Implementing the Chinese Wall Security Model in Workflow Management Systems

Yu-Cheng Hsiao and Gwan-Hwan Hwang
Department of Computer Science and Information Engineering
National Taiwan Normal University, Taipei, Taiwan
swanky.hsiao@gmail.com, ghhwang@csie.ntnu.edu.tw

Abstract- The Chinese wall security model (CWSM) was designed to provide access controls that mitigate conflict of interest in commercial organizations, and is especially important for large-scale interenterprise workflow applications. This paper describes how to implement the CWSM in a WfMS. We first demonstrate situations in which the role-based access control model is not sufficient for this, and we then propose a security policy language to solve this problem, also providing support for the intrinsic dynamic access control mechanism defined in the CWSM (i.e., the dynamic binding of subjects and elements in the company data set). This language can also specify several requirements of the dynamic security policy that arise when applying the CWSM in WfMSs. Finally we discuss how to implement a run-time system to implement CWSM policies specified by this language in a WfMS.

Keywords- Workflow management system (WfMS), Chinese wall security model (CWSM), Role-based access control (RBAC).

1. INTRODUCTION

A WfMS is intrinsically a network-based application [1,2,3]. For a WfMS with a single workflow engine, the participants in activities usually communicate with the workflow engine from different locations via a network system. This requires communication security to be maintained. Furthermore, for a distributed WfMS in which the activities of the workflow process can be executed in different workflow engines [4,5,6], the process instance should be exchanged or transmitted among workflow engines via the network. Thus, a secure network-based WfMS is required to implement major security features such as authentication, confidentiality, data integrity, and nonrepudiation [7,8]. Confidentiality involves prohibiting unauthorized disclosure of information such as the process instances or external data of a WfMS during its execution. Therefore, a WfMS usually has an access control mechanism that the system designer can use to specify how to restrict access to authorized users.

This paper presents how to implement the CWSM in a WfMS. The CWSM, which is also called the Brewer and Nash model [9], and is constructed to provide information security or access controls that can change dynamically. This model was designed to provide controls that avoid conflict of interest (COI) in commercial organizations. Data are viewed in this model as consisting of objects that belong to particular companies. Access to particular parts of data is not constrained by their attributes but rather by what data the subject already holds access rights to. Note that the access related to read, write, or read-and-write operation. The company data set is categorized into mutually disjoint COI classes, as shown in Figure 1. For example, banks, oil companies, and airline companies belong to different COI classes, and the CWSM policy prohibits information flows from one company to another that cause COI. Thus, if a subject accesses the information of bank B1, then s/he is not allowed to access information of other companies within the same COI class, such as that of bank B2 or Bi. However, the subject can access information in another COI class, such as objects of oil company O1.

The CWSM proposes the following mandatory read and write rules:

- **BN read rule**: Subject S can read object O only if O is from the same company information as some object read by S, or O belongs to a COI class within which S has not read any object.

- **BN write rule**: Subject S can write object O only if S can read O by the BN read rule, and no object can be read that is within a different company data set from the one for which write access is requested.

Many access control mechanisms have been proposed and implemented. Access control is traditionally implemented using an access control matrix (ACM), in which an access request from a subject can be granted if the requested access right is recorded in the matrix [10]. The role-based access control (RBAC) model is regarded as a neutral policy and has been the most popular security model in recent years [11]. Data access is restricted to authorized users in this approach,
and it represents a newer alternative approach to mandatory access control and discretionary access control [10]. Roles are created for various operations within an organization, with the permissions to perform certain operations being assigned to specific roles. Subjects are assigned particular roles, through which they acquire the permissions to perform particular operations. RBAC has been applied to access control in WfMSs [12,13,14,15]. Although RBAC has been successful in various applications, we have found it to be insufficient for conducting higher level access control such as the CWSM in WfMSs. When the company information used in a workflow process is static, then RBAC can implement the BN read rule. However, RBAC has problems in implementing the BN read rule if the company information is dynamic. There are at least two cases to consider. The first case is where the same workflow definition uses different company information. Although we know the structure of the company information, we need to design different codes to apply RBAC to them. The second case is where the company information may be created or updated dynamically during the execution of the workflow process. In this case the codes for implementing RBAC need to bind subject to objects dynamically. A lack of knowledge of the COI class and company data set prior to executing the workflow process makes it impossible to design the codes for operating RBAC in advance. In the second situation we need to apply the BN write rule during the execution of the workflow process. Because the application of the BN write rule requires the use of the object access history and the company information to decide if permission for a write operation should be granted, it is impossible to implement the BN write rule purely in RBAC without storing the object access history and the company information in the workflow engine. The main factor making RBAC difficult to implement CWSM is access permission in RBAC being controlled by the binding relationships among subjects, roles, and objects. However, access control in the CWSM depends on the object access history and company information to determine if an access request should be granted. Section 2 provides several examples to illustrate these two situations.

