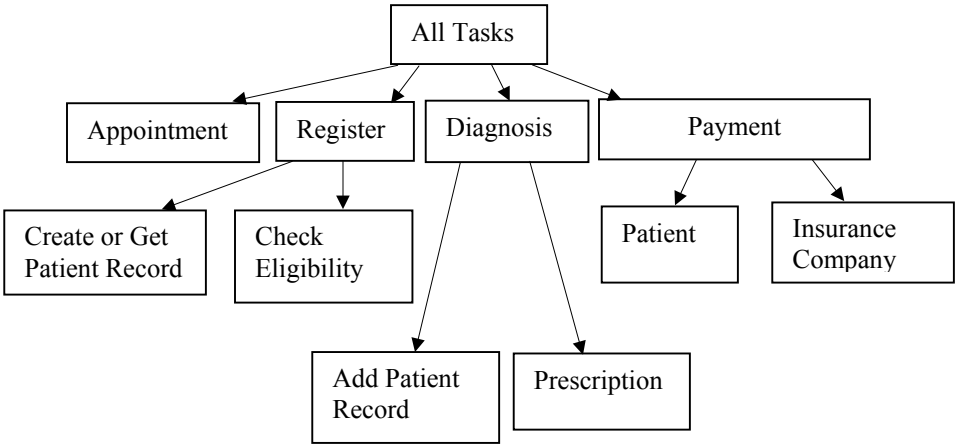

Figure 4. An example of task hierarchy

4.1.3 Object

The third component is object. Each object $\mathbf{o} \in \mathbf{O}$ includes a set of objects $\{o_1, o_2, \ldots, o_n\}$. For any $\mathbf{o} \in \mathbf{O}$, we define a group of attributes **security-attri(o)** to describe the access control-related properties of **o**. Every attribute in **security-attri(o)** is called a security attribute of **o**. Such an object definition is general and is adequate for defining diversified kinds of data and files. We can treat a set of document files, a set of audio/video files, a set of executable files, a class of objects in an object-oriented database, a relation instance in a relational database, or even all relation instances in a database as an object $\mathbf{o} \in \mathbf{O}$.

Next, we introduce a classification for application data. It is especially helpful for supporting process instance-based access control automatically. The classification is from a particular workflow process's point of view. We divide application data into two types, *exogenous* and *endogenous*. Endogenous data refer to those data that are generated by the process instances of this process. They can be further divided into two subtypes *historical* and *current*. Historical data refer to those data that are produced by the process instances—which have been completed and do not exist any more. We call such process instances *finished process instances*. Current data refer to those data that are produced by the instances of this process—which are being executed and continue to exist for some

time. We call such process instances *running process instances*. Exogenous data refer to those data that come from the outside of this workflow process.

We adopt three domains, **current**, **historical**, and **exogenous,** to store different types of data. For any object **o** ∈ **O**, they must belong to one and only one of these domains. In our model, we wish to support process instance-based access control automatically only in the current domain. Two major reasons are as follows. Firstly, application data in the current domain need such an access control mechanism much more than those in either the historical domain or exogenous domain. Secondly, it can be implemented automatically only for data in the current domain. Because objects in the current domain are created by currently active process instances of the process definition, the workflow system can identify which process instance creates them without human intervention. For historical and exogenous data, determining which objects could be accessed by users in which process instance is very difficult. However, if necessary, similar access control rules can be enforced for historical and exogenous data by using content-based access control. For supporting process instance-based access control, every object **o** ∈ **O** in the current domain gets a security attribute ProcessInstanceID automatically.

4.1.4 Constraint

The fourth component is constraint. **C** is a set of constraints. Each constraint is a Boolean expression with the following syntax:

<Boolean-expression> ::= <conjunctive-item> {OR <conjunctive-item>}

<conjunctive-item > ::= <compare-predicate> {AND <compare-predicate>}

<compare-predicate> ::= <left-value> <operator> <right-value>

<left-value> ::= <security-attribute-variable>

<right-value> ::= <constant> | <workflow-system-environment-variable>

| < security-attribute-variable >

<operator> ::= ‘=’ | ‘!=’ | ‘<>’ | ‘>’ | ‘<’ | ‘>=’ | ‘<=’

These elements are mostly self-explanatory. Some additional explanation is in order for the basic operands and operators.

