Chapter 6

IMPLEMENTING WORKFLOW SYSTEMS

Lucas Dreyer and Martin Olivier

Abstract Workflow systems are becoming increasingly important as organisations automate their paper-based processes. Workflow systems make it possible to track processes and the responsibilities of individuals within these processes with relative ease. It is further possible to implement complex constraints including security and access control with workflow systems. Separation-of-duty is a security constraint that recently started to find its way into the workflow environment. A workflow model is presented in this paper that implements role-based access control and separation-of-duty. This model extends previous research by including semantics that can be used to implement the model. A prototype of the model has been implemented with Microsoft's SQL Server, the Distributed Component Object Model (DCOM) and Visual Basic.

Keywords: Workflow, access control, separation-of-duty, DCOM

1. Introduction

Workflow systems are becoming increasingly important as organisations use them to automate their internal processes [2]. With a paper-based process it is often difficult to determine how far it has progressed and who is responsible for the next steps. The result is that individuals can form a bottleneck in a paper-based process and tracking them down can be challenging in a complex organisation. In contrast to a paper-based system most workflow systems have the ability to quickly and graphically show what tasks are outstanding and who is responsible for them. Workflow systems can further help to reduce errors, which typically occur in paper-based processes.

A further advantage of workflow systems is that because of their high-level of automation they can support more complex constraints, including security, than what paper-based systems are typically capable of. For instance, a workflow system typically grants authorisations dynamically to users to enable them to perform their duties. These authorisations are then revoked after the users have completed their tasks. This type of dynamic constraint is difficult to im-

The original version of this chapter was revised: The copyright line was incorrect. This has been corrected. The Erratum to this chapter is available at DOI: 10.1007/978-0-387-35697-6_26

E. Gudes et al. (eds.), *Research Directions in Data and Applications Security*

plement, even with built-in security of modern operating systems such as role-based access control.

In addition to dynamically allocating authorisations to users, a workflow system must be able to support separation-of-duty [7]. Separation-of-duty constraints prevent users from committing fraud within the workflow environment. For instance, according to separation-of-duty a single person should not be able to prepare and approve a cheque. However, not many workflow systems provide declarative separation-of-duty constraints and these typically have to be coded on an ad-hoc basis. As a result separation-of-duty in workflow systems is currently an active research field [4].

This paper contains a description of a workflow model that implements dynamic access control as well as separation-of-duty. Emphasis is further placed on the practical implementation of this model. A prototype of the workflow model was implemented with Microsoft technologies including SQL Server, Microsoft's Distributed Component Object Model (DCOM) and Visual Basic [1, 6, 3].

The following section contains an overview of the workflow model. This will be followed by a description of the semantics of the workflow model. Next an implementation of a prototype workflow system with DCOM will be described. This paper will be concluded by some points for further research.

2. Workflow Model

The workflow model that is described in this paper is based on role-based access control [5, 8]. According to role-based access control a system Σ can be represented with the tuple Σ = <S, R, E, PR, A> where S is the set of subjects, R is the set of roles, E the set of entities, PR the set of privileges and A the set of authorisations. A = {<r, e, pr> | r $\in R$, e $\in E$, pr $\in PR$} where <r, e, pr> denotes that role r has privilege pr on entity e. Each subject is further assigned to a role and the currently assigned role of subject s is indicated by R (s) = r. The roles form a partial order where $r_1 \leq r_2$ and $r_1, r_2 \in R$ indicates that r_2 has all the access privileges of r_1.

Definition 1: Workflow sets
S = $\{s_1, s_2, ...\}$ is the set of subjects,
R = $\{r_1, r_2, ...\}$ is the set of roles,
E = $\{e_1, e_2, ...\}$ is set of entities,
PR = $\{pr_1, pr_2, ...\}$ is the set of privileges, and
RH: R $\times$ R is the role hierarchy, a partial order on R and written as $\leq$

The entities form a hierarchy and child entities are contained within parent entities. Parent and child relationships can be used to simplify security admin-

istration by making use of privilege inheritance. For instance, a role can be given access to a whole database, which can contain a number of entities such as tables, views, stored procedures etc. If a role must not get access to a whole container then access control can be made more specific to individual tables, views or stored procedures.

Entity hierarchies are indicated by the $E_{Hierarchy}$: E × E → B mapping where it is assumed that B is the set of Boolean values B = {true, false} from now on. The tuple $<e_1, e_2$, true$> \in E_{Hierarchy}$ where $e_1, e_2 \in$ E indicates that e_1 is the parent of e_2 and the privileges on e_1 propagate through to e_2. Similarly $<e_1, e_2$, false$> \in E_{Hierarchy}$ indicates that privileges do not propagate from e_1 to e_2. The set $E_{Parent}\ (e_1) = \{e_2 \mid (<e_2, e_1, b> \in E_{Hierarchy}) \wedge (b \in B)\}$ indicates the parent entities of e_1. Similarly the set $E_{Child}\ (e_1) = \{e_2 \mid (<e_1, e_2, b> \in E_{Hierarchy}) \wedge (b \in B)\}$ indicates the child entities of e_1.