In this paper we show how to implement the CWSM in a WfMS. It is obvious that we should solve the problems that cannot be solved by the popular RBAC model. Our solution is to design a security policy language that allows the system developer to specify the CWSM policy in a WfMS. The proposed language is designed to satisfy the following dynamic security requirements for the CWSM:

1. **Dynamic binding between subjects and companies:** The binding of a subject and a company can be dynamically implemented during the workflow execution. For example, according to the company information shown in Figure 1, if a subject requests access to objects of bank B1 and this access is successful, then s/he is bound to B1. This is the basic requirement that makes the information security access controls of the CWSM able to change dynamically.

2. **Dynamic company information manipulation:** The company information should be able to be created, chosen, or updated dynamically during the workflow execution. As a result, the same workflow definition can use different company information.

3. **Privileged subject management:** The binding of subjects to company information can be controlled during the execution of the workflow process. In some situations, not all data accesses of the participants in a WfMS should be controlled by the CWSM.

4. **Temporal-based enforcement:** The time or duration when the CWSM policy is applied should not be static. In some situations, we may want to set up the CWSM policy to be applied only on part of the activities in a workflow process.

In general, implementing the CWSM should depend on the dynamic behavior of the workflow process. That is, it should be possible to synchronize operations related to the CWSM policy with the progression of the workflow process. This enhances the flexibility of the CWSM because our framework synchronizes access control with the execution of WfMSs. According to the security policy specified by the proposed language, the workflow engine creates data access history tables that store dynamic data accesses performed by subjects. Instead of specifying each data access according to the company information, the workflow engine consults the data access history tables and company information to control the data access so as to implement the CWSM security requirement.

The remainder of the paper is organized as follows. Section 2 uses several examples to demonstrate the limitation of applying RBAC to implement the CWSM in a WfMS. Section 3 presents the proposed language that is used to specify the CWSM policy in a WfMS. Section 4 discusses how to implement the CWSM policy in a WfMS in the proposed language, and the implementation details are presented in Section 5. Conclusions are drawn in Section 6.

2. **LIMITATION IN APPLYING RBAC TO IMPLEMENT THE CWSM**

In the section we present several examples to illustrate the limitations of applying RBAC to implement the CWSM in WfMSs. Figure 2 provides the first example, showing two sets of company information (CI1 and CI2) and a portion of a workflow definition of a business process. Four subjects (John, Mary, Ken, and Leo) are involved in this workflow, and A, B, ..., I are activities. The terms “subject” and “participant” are used interchangeably in this paper, since the latter is a formal term in a WfMS and the former is widely used in access control models such as RBAC – together they represent persons or identities that access data and perform tasks. Assume that we need to apply the CWSM based on company information CI1. Leo is the administrator of John, Mary, and Ken, and is responsible for distributing work to these three persons in activity A. John, Mary, and Ken need to process company data and review the specification of merchandise for one company in CI1. Note that they only perform read
operations to objects, and hence only the BN read rule needs to be enforced in advance. Leo assigns three companies C1, C2, and C3 exclusively to John, Mary, and Ken. For example, if Leo assigns company C1 to John, then John is responsible for the data processing of that company and reviewing the specification of merchandise for company C1 in activities B and F, respectively. With the BN read rule, if company C1 is assigned to John, then John is prohibited from accessing the individual objects owned by companies C2 and C3. In order to implement this security requirement in RBAC, we create three roles R_C1, R_C2, and R_C3 that have been granted the permissions needed to access the individual objects of companies C1, C2, and C3, respectively. After executing activity A, we should assign roles to John, Mary, and Ken according to the decisions made by Leo. The assignment of roles can be represented by the pseudo codes shown in Figure 3A. We assume that the access control mechanism of a WfMS implements the CWSM in RBAC. We use another motivation example to demonstrate this situation.