- A constant can be any type of string, float, integer, etc.;
- The current process’s ID, the performer of the current task are examples of workflow system environment variables;

- A security attribute variable is an attribute in set **security-attri**(**o**) of an object **o** ∈ **O**. We use **rel(c)** to denote the set of all objects **o** ∈ **O** whose security attributes appear in constraint **c** as attribute variables. Because **c** is always with a particular object **o** in an authorization, any *valid* **c** in **C** the following two conditions must be hold at the same time: (a) $\exists$**o** (**o** ∈ **O** ∧ **o** ∈ **rel**(**c**)); and (b) $\neg\exists$($\mathbf{o_1}$, $\mathbf{o_2}$) ($\mathbf{o_1} \in \mathbf{O} \wedge \mathbf{o_2} \in \mathbf{O} \wedge \mathbf{o_1} \neq \mathbf{o_2} \wedge \{\mathbf{o_1}, \mathbf{o_2}\} \subseteq$ **rel**(**c**)). That is to say, one and only one object's security attributes may appear in any **c** ∈ **C**.
- Both the left and right sides of a <compare-predicate> need to be of same data type. All the operators can be used for numerical comparison, but for string comparison, only two operators '=' and '!=' can be used.

4.1.5 Privilege

The last component is a set of privileges (or access rights) **P**, which indicates the access modes subjects can exercise on the objects. Although adjustable, we illustrate with eight types of privileges: *select*, *update*, *delete* and *insert* for database objects, *read*, *edit*, *destroy*, and *new* for document files.

4.2 Access control model

The access control model is formalized in the following definition.

Definition 1. The access control model consists of the following components and relationships among those components:

- **U**, **R**, **T**, **O**, **C** and **P** represent user, role, task, object, constraint, and privilege, respectively as introduced in Section 4.1;
- **RoleHierarchy** ⊆ **R**×**R** is a partial order on **R** called the role dominance relationship. $<_R$ is used to represent that relationship;
- **TaskTree** ⊆ **T**× **T** is a partial order on **T** called the task inclusion relationship. We use $<_T$ to represent the inclusion relationship.
- **UserRoleAssignment** ⊆ **U**×**R** is a many-to-many user to role assignment relationship.
- **RoleTaskAssignment** ⊆ **R**×**T** is a many-to-many role to task authorization relationship.

- All objects in **O** are divided into three types (domains): **current**, **historical** and **exogenous**. Any object **o**∈**O** belongs to one and only one of the above three types (domains).
- **ObjectPrivilege** ⊆ **O**×**P** is a many-to-many object to privilege possession relationship.
- **PermissionAssignment1** ⊆ **RoleTaskAssignment**×**ObjectPrivilege**×**C** is a permission relationship from role and task to object and access privilege (*select*, *read*, *update*, *edit*, *delete* or *destroy*) with a certain constraint.
- **PermissionAssignment2** ⊆ **RoleTaskAssignment**×**ObjectPrivilege** is a permission relationship from role and task to object and access privilege (*insert*, *new*, *select*, r*ead*, *update*, *edit*, *delete* or *destroy*).

4.3 Authorization rules and specifications

4.3.1 Authorization rules

In our authorization mechanism, an authorization is a 5-tuple (**r**, **t**, **o**, **p**, **c**) or a 4-tuple (r, **t**, **o**, **p**) where **r** ∈ **R**, **t** ∈ **T**, **o** ∈ **O**, **p** ∈ **P**, **c** ∈ **C**, (**r**, **t**) ∈ **RoleTaskAssignment**, and (**o**, **p**) ∈ **ObjectPrivilege**.

Tuple (**r**, **t**, **o**, **p**, **c**) indicates that any user who plays role **r** is authorized to exercise privilege **p** on some restricted objects in set object **o** when he/she performs task **t**. If **o** ∈ **O** belongs to the historical or exogenous domain, then the Boolean expression in **c** must be satisfied for these component objects in **o**. If **o** ∈ **O** belongs to the current domain, then **c'** = **c** and $\mathbf{c}_{pi}$ must be satisfied. Here $\mathbf{c}_{pi}$ becomes true if the user is a member of the user group of the current process instance. In such a way process instance-based access control can be supported automatically.

Two situations need to be considered.

- **p** = *insert* (*new*): any user who plays role **r** is authorized to insert new tuples into **o** (assuming **o** is a relation instance in a database), or create a new document file in **o** (assuming **o** is a set of document files) when performing task **t**;
- **p** = *select* (*read*), or *update* (*edit*), or *delete* (*destroy*): For data in either historical or exogenous domain, any user who plays role **r** is authorized to exercise privilege **p** on

all tuples (or document files) of **o** when performing task **t**. For data in the current domain, constraint $\mathbf{c}_{pi}$ must be satisfied for those tuples.

Example 1. The definition of the HealthCareRecord object is given in the Appendix. Tuple (Internist, Diagnosis, HealthCareRecord, select, PatientName = "John Smith") states that when performing the Diagnosis task, any internist is authorized to query those tuples with "John Smith" being their patient's name in the HealthCareRecord table.

Besides explicit authorizations, additional authorizations can be derived through the following rules.