Definition 2: Entity hierarchy

$E_{Hierarchy}$: E × E → B is the hierarchy of entities,

$E_{Parent}\ (e_1) = \{e_2 \mid (<e_2, e_1, b> \in E_{Hierarchy}) \wedge (b \in B)\}$ is the set of entity parents of an entity, and

$E_{Child}\ (e_1) = \{e_2 \mid (<e_1, e_2, b> \in E_{Hierarchy}) \wedge (b \in B)\}$ is the set of entity children of an entity

The definition of a system can be extended to denote the notion of a process. A process can be seen as a collection of operations that are executed within a system. For instance, an order process would typically consist of the submitting, approval and execution of the order. The processes in a system will be indicated with the set P = $\{p_1, p_2, ..., p_n\}$.

Processes in a system form a partial order. Two processes, therefore, execute either in parallel or the one process executes before the other one. The P_{Before}: P × P relation indicates the relative order between processes. The tuple $<p_1, p_2> \in P_{Before}$ where $p_1, p_2 \in$ P indicates that p_1 occurs before p_2. The set $P_{Before}\ (p_1) = \{p_2 \mid <p_2, p_1> \in P_{Before}\}$ indicates the processes before p_1. Similarly the set $P_{After}\ (p_1) = \{p_2 \mid <p_1, p_2> \in P_{Before}\}$ indicates the processes after p_1.

Definition 3: Process sets and relations

P = $\{p_1, p_2, ...\}$ is the set of processes,

P_{Before} : P × P is a relation that indicates the relative order of processes. It is a partial order that is written as ≤,

$P_{Before}\ (p_1) = \{p_2 \mid <p_2, p_1> \in P_{Before}\}$ is the set of processes that must occur before a specified process, and

$P_{After}\ (p_1) = \{p_2 \mid <p_1, p_2> \in P_{Before}\}$ is the set of processes that must occur after a specified process

As is the case with the entities in a system, the processes form a hierarchy. A process can, therefore, consist of a number of sub-processes. For instance, an order process can consist of sub-processes for submitting, approving and executing the order. An alternative to using a process hierarchy is to define a system recursively as consisting of sub-systems, where each sub-system can in turn consist of further sub-systems.

Process hierarchies, similarly to entity hierarchies, can be used to propagate access privileges from parent processes to child processes. A parent-child relationship between two processes must, therefore, be specified as propagating or non-propagating. Since processes form a partial order it is necessary to define the first and final (or terminating) sub-processes within a process. The parent-child relationships can be used for this purpose. The relation $P_{Hierarchy}$: P $\times$ P $\times$ B $\times$ B $\times$ B indicates parent-child relationships. The tuple $<p_1$, p_2, propagating: true, first: true, final: false$> \in P_{Hierarchy}$ indicates that p_2 is a child process of p_1 and the relationship is propagating. p_2 is further the first sub-process within p_1 but not a final process. The set $P_{Parent}\ (p_1) = \{p_2 \mid (<p_2, p_1, b_1, b_2, b_3> \in P_{Hierarchy}) \wedge (b_1, b_2, b_3 \in \text{B})\}$ indicates the parent processes of p_1. Similarly the set $P_{Child}\ (p_1) = \{p_2 \mid (<p_1, p_2, b_1, b_2, b_3> \in P_{Hierarchy}) \wedge (b_1, b_2, b_3 \in \text{B})\}$ indicates the child processes of p_1.

Definition 4: Process hierarchy sets and relations

$P_{Hierarchy}$: P $\times$ P $\times$ B $\times$ B $\times$ B is a relation that indicates the hierarchy of processes,

$P_{Parent}\ (p_1) = \{p_2 \mid (<p_2, p_1, b_1, b_2, b_3> \in P_{Hierarchy}) \wedge (b_1, b_2, b_3 \in \text{B})\}$ is a set that contains the parent processes of a specified process, and

$P_{Child}\ (p_1) = \{p_2 \mid (<p_1, p_2, b_1, b_2, b_3> \in P_{Hierarchy}) \wedge (b_1, b_2, b_3 \in \text{B})\}$ is a set that indicates the child processes of a specified process