RBAC is insufficient for implementing the CWSM in RBAC. We use another motivation example to demonstrate this situation.

Figure 2: The first motivation example

RBAC is insufficient for implementing the CWSM in this example because the actual operations of access control change according to the company information, which means that we have to specify the role for each participant separately, even if the company information is static. This motivation example demonstrates that a change in the company data set actually corresponds to a different security access control policy. The RBAC model has to change the role assignment rules when the company information is altered. However, if the company information is generated dynamically during the execution of the workflow process, it is much more difficult to implement the CWSM in RBAC.

Figure 3: Pseudo codes of RBAC

The second motivation example is an extension of the first example. Referring to Figure 4, in activity A' Leo does not perform work distribution as in activity A of the previous example, instead only preparing the related information of the three companies, which means that the company information is dynamically created in activity A'. The preparation or creation of company information is unpredictable since it may depend on the results of previously executed activities. Activities B', C', and D' are three parallel activities. John, Mary, and Ken choose one of the three companies prepared by Leo in A' in a first-come-first-served manner. For example, if John made the choice first, he can choose among the available three companies. Also, he would be responsible for reviewing the specification of merchandise in activity F of the company he chose. The example shows a scenario involving commercial security policy, which is a very popular application of the CWSM [16]. For a set of companies that partially compete with each other and a group of consultants, a consultant is not allowed to work for a company if he has insider knowledge of a competitor. The goal of the CWSM is to prevent information flows from competing companies to the same consultant. Figure 5 shows the pseudo codes of the RBAC operations that implement the security requirement of the example. The codes look similar to that in Figure 3, which implements the security requirement of the first motivation example. However, the codes are divided into four parts and distributed separately to activities A', B', C', and D', which makes them much more difficult to maintain than the codes shown in Figure 3.
has read access to objects of bank B2 and oil company O1. If John is allowed write access to objects of O1, a Trojan-Horse-infected subject executing with John’s privileges can transfer information from objects of B1 to those of O1, which can be read by Mary, who then can read information about banks B1 and B2. It is obvious that if the WfMS wants to comply with the BN write rule for data access, it must keep the run-time history of data accesses and the company information in the workflow engine. Thus, the requirements of the BN write rule cannot be implemented purely by applying RBAC without referring to the history of data accesses and the company information.

3. Language To Specify The Security Policy For The CWSM In WfMS

As discussed in Section 2, specifying a CWSM policy in RBAC is complicated by the need to set up sophisticated role assignments based on the company information and subjects that are required to be involved in the security framework. A convenient way to solve this problem would be to control the read/write access to objects issued by all the subjects involved in a WfMS according to all the available company information automatically. However, this method does not work in a WfMS. In the motivation example shown in Figure 2, Leo is responsible for the work distribution in activity A. It is obvious that Leo should be able to access objects of all companies to perform his task in this business process. If the CWSM of company information C11 is applied to Leo, then he will not be able to perform his task in activity A. It is obvious that Leo needs to refer to data objects of all three companies to distribute work related to the three companies to John, Mary, and Ken. The examples shown in Section 2 provide the motivation to design a language to specify security policy based on the CWSM for WfMSs. First of all, we should be able to bind the subjects to certain company information, which is a basic operation in the CWSM. We should then be able to control when the CWSM associated with the binding starts or ceases. Also, applying the CWSM can be synchronized with the progress or the dynamic behavior of the WfMS. We call this language the “Chinese Wall Security Policy Specification Language for WfMS” (CWSPSL). We use the Backus Naur Form [17] to show the syntax of the CWSPSL (see Figure 6A). Note that all the nonterminal symbols in the figure are underscored.

There are several types of commands in the CWSPSL:

- **LoadCompanyInformation**: Company information that will be used should first be loaded into the workflow engine. Since company information can be represented by a tree data structure, we propose storing the company information in an XML document since such a document intrinsically has a tree data structure [18]. An example of an XML document that represents company information is shown in Figure 6B. Line 1 in Figure 6C shows an example of a **LoadCompanyInformation** statement. Note that variable C11 now represents an entity of the company information.
Subject and company information binding: This statement binds a group of subjects to a set of company information. The binding is assigned to a variable that is subsequently used to start or cease the associated CWSM policy; see line 3 in Figure 6C.