1. An authorization given to a role **r** in performing task **t** propagates to all roles which precede **r** in the role hierarchy (that is, to all roles **r'** such that **r'** $<_R$ **r**).
2. An authorization given to a role **r** in performing task **t** propagates to all sub-tasks that are included in **t** (that is, to all tasks **t'** such that **t'** $<_T$ **t**).

4.3.2 Some specifications about authorizations

There is a close relationship between two kinds of authorizations in the workflow system. One is **RoleTaskAssignment** that stipulates which role is allowed to execute a given task, the other is **PermissionAssignment** (including **PermissionAssignment1** and **PermissionAssignment2**) that stipulates which role is allowed to access which object within a certain task provided that role is permitted to perform that task. In this paper, we mainly discuss the latter. Obtaining the privilege of executing a task is a prerequisite for a user to access data in that task. What is more, if a user undertakes a particular task, but not any other task, then he/she is eligible to access data stipulated by data authorization for that particular task. In Definition 1, the consistency of two kinds of authorizations can be guaranteed by specifying **RoleTaskAssignment**, but not **Role**×**Task** as components of **PermissionAssignment1** and **PermissionAssignment2**.

In this current work, to reduce the complexity of the access control model, only positive authorizations are considered, so no explicit authorization conflict occur. For a user who plays one role, he/she will possess the union of privileges of those authorization tuples for that role.

Different WfMSs may have their own ways to define task assignment. Some research has been done on this issue [BFA99, CCF95, AH96a] to support various kinds of constraints. In this paper, both UserRoleAssignment and RoleTaskAssignment are used

for that purpose. If necessary, this part can be expanded to support more complicated constraints such as dynamic and temporal constraints. Such changes or enhancements may not affect our access control model.

4.3.3 Some particular constraints

In this section, we discuss how to support some regulations proposed in Section 3 by defining particular constraints with authorization rules. A related discussion about process instance-based user group will be given in Section 5.3.

For Regulation 5 discussed in Section 3.6, we can use the following method. When a patient exits the hospital after visiting the Gynecology, Venereology, or Psychiatry Department, he/she should decide if he/she wants to allow his/her healthcare records to be accessed or not later on by physicians in other departments. We define a security attribute *ArgeeToAccess* for that. If he/she agrees, then we use a "yes" as that attribute's value; otherwise, we put a "no" in it.

Another way to support Regulation 5 is to let every patient have a PIN-code/password for all his/her historical healthcare records in the Gynecology, Venereology, and Psychiatry Departments. When other physicians hope to access these records, they need the patient to be present and input his/her PIN-code/password into the system to indicate his/her approval of such a query. A security attribute *Password* can be used for this purpose.

Privilege propagation is needed in some situations. We have seen such application requirements in Section 3.4. Now let us see how the model supports that. When a pharmacist is asked to provide a medicine consultation for a patient, the pharmacist may obtain different privileges if the consultations come from different physicians.

Example 2. The following 3 tuples are used to represent the privileges of a pharmacist on three objects (table PHR, HPHR, and IMHR) when performing the MedicineConsulting task. The definitions of these tables are given in the Appendix. Both PHR and IMHR are in the current domain, and HPHR is in the historical domain.

1. (Pharmacist, MedicineConsulting, PHR, select);
2. (Pharmacist, MedicineConsulting, HPHR, select, #ThisInstance.PatientID = PatientID and #Task(Diagnosis)Performer.Role = "Psychiatrist" or #ThisInstance.PatientID = PatientID and ArgeeToAccess = "Yes");

3. (Pharmacist, MedicineConsulting, IMHR, select).

Here we have two environment variables. One is #Task(Diagnosis)Performer.Role, which gives the name of the role that the user who executes task Diagnosis plays. Another is #ThisInstance.PatientID, which is a user-defined variable and gives the Patient's ID for the current process instance. The first and third tuples indicate that the pharmacist can query the healthcare record that is owned by the current process instance. The second tuple is for a historical record HPHR. #ThisInstance.PatientID = PatientID and #Task(Diagnosis)Performer.Role = "Psychiatrist" means that if the consultation seeker is a psychiatrist, then the pharmacist is allowed to check the patient's historical record in the Psychiatry department. #ThisInstance.PatientID = PatientID and ArgeeToAccess = "Yes" means that if the consultation seeker is not a psychiatrist, then the pharmacist is allowed to check the patient's historical record in the Psychiatry department with the agreement from the patient.

5. Implementation issue

In this section, we discuss the implementation of access control mechanisms, based on the METEOR WfMS consisting of a workflow designer/builder, enactment service and respository. METEOR's compresensive workflow model has support for role and security specification [KFSK99], and its fully distributed enactment services [KSM99, SK99, IC], support various types of tasks, exception handling, and workflow adaptation. At build time, we specify data authorization along with other aspects of workflow process definition. They are kept as metadata and stored in the repository of the workflow system.