Each process is a process type or an instance of a process type. The set P_{Type} indicates the process types in a system while P / P_{Type} contains the process instances. The relation $P_{Instance}$: P $\rightarrow 2^P$ maps each process type to its instances. The tuple $<p_1, p_2> \in P_{Instance}$ where $p_1 \in P_{Type}$ and $p_2 \in$ (P / P_{Type}) indicates that p_2 is an instance of p_1. The set $P_{Instance}\ (p_1) = \{p_2, p_3, ...\}$ where $p_1 \in P_{Type}$ and $p_2, p_3, ... \in$ (P / P_{Type}) indicates that $p_2, p_3, ...$ are instances of p_1. Conversely the set $P_{Type}\ (p_1) = p_2$ where $p_1 \in$ (P / P_{Type}) and $p_2 \in P_{Type}$ indicates that p_2 is the type of p_1.

Definition 5: Process type sets and relations
P_{Type} = $\{p_1, p_2, ...\}$ is the set of process types,
P_{Type} (p_1) = p_2 indicates the type of a process instance, and
$P_{Instance}$: P $\rightarrow$ 2^P is a relation that maps each process type to its instances

An operation or operations can be associated with a process where an operation represents a set of actions that must be executed within the context of the process. The structure of an operation will not be defined as part of the workflow model but will instead be left to the implementation of the workflow model. The set OP = $\{op_1, op_2, ...\}$ denotes the set of operations in a system. The set OP (p_i) denotes the operations for process p_i where $p_i \in$ P.

Definition 6: Operation sets
OP = $\{op_1, op_2, ...\}$ is the set of operations in a system, and
OP (p_i) = $\{op_1, op_2, ...\}$ is the set of operations associated with a process

When a process is created by a subject within a role, the subject is assigned to the process. It is necessary to know which exact subject created a process in order to implement separation-of-duty and other security constraints. The $P_{Subject}$: P $\rightarrow$ S function maps each process to the subject that created it. $P_{Subject}$ (p_1) = s_1 where $p_1 \in$ P and $s_1 \in$ S indicates that s_1 is assigned to p_1.

Authorisations are defined per process (as opposed to being system-wide as is the case with traditional role-based security). A further difference between access control in workflow and traditional access control is that privileges to entities as well as processes can be assigned to roles. Authorisations are indicated by the mapping A: P $\times$ R $\times$ (E $\cup$ $\{\oslash\}$) $\times$ (P $\cup$ $\{\oslash\}$) $\times$ PR. The tuple <p_1, r_1, e_1, $\oslash$, read> where $p_1 \in$ P, $r_1 \in$ R, $e_1 \in$ E and read $\in$ PR indicates that r_1 has within the context of p_1 read access to e_1. Note that '$\oslash$' means that the authorisation is not applicable to a process. Similarly the tuple <p_1, r_1, $\oslash$, p_2, create> where $p_1, p_2 \in$ P, $r_1 \in$ R and create $\in$ PR indicates that r_1 has within the context of p_1 create access to p_2.

Process constraints are further defined to implement security restrictions such as separation-of-duty. The mapping C: P $\times$ P $\times$ (Z $\cup$ $\{\oslash\}$) $\times$ B denotes the process constraints. Note that Z represents the set of integers. The tuple <p_1, p_2, 0, true> $\in$ C where $p_1, p_2 \in$ P indicates that p_1 and p_2 on the same level (hence the level difference between them is 0 and they, therefore, have the same parents) must have the same subject (as is indicated by true). The tuple <p_1, p_2, -1, false> $\in$ C where $p_1, p_2 \in$ P indicates that for p_2 on one level higher than p_1 (hence p_2 is a parent of p_1: $p_2 \in P_{Parent}$ (p_1)) the subjects that created p_1 and p_2 must be different. A level difference of 1 means that

p_2 is a child of p_1. Finally, the tuple $<p_1, p_2, \oslash$, false$>$ $\in$ C where $p_1, p_2 \in$ P indicates that the same subject cannot be associated with p_1 and p_2 in the whole system.

Definition 7: Authorisations and constraints

$P_{Subject}$: P $\rightarrow$ S is a function that maps each process to the subject that created it,

$P_{Subject}$ $(p_1) = s_1$ indicates the subject that has created p_1,

A: P $\times$ R $\times$ (E $\cup$ $\{\oslash\}$) $\times$ (P $\cup$ $\{\oslash\}$) $\times$ PR is a relation that indicates the authorisations that roles have on processes or entities within the context of a set of processes, and

C: P $\times$ P $\times$ (Z $\cup$ $\{\oslash\}$) $\times$ B is a relation that indicates the process constraints. Processes that are separated by a specific distance in the process hierarchy (for instance, siblings or parents and children) can be forced to be created by the same or different subjects

A process starts its life as a new process and as subsequent processes are created (as indicated by the precedence relation) the process will change to an old process. A new process is further incomplete until its associated operations and sub-processes are completed. The creation of processes and their change of state will be discussed in more detail in the following section. The set P_{New} indicates which processes are new. For instance, $p_1 \in P_{New}$ where $p_1 \in$ P indicates that p_1 is new. The set $P_{Completed}$ indicates which processes are completed.