Subject and company information ignore: We assume that all the accesses to objects should be controlled by the workflow engine. However, if the designer wants some subjects not to be controlled by the CWSM, s/he can set these subjects to be able to perform read and write operations to objects contained in certain sets of company information without limitation.

EnforceCWSM: This starts a CWSM policy according to a binding defined in a Subject and company information binding or Subject and company information ignore statement.

CeaseCWSM: This stops the execution of a CWSM policy according to a binding defined in a Subject and company information binding or Subject and company information ignore statement.

TouchR: This command has two parameters, SubjectName and CompanyName, and sets the status of individual objects of company CompanyName as having been read by subject SubjectName.

TouchRW: This command is very similar to TouchR, setting the status of individual objects of company CompanyName as having been read and written by subject SubjectName.

CheckR: This command has two parameters, SubjectName and CompanyName, and is used to check if subject SubjectName should be granted permission to read objects of company CompanyName.

CheckRW: This to used to check if subject SubjectName should be granted permission to read and write objects of company CompanyName.

The CWPSL statements are used to set up the CWSM policy of a workflow process. We assume that the scripts written in the CWPSL are embedded in the execution codes of activities, so that access control based on the CWSM can be synchronized with the progress of the workflow process. This means that these scripts may be executed during the execution of workflow processes. The statements “b1=CWSM(CompanyInformation(CI1), Subject(John, Mary, Ken)); Enforce (b1);” specify enforcing the BN read and write rules to subjects John, Mary, and Ken according to company information CI1 as shown in Figure 2. We can easily change the associated company information to CI2 using the statements “b2=CWSM(CompanyInformation(CI2), Subject(John, Mary, Ken)); Enforce (b2);”. Also, we can specify to apply multiple sets of company information to the same group of subjects as shown in Figure 6C. The CWPSL codes in Figure 6D show how to specify the security policy for the first motivation example shown in Section 2. The statement in line 1 first retrieves the company information used in the workflow process. We assume that the routine “CompanyInformationProcessedByWfProcess()” generates the information of a company. Second, lines 2 and 3 use the command “CWSMignore()” to instruct the system to grant permission for any access requests of objects that are included in company information CI to Leo. Leo then performs work distribution in line 4. Lines 4 and 5 specify the data accesses of three subjects to be controlled by the company information CI. Finally, lines 7, 8, and 9 set up the statuses of individual objects in the company information according to the work distribution performed by Leo.

We now describe how the CWPSL satisfies the four dynamic security requirements mentioned in Section 1. First, the TouchR and TouchRW statements can be used for requirement 1. When the workflow process decides to bind a subject to an element in the company data set, the execution of TouchR or TouchRW achieves the binding. Requirement 2 can
be satisfied by using the **LoadCompanyInformation** statement. The workflow process can prepare the company information in some activities and then execute this statement to load the information of a company to start CWSM-based access control. It is obvious that the workflow process can choose among several sets of company information before it starts access control. Requirement 3 can be met by the **Subject and company information binding** and **Subject and company information ignore** statements. As long as the two statements can be embedded in the execution code of the activities, we can dynamically set up the access of objects contained in the company information by the specified subjects to be controlled or ignored. Finally, requirement 4 is satisfied by **EnforceCWSM** and **CeaseCWSM** statements, since it can control when to start or cease the CWSM-based access control specified by the programmer. The CWSPSL statements are generally designed to be embedded in the execution code of the workflow process. Thus, we can manipulate CWSM-based access control according to the dynamic behavior of the workflow process.

4. RUN-TIME SYSTEM FOR THE CWSPSL AND IMPLEMENTATION

In this section we present the proposed architecture of the run-time system that can implement the security policy specified in the CWSPSL. Figure 7 shows the proposed architecture. The subject accesses objects via the workflow engine. When the workflow engine receives a request from the subject to access an object, it checks the company information as well as the history tables according the BN read and write rules to decide if the request should be granted. The history tables may need to be updated after objects are accessed by subjects. In general, if the required access control is not related to the CWSM policy, we can employ RBAC to control the data access during the execution of the workflow process. However, if the accesses of any data object need to be controlled according to the BN read and write rules, we first establish the company information including the COI class and company data set. These data objects are then allocated to the companies in the company data set that own them. Finally, during the run-time of the workflow process, the workflow engine can decide if a request from a subject to access some object should be granted according to the BN read and write rules.