In general, a repository is a persistent object manager with an object model designed to help users manage descriptions of metadata. A workflow repository is a database of information about workflow processes, data, organizations and other components of workflow design such as task realizations, communication protocols, external applications and resource interfaces. Key advantages of storing workflow design information in a repository include convenient versioning, lifecycle management, querying of objects, and sharing design information between various tools.

5.1 Workflow repository

Part of the workflow repository that is related to authorization and access control is shown in Figure 5. We use UML (the Unified Modeling Language) to describe them.

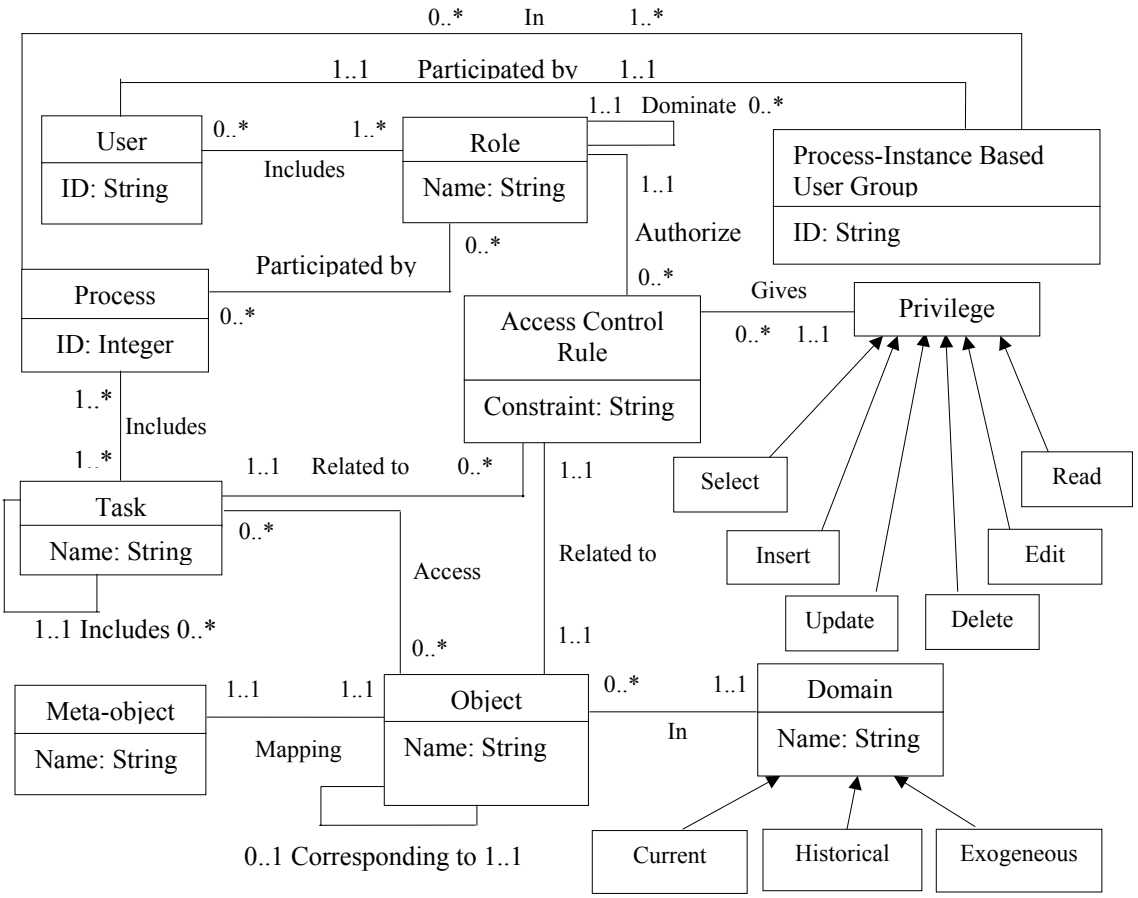

Figure 5. Some metadata related to authorization and access control

In Figure 5, there is a one-to-one relationship between the objects of class **object** (all of its instances are **O**, see Section 4.1.3) and **meta-object**. In class **meta-object**, every object includes all the security attributes, ID, and other information of an object of class **object**. The class **object** may include several sub-classes, such as relation, document file and so on, depending on the requirements of the specific application. Accordingly, there are corresponding sub-classes in **meta-object**. For every object $\mathbf{o} \in \mathbf{O}$ in the current domain, there must be a corresponding object $\mathbf{o'} \in \mathbf{O}$ in the historical domain, and vice versa. The *Corresponding to* relationship between them shows such a relationship

between two objects. The workflow system maintains invocations for databases or other kinds of applications.