Definition 8: Process lifetime

$P_{New} = \{p_1, p_2, ...\}$ is a set that indicates the new processes in a workflow system, and

$P_{Completed} = \{p_1, p_2, ...\}$ is a set that indicates the completed processes in a workflow system

A system consisting of processes can be represented by the tuple Σ = <S, E, $E_{Hierarchy}$, R, P, OP, P_{Type}, $P_{Instance}$ P_{Before}, $P_{Hierarchy}$, PR, A, C> where S is the set of subjects, E is the set of entities, $E_{Hierarchy}$ maps entities to their children, R is the set of roles, P is the set of processes, OP is the set of operations, P_{Type} is the set of process types, $P_{Instance}$ maps process types to their instances, P_{Before} maps processes to their preceding and succeeding processes, $P_{Hierarchy}$ maps processes to their children, PR is the set of access privileges, A is the set of authorisations and C is the set of process constraints. Note that the sets P_{New}, $P_{Completed}$ and $P_{Subject}$ are maintained for the life-time of the system although they do not form part of the system specification.

The structure of a workflow model was described in this section. An example will be used in the following section to illustrate the semantics of the workflow model.

3. Semantics

A cheque-order example will be used in this section to illustrate the semantics of the workflow model that was introduced above. Assume that there are clerks and managers in an organisation and clerks prepare cheques, managers approve the cheques and clerks issue the cheques. The roles are, therefore, R = {clerk, manager}.

It will be assumed in this example that a cheque is represented by a process type. The clerk role will then have the create privilege on the cheque process type. Within the cheque process type will be sub-process types for the approval and issuing of cheques. The manager and clerk roles will have create privileges on these types respectively. A computer system is further assumed to consist of a global process within which all the workflow processes will be created. This global process will be denoted by χ with type X.

R = {clerk, manager}, S = {John, Sarah, James}
R (John) = clerk, R (Sarah) = manager, R (James) = clerk
P_{Type} = {cheque, approve, issue, X}, $P_{Instance}$ (X) = {χ}
$P_{Hierarchy}$ = {
<X, cheque, propagating: false, first: true, final: false>,
<X, cheque, propagating: false, first: false, final: false>,
<cheque, approve, propagating: false, first: true, final: false>,
<cheque, issue, propagating: false, first: false, final: true>},
P_{Before} = {<approve, issue>}, PR = {create}
A = {
<X, clerk, $\oslash$, cheque, create>,
<cheque, manager, $\oslash$, approve, create>,
<cheque, clerk, $\oslash$, issue, create>}
C = {
<cheque, approve, 1, false>,
<cheque, issue, 1, false>,
<approve, issue, 0, false>}

The above specifies that the roles are clerk and manager. John and James are clerks while Sarah is a manager. 'cheque', 'approve' and 'issue' are process types. 'approve' and 'issue' are further child types of 'cheque' and the privileges on cheque do not propagate from 'cheque' to 'approve' and 'issue'. 'approve' is the first child of 'cheque' and 'issue' is the last child of 'cheque'.

Finally, instances of 'approve' must be created and completed before instances of 'issue' can be created.

The only privilege that will be used in this example is 'create' (creation of processes). The authorisations indicate that within the main process type (X) a clerk can create a cheque process. Within an instance of a cheque process type a manager can create an 'approve' sub-process and a clerk can create an 'issue' sub-process. The process constraints specify that the subject who created a cheque process cannot approve or issue it. The subject that approves a cheque can further not issue it.

The creation of processes within the workflow system will now be described. Initially only the main process χ exists and this process is a new process: P = $\{\chi\}$ and P_{New} = $\{\chi\}$. A clerk (for instance, John) can now create a new instance of a cheque process (called $cheque_1$): P = $\{\chi, cheque_1\}$, $P_{Hierarchy}$ = $\{<\chi, cheque_1$, propagating: false, first: true, final: false>$\}$, P_{New} = $\{\chi, cheque_1\}$ and $P_{Subject}$ = $\{<cheque_1$, John>$\}$. Within a cheque instance the manager role is authorised to approve the cheque and this is done by creating an 'approve' sub-process within a cheque. Note that other instances of 'cheque' can be created in the mean time since 'cheque' is marked as both a first child of χ