![Figure 7: Architecture for supporting the execution of the CWSPSL in a WMS](image)

Whilst the company information is built during the run-time of the workflow process, it must be ready before we start enforcing the CWSM policy. Each data object that can be accessed during the workflow execution should be attached to at least one company information if its access control is to follow the BN read and write rules. We first classify companies into different COI classes. We identify which company each static intraprocess and external data object belongs to, and designate it as an object of that company. Objects that are created during the execution of the workflow process need to be added to the company information that owns them.

Each item of used company information has a corresponding history table, which is a two-dimensional array created during the execution of the workflow process. Actually, a history table can be defined as a projection function, HT, that maps a subject and a company into the corresponding access history and is designated as **HT**: S×C→Q, where S is the set of subjects, C is the set of companies, and Q= {R, RW, ∅, I}. **HT(s,c)** is defined as follows:

- **HT(s,c)**=R if subject s has read the objects of company c.
- **HT(s,c)**=RW if subject s has read and written the objects of company c.
- **HT(s,c)**=∅ if subject s has neither read nor written the objects of company c.
- **HT(s,c)**=I if we do not have to control the access of subject s to objects of company c.

Whenever the system executes a **LoadCompanyInformation** command, the access controller creates history tables for the loaded company information. In the execution of the **Subject and company information binding** command, the access controller records the binding relationship between subjects and company information. The **EnforceCWSM** command will add subjects to history tables. In the case where a subject s is supposed to be bound to a specific company information, the **EnforceCWSM** command will add a row for s to the corresponding history table of the information of that company, with elements equal to “∅”. The **CeaseCWSM** command will delete the rows of subjects from history tables. The processing of the **Subject and company information ignore** command is very similar to that of the **Subject and company information binding** command. However, we initialize the elements of history tables to be “I” in the subsequent **EnforceCWSM** command.

For the statements “b1=CWSM(CompanyInformation(C1,C2), Subject(John, Mary, Ken)); Enforce (b);”, subjects John, Mary and Ken are set to apply the CWSM to two sets of company information C1 and C2. Since C1 has three companies (C1, C2, and C3) with the COI class “Bank”, we create a history table as shown in Figure 8A. Similarly, we create other history tables for the binding between C12 and subject groups John, Mary, and Ken (see Figure 8B). The two commands
“b2=CWSMIgnore(CompanyInformation(C1), Subject(Leo));
Enforce (b2)” will add a row for Leo to the history table of
C1, as shown in Figure 8C. Figure 8D shows the situation
where John, Mary, and Ken have read and written to the
objects of companies C1, C2, and C3, respectively.

![Figure 8: Example history tables](image)

The history table clearly indicates whether a subject has
ever accessed objects of a company in the company data set.
The workflow engine is responsible for maintaining the
coherence and the correctness of the history table. Before each
operation (read or write) of objects that belong to a company
in the company data set, the workflow engine needs to look up
an appropriate history table to check if it is allowed.

According to the BN read and write rules, the engine should
deny this access if the subject had ever accessed the objects of
another company in the same company data set. The
“Check_Read” and “Check_Write” algorithms in Figure 9
allow the workflow engine to verify if the access violates the
BN read and write rules, respectively, by implementing the
CheckR and CheckRW commands. Also, the workflow engine
may need to update some history tables after accessing an
object. The “Touch_Read” and “Touch_ReadWrite”