The objects of class **process** indicate all running instances of that process. There is a *Participated by* relationship between **process** and **role** indicating all participants in a process instance. Similarly, there is a *Participated by* relationship between **process** and **process instance-based user group**.

Almost all the information about users, roles, objects, tasks, user-to-role mapping, role-to-task mapping, and access control rules is specified at build time by the workflow designer. However, process and process instance-based user group are two exceptions, which are created by the enactment service of the WfMS when it starts to run new instances.

5.2 Creating metadata tables for objects

For every object **o** $\in$ **O**, we have to create a table to keep its metadata. The metadata of an object may include many kinds of things, such as identification, type, location, application invocation, security attributes, and so on of the object. Here we only discuss attributes of two types-- identification and security attributes. Putting identification of every element of **o** in its metadata table can facilitate the one-to-one mapping relationship between an object and its metadata.

For every object created in the current domain, we add an attribute ProcessInstanceID to it for the purpose of access control. Some other security attributes may also be added according to the application requirement.

5.3 Process instance-based user group

The current domain includes data or files that are generated by running process instances, so it is possible for us to enforce such an access control rule automatically. In the current domain, a security attribute *ProcessInstanceID* is defined for all objects. The value of ProcessInstanceID for an object denotes the ID of the process instance that owns the object. In the constraint component of an authorization rule, the following predicate, ProcessInstanceID = #ThisInstance.ID, is used for that purpose. Here #ThisInstance.ID is

a workflow system environment variable, which indicates the ID of the process instance that the user is working with. ProcessInstanceID is a security attribute variable.

Example 3. In Section 3, Regulation 1 stipulates that only a member of a process instance-based user group is allowed to access the object owned by that group. For this, workflow designers can use the following three authorization tuples, among several others. In this example, we use the role hierarchy shown in Figure 2, and the task hierarchy shown in Figure 4.

1. (Internist, Diagnosis, IMHR, select),
2. (Internist, Diagnosis, IMHR, update),
3. (HealthCareProvider, Register, IMHR, insert).

The definitions of IMHR (Internal Medicine Healthcare Record) and its security attributes security-attri(IMHR) are given in the Appendix. Because IMHR is in the current domain, the access control mechanism will add a basic predicate, ProcessInstanceID = #Thisinstance.ID, to both 1 and 2 above automatically without human intervention. The same treatment is needed for Example 2 discussed before.

According to tuples 1 and 2, an internist can check and update the records of the patient whom he/she is treating when performing the Diagnosis task. The third tuple indicates that healthcare providers can insert new records for patients in task Register.

5.4 Access control calculation

An access control calculation is the evaluation of a Boolean function **B(r**, **t**, **o**, **p**, **c)**—the permission function—on the identity of role **r** who asks for the execution of an operation **p** on some of the elements of an object **o** in which **c** is satisfied in a task **t**. Because we use a general structure for different kinds of objects (e.g., relations, files), the same access control calculation can work for all of them.

In a WfMS, access requests are issued to users with specified **r**, **t**, **o**, and **p**. The following processes are used to determine all the objects allowed to be accessed. First, we find out all the corresponding access control rules (explicitly given in the repository or derived ones) about **r**, **t**, **o**, and **p**: (**r**, **t**, **o**, **p**, $\mathbf{c_1}$), (**r**, **t**, **o**, **p**, $\mathbf{c_2}$), … (**r**, **t**, **o**, **p**, $\mathbf{c_n}$). Second, we perform the following query operation to the corresponding **meta-object** table of **o** in the repository (suppose the **meta-object** table of **o** is called **meta-object** (**o**)):

Select ID(**o**)

From **meta-object** (**o**)

Where **c'** and (**c_1** or **c_2** or … or **c_n**);

Here ID(**o**) is composed of the ID attributes of **o**. If **o** is a relation in a relational database, then we can adopt the primary key of it as the ID attributes of **meta-object** (**o**). If **o** is a set of files, then we can adopt file names as the ID attributes of **metadata** (**o**). c' is the part which is added automatically by the workflow system to support such as process instance-based access control or privilege propagation in certain circumstances. Because only positive authorizations are considered, so (**c_1** or **c_2** or … or **c_n**) is used for providing the union of privileges of those authorization tuples.

After finishing the query operation, we get a set of object IDs with which the user wants to operate, and at the same time check that he/she has adequate privilege to do so. Then, we carry out corresponding processing for different kinds of objects. If **o** is a relation in a relational database, we can do a join operation between **o** and the result table we get at the second step. If **o** is a set of files, then we can find the corresponding file according to the name and category information we get. If **o** is a special kind of data, then we need to invoke the corresponding application interface to access it.