algorithm corresponds to the commands TouchR or TouchRW,
respectively.

```
Algorithm Check_Read(c,o): Check if a subject c has the right to read an object o owned by a company.
Input: A subject c and an object o.
Output: Returns True if c has the right to read o.
(1) Let c be a company that both owns object o and is in company information CI.
(2) If (The corresponding history table of CI is not created) THEN
RETURN False.
ELSE
Let HT be the corresponding history table of CI.
END IF
(3) IF (HT(c,o)=="I", "R", or "RW") THEN
RETURN True.
END IF
(4) Let T be the set of companies that are in the same COI class as c, excluding c.
(5) FOR each element c’ in T, IF (HT(c’,o)=="R" or "RW") THEN
RETURN False.
END IF
END FOR
(6) RETURN True.
```

```
Algorithm Check_Write(c,o): Check if a subject c has the right to write an object o owned by a company.
Input: A subject c and an object o.
Output: Returns True if c has the right to write to o.
(1) Let c be a company that both owns object o and is in company information CI.
(2) If (The corresponding history table of CI is not created) THEN
RETURN False.
ELSE
Let HT be the corresponding history table of CI.
END IF
(3) IF (HT(c,o)=="T") THEN
RETURN True.
END IF
(4) IF (Check_READ(c,o) returns False) THEN
RETURN False.
END IF
(5) LET T be the set of companies that belongs to all the COI classes except that of c.
(6) FOR each element c’ in T,
```

```
Algorithm Touch_Read(c,o): Set subject c to have previously read an object o owned by a company.
Input: A subject c and an object o.
Output: Returns True if c has successfully updated the history table, otherwise it returns False.
(1) Let c be a company that both owns object o and is in company information CI. Let HT be the corresponding history table of CI.
(2) IF (Check_READ(c,o) returns True) THEN
Update HT(c,o) to "R".
RETURN True.
ELSE
RETURN False.
END IF
```

```
Algorithm Touch_Write(c,o): Set subject c to have previously read and written to an object o owned by a company.
Input: A subject c and an object o.
Output: Returns True if c has successfully updated the history table, otherwise it returns False.
(1) Let c be a company that both owns object o and is in company information CI. Let HT be the corresponding history table of CI.
(2) IF (Check_Write(c,o) returns True) THEN
Update HT(c,o) to "RW".
RETURN True.
ELSE
RETURN False.
END IF
```

```
Algorithm Perform_Read(c,o): Read an object o owned by a company by subject c.
Input: A subject c and an object o.
Output: Returns the value of object o if it has successfully read object o, otherwise it returns False.
(1) Let c be a company that both owns object o and is in company information CI.
Let HT be the corresponding history table of CI.
(2) IF (Check_Read(c,o) returns True) THEN
RETURN the value of object o.
ELSE
RETURN False.
END IF
```

```
Algorithm Perform_Write(c,o): Write o to an object owned by a company by subject c.
Input: A subject c, an object o, and a value v to be written to o.
Output: Returns True if c has successfully written to object o, otherwise it returns False.
(1) Let c be a company that both owns object o and is in company information CI.
Let HT be the corresponding history table of CI.
(2) IF (Check_Write(c,o) returns True)THEN
Write v to o.
Update HT(c,o) to "RW".
RETURN True.
ELSE
RETURN False.
END IF
```

```
Algorithm List_Accessable_Objects(c): List all the objects that can be accessed by subject c.
Input: A subject c.
Output: Returns the names of objects and their permissions.
(1) FOR each object o in the system
IF (Check_Write(c,o) returns True) THEN
Show that subject c has read and write permission for object o and display o.
ELSE
Show that subject c has read permission for object o and display o.
ELSE
Show that subject c has no access permission for object o.
END IF
END FOR
```

Figure 9: Algorithms for the CWSM

The “Perform_Read” and “Perform_Write” algorithms
shown in Figure 10 perform passive read and write operations
on data objects. The “List_Accessable_Objects” algorithm is
the skeleton to perform active data access for all the objects.

```
Algorithm Perform_Read(c,o): Read an object o owned by a company by subject c.
Input: A subject c and an object o.
Output: Returns the value of object o if it has successfully read object o, otherwise it returns False.
(1) Let c be a company that both owns object o and is in company information CI.
Let HT be the corresponding history table of CI.
(2) IF (Check_Read(c,o) returns True) THEN
RETURN the value of object o.
ELSE
RETURN False.
END IF
```

```
Algorithm Perform_Write(c,o): Write o to an object owned by a company by subject c.
Input: A subject c, an object o, and a value v to be written to o.
Output: Returns True if c has successfully written to object o, otherwise it returns False.
(1) Let c be a company that both owns object o and is in company information CI.
Let HT be the corresponding history table of CI.
(2) IF (Check_Write(c,o) returns True) THEN
Write v to o.
Update HT(c,o) to "RW".
RETURN True.
ELSE
RETURN False.
END IF
```

```
Algorithm List_Accessable_Objects(c): List all the objects that can be accessed by subject c.
Input: A subject c.
Output: Returns the names of objects and their permissions.
(1) FOR each object o in the system
IF (Check_Write(c,o) returns True) THEN
Show that subject c has read and write permission for object o and display o.
ELSE
Show that subject c has read permission for object o and display o.
ELSE
Show that subject c has no access permission for object o.
END IF
END FOR
```