5.5 Automatic object migration

We use two domains to store the current and historical data separately. When a process instance is terminated, all data created in the current domain should be migrated to the historical domain. This can be done automatically by the workflow system. If **o** and **o'** are two corresponding objects in the current and historical domain, we need to change the content of **meta-object**(**o**) and **meta-object** (**o'**). Suppose the related object is $o_1 \in$ **o**, we should delete tuple o_1 in **meta-object** (**o**) and insert a new tuple o'_1 into **meta-object** (**o'**). If **o** is a file set, that is all. No change is needed for the file itself. If **o** and **o'** are two corresponding relation tables in current and historical domain, we have to change the contents of these tables. First we insert tuples into **o'**:

Insert into **o'**

Select *

From **o**

Where ProcessInstanceID = #ThisProcess.ID;

Then delete the old ones from **o**:

Delete from **o** where ProcessInstanceID = #ThisProcess.ID;

Such object migration should be performed for every pair of tables that are located in the current and historical domain, respectively.

5.6 Applicability of this model

Though the implementation is based on METEOR, the model presented in this paper is general enough to apply to many other WfMSs as well.

First, all concepts and components used (e.g. user, role, task, process instance, process instance-based user group, and so on) in our model are common and exist in most WfMSs. Second, there are two different ways for WfMSs to deal with application data. One is to manage them directly by the WfMS, another is by using another application system such as a database system to manage them. The implementation discussed in this section can deal with both situations (in the latter case, two kinds of application systems, which are database and file management system, have been considered). For either of the cases, if we want the data to be protected at the workflow level, the related information (metadata) of those data needs to be provided, and corresponding access control rules need to be set in the WfMS. Besides, for those data managed by a database or file management system, the invocation to the application system needs to be controlled in a systematic way by the WfMS. Direct entry to the application system should be avoided, because this will allow the access control protection in the WfMS to be by-passed. Moreover, using a database (repository) to store those access control metadata and evaluate access control requests is a good practice. As to some other kinds of legacy application systems, a general solution may not be available and a case-by-case analysis is needed for better security solutions.

6. Conclusions

Significant new access control requirements in workflow applications exist. In this paper, we used a healthcare workflow application to illustrate these requirements, and presented an authorization and access control mechanism to support such workflow applications.

Several important aspects, such as task, role, process instance-based user group, and object content, were introduced and considered in the model presented. Our approach used three object domains-- current, historical and exogenous data. This provided the access control model a clearer structure and higher usability than putting all three together.

Key contributions of this paper include the following:

- Through discussion of access control requirements for application data in healthcare workflow applications, we observe the necessity for WfMSs to provide an access control mechanism for application data in many situations.
- The concept of process instance-based user group is introduced and has been identified as an important aspect in the access control model of WfMSs.
- A comprehensive access control mechanism has been provided for application data used in WfMSs. Four aspects, which are role, task, process instance-based user group, and data content, are considered in the access control mechanism.
- A classification of application data used in WfMSs is given. Based on that, a general structure is proposed for handling different kinds of data and files in a unified way.

The work presented in this paper can be extended in several directions. First, more sophisticated authorization methods such as negative authorization may be introduced into the model. Second, how to make it more convenient for the workflow designers to specify access control rules deserves attention. A graphical tool can be defined for such a purpose. The third issue is how to support dynamic authorization at run time in a WfMS. The fourth direction is security solutions for applications of WfMSs across organizational boundaries. For example, if two or more WfMSs work in a cooperative way, and some of them use the access control model proposed in this paper, an inter-workflow security schema needs to be worked out. Because of the diversified nature of application systems, a case-by-case analysis and solution is needed for each of them for reliable security.

Acknowledgements

We gratefully acknowledge Dr. Krys Kochut's critical contributions to the METEOR project and Dr. Budak Arpinar's participation on repository and his feedback on this research. Dr. Myong Kang's (NRL) guidance has also been very valuable for this research.

References

[AH96a] Atluri V. and Huang W-K. An Authorization Model for Workflows. Proceedings of the Fifth European Symposium on Research in Computer Security, Rome, Italy, and Lecture Notes in Computer Science, No.1146, Springer-Verlag, September, 1996, pages 44-64.

[AH96b] Atluri V. and Huang W-K. An Extended Petri Net Model for Supporting Workflows in a Multilevel Secure Environment. Proc. of the IFIP Working Conference on Database Security. pp. 199 – 216.

[AHB97] Atluri V., Huang W-K. and Bertino E. An Execution Model for Multilevel Secure Workflows, 11th IFIP Working Conference on Database Security, August 1997.

[AHB01] Atluri V., Huang W-K. and Bertino E. A Semantic-Based Execution Model for Multilevel Secure Workflows. Volume 8, Number 1, 2000. Journal of Computer Security.

[B01] Barthelmess P. Security in Workflow Systems. URL: http://csel.cs.colorado.edu/~barthelm/security/

[BBFR98] Bertino E., Buccafurri F., Ferrari E., and Rullo P. An Authorization Model and Its Formal Semantics. Proceedings of 5th European Symposium on Research in Computer Security. Louvain_la_Neuve, Belgium, Sep. 16-18, 1998. Pages 127-142.

[BFA99] Bertino E., Ferrari E. and Atluri V. An Approach for the Specification and Enforcement of Authorization Constraints in Workflow Management Systems, ACM Transactions on Information Systems Security, February 1999, Vol 1, No. 1.

[BM82] Bussolati U. and Martella G. Data Security Management in Distributed Databases. Information systems, 7(3): 217-228, 1982.

[CCF01] Castano, S., Casati F., and Fugini M. Managing Workflow Authorization Constraints through Active Database Technology, Information Systems Frontiers, 3(3) September 2001.

[CFMS95] Castano S., Fugini M., Martella G., and Samarati P. Database Security. Addison-Wesley Publishing Company, 1995.

[CKS99] Corley, J., Karp, W., and Sheth, A., "IT at the hearth of Healthcare," Cover Story for the special issue on Healthcare, Silicon India, October 1999, http://www.siliconindia.com/magazine/Oct99it.html

[H94] Hollingsworth D., Workflow Management Coalition: the Workflow Reference Model, Workflow Management coalition, Nov., 29, 1994. http://www.aiim.org

[IC] METEOR EAppS, Infocosm, Inc. http://www.infocosm.com

[KFSK99] Kang, H., Froscher J., Sheth A., Kochut K., and Miller J., A Multilevel Secure Workflow Management System'. In: M. Jarke and A. Oberweis (eds.): 11th International

Conference on Advanced Information Systems Engineering - CAiSE'99, Vol. 1626 of Lecture Notes in Computer Science. pp. 271 – 285.

[KSM99] Kochut, K, Sheth, A., Miller, J., "Optimizing Workflow," *Component Strategies,* Vol. 1, No. 9 (March 1999) pp. 45-57.

[MFWA99] Miller J., Fan M., Wu S., Arpinar A. Sheth P., and Kochut K. Security for the METEOR Workflow Management System, UGA-CS-LSDIS Technical Report, University of Georgia (June 1999) 33 pages.

[RT97] Thomas C. Rindfleisch. Privacy, Information Technology, and Health Care. Communication of the ACM, August 1997, Vol 40, No. 8, page 93-100.

[SCFY96] Sandhu R., Coyne E., Feinstein H. and Youman C. Role-Based Access Control Models, IEEE Computer, Volume 29, Number 2, February, 1996, pages 38-47.

[SKMW96] Sheth A. P., Kochut K. J., Miller J. A., Worah S., Das S., Lin C., Palaniswami D., Lynch J. and Shevchenko I. Supporting State-Wide Immunization Tracking Using Multi-Paradigm Workflow Technology, Proceedings of the 22nd International Conference on very Large Data Bases, Bombay, India, September 1996, 263-273.

[SK99] Sheth A. and Kochut K. J. Workflow Application to Research Agenda: Scalable and Dynamic Work Coordination and Collaboration Systems, in Workflow Management and Interoperability, A. Dogac et al Eds., Springer Verlag, 1999, pp. 35-59.

[SO99] Sadiq S. and Orlowska M. Architectural Considerations for Systems Supporting Dynamic Workflow Modification. Proceedings of the workshop of Software Architectures for Business Process management at the CaiSE'99, Heidelberg, June 14-15, 1999.

[SS96] Sandhu R. and Samarati P. Authentication, Access Control, and Audit, ACM Computing Surveys, Vol. 28, No. 1, March 1996, pages 241-243.

[SWK+97] Sheth, A., Worah, D., Kochut, K., Miller, Zheng, J., Palaniswami, D., Das, S., "The METEOR Workflow Management System and its use in Prototyping Healthcare Applications,", Proceedings of the Towards An Electronic Patient Record (TEPR'97) Conference, April 1997, Nashville, TN.

[VA98] Valia, R. and Y. Al-Salqan: 1997, 'Secure workflow environment'. In: Sixth IEEE Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises. pages 269 – 276.

[WfMC96] Workflow Management Coalition. Terminology and Glossary, Jun., 1996.

[WfMC98] Workflow Management Coalition. Workflow Security Considerations – White Paper, Feb., 1998.

Appendix: The authorization solution to the example of Section 3

In the following we present an authorization solution to the example in Section 3. Only two departments that are Internal Medicine and Psychiatry, and four tasks that are Check, Diagnosis, MedicineDispensing, and MedicineConsultion, are considered. However, that is enough for us to show how the mechanism works for such applications.