Figure 10: Algorithms for performing passive and active data access in the
CWSPL

5. IMPLEMENTATION AND EXPERIMENT

To demonstrate the feasibility of the proposed run-time system
for the CWSPL, we implemented the run-time system and
tested it in two Java-based WMMSs [19,20], with all
the execution codes implemented in the Java programming
language and each CWSPL command implemented as a Java
method. The source code of the corresponding Java programs
can be downloaded at http://www.csie.ntnu.edu.tw/~ghhwang/CWSPSL_Java.zip.
We call this software package CWSPL_Java. Figure 11
shows an example of invoking the CWPSL_Java package in Java.

```java
\begin{verbatim}
  // Create company information
  CompanyInformation ci1 = new CompanyInformation("CI_1");
  CoClass oilClass = new CoClass("Oil");
  CoClass bankClass = new CoClass("Bank");
  Company oil1 = new Company(oilClass, "O1");
  Company bank1 = new Company(bankClass, "B1");
  Company bank2 = new Company(bankClass, "B2");

  cis.add(ci1);
  cis.add(ci2);

  // Load company information
  CompanyInformation ci1 = ChineseWall.loadCompanyInformation(ci1File);
  CompanyInformation ci2 = ChineseWall.loadCompanyInformation(ci2File);

  // Create company information
  ci1.addDataObject(oil1, "O1_Data");
  ci2.addDataObject(b1_Obj);

  // Enforce
  ChineseWall.enforce(bind1);
  ChineseWall.enforce(bind2);

  // Field access
  ci1.addDataObject(o1_Obj);

  // Ignore
  CwsmBinding bind2 = ChineseWall.ignore(bind1, null, ignoreSubjects);

  // Add subjects
  subjects.add(john);
  subjects.add(mary);

  // Add company information
  cis.add(ci2);
  cis.add(ci1);

  // Add subjects
  subjects.add(john);
  subjects.add(John);

  // Ignore subjects
  ignoreSubjects.add(john);
  ignoreSubjects.add(john);

  // Enforce
  ChineseWall.enforce(bind1);

  // Add company information
  cis.add(ci1);
  cis.add(ci2);

  // Add subjects
  subjects.add(john);
  subjects.add(mary);

  // Ignore subjects
  ignoreSubjects.add(john);
  ignoreSubjects.add(mary);

  // Enforce
  ChineseWall.enforce(bind2);
\end{verbatim}
```

Figure 11: An example of a Java program that invokes the CWPSL_Java program.

Our preliminary implementation revealed that the CWPSL and the run-time system proposed in this paper can dramatically reduce the effort of specifying CWSM-related security policy. It also makes the debugging process much easier since the CWPSL codes are always concise and generic.

6. CONCLUSION AND FUTURE WORK

In this paper we have described how to implement the CWSM in a WfMS. The presented examples demonstrate that RBAC with dynamic role-subject and role-permission binding can be applied to implement the CWSM in certain situations. However, we also show that RBAC cannot handle general cases of the CWSM — the BN read rule can be simulated in RBAC with some limitations, whereas the BN write rule cannot be implemented in RBAC. We propose a model language that the system developer can use to specify the CWSM. Access control based on the CWSM can be integrated with the dynamic behavior of the workflow process. In addition to supporting the intrinsic dynamic access control mechanism defined in the CWSM (i.e., the dynamic binding of subjects and elements in the company data set), this language can specify several requirements of the dynamic security policy that arise when we want to apply the CWSM in WfMSs. The implementation and experiment demonstrate the feasibility of the proposed scheme.

REFERENCES