The definitions of five tables and their metadata are given. They are HealthCareRecord, IMHR (Internal Medicine Healthcare Record), PHR (Psychiatry Healthcare Record), HIMHR (Historical Internal Medicine Healthcare Record), and HPHR (Historical Psychiatry Healthcare Record). HealthCareRecord is in the exogenous domain. Both IMHR and PHR are in the current domain, while HIMHR and HPHR are in the historical domain. IMHR and HIMHR are corresponding tables in the current and historical domains, so do PHR and HPHR.

For both tables IMHR and PHR, which are in current domain, the predicate "ProcessInstanceID = #ThisInstance.ID" has been added for every authorization tuple hare.

HealthcareRcord (Name, SSN, Sex, BirthDate, InsuranceCompany, InsuranceNo, Address, TelNo);
meta- HealthcareRcord (SSN, ProcessInstanceID);

IMHR (Name, SSN, Sex, BirthDate, Weight, BloodPressure, PulseRate, Symptom, Diagnosis, Prescription, nextAppointmentDate, PhysicianName);
meta-IMHR (ReferenceID, PatientID, PhysicianID, ProcessInstanceID);

HIMHR (Name, SSN, Sex, BirthDate, Weight, BloodPressure, PulseRate, Symptom, Diagnosis, Prescription, nextAppointmentDate, PhysicianName);
meta-HIMHR (ReferenceID, PatientID, PhysicianID);

PHR (PatientID, PatientName, PhysicianID, PhysicianName, SSN, Address, ParentsStative, Sympotom, Diagnosis, Prescription);
meta-PHR (ReferenceID, PatientID, PhysicianID, ArgeeToAccess, ProcessInstanceID);

HPHR (PatientID, PatientName, PhysicianID, PhysicianName, SSN, Address, ParentsStative, Sympotom, Diagnosis, Prescription);
meta-PHR (ReferenceID, PatientID, PhysicianID, ArgeeToAccess);

The authorization rules are defined as follows:

1. (Pediatrician, Diagnosis, PHR, select, ProcessInstanceID = #ThisInstance.ID),
2. (Pediatrician, Diagnosis, PHR, update, ProcessInstanceID = #ThisInstance.ID),
3. (Pediatrician, Diagnosis, HPHR, select, PhysicianID = #ThisInstance.PhysicianID),
4. (Physician, Diagnosis, HPHR, select, #ThisInstance.PatientID = PatientID and ArgeeToAccess = “Yes”),
5. (Internist, Diagnosis, IMHR, select, ProcessInstanceID = #ThisInstance.ID),
6. (Internist, Diagnosis, IMHR, update, ProcessInstanceID = #ThisInstance.ID),
7. (Internist, Diagnosis, HIMHR, select, PhysicianID = #ThisInstance.PhysicianID”),
8. (Physician, Diagnosis, HPHR, select, #ThisInstance.PatientID = PatientID),
9. (Nurse, Check, PHR, select, ProcessInstanceID = #ThisInstance.ID),
10. (Nurse, Check, PHR, update, ProcessInstanceID = #ThisInstance.ID),
11. (Nurse, Check, HPHR, select, ProcessInstanceID = #ThisInstance.ID),
12. (Nurse, Check, HPHR, update, ProcessInstanceID = #ThisInstance.ID),
13. (Nurse, Check, IMHR, select, ProcessInstanceID = #ThisInstance.ID),
14. (Nurse, Check, IMHR, update, ProcessInstanceID = #ThisInstance.ID),
15. (Nurse, Check, HIMHR, select, ProcessInstanceID = #ThisInstance.ID),
16. (Nurse, Check, HIMHR, update, ProcessInstanceID = #ThisInstance.ID),
17. (Pharmacist, MedicineConsulting, PHR, select, ProcessInstanceID = #ThisInstance.ID),
18. (Pharmacist, MedicineConsulting, HPHR, select, ThisInstance.PatientID = Patient.ID and #Task(Diagnosis).RoleName = “Pediatrician” or “#ThisInstance.PatientID = Patient.ID” and ArgeeToAccess = “Yes”),
19. (Pharmacist, MedicineConsulting, IMHR, select, ProcessInstanceID = #Thisinstance.ID),

20. (Pharmacist, MedicineConsulting, HIMHR, select, ThisPharmacist.Patient_ID = Patient.ID).

Authorization rules are arranged according to their use in different tasks. Rules 1-8 are used in the *Diagnosis* task. Rules 9-16 are for the *Check* task, and rules 17-20 are for the *MedicineConsulting* task. No authorization rule is needed for the *MedicineDispensing* task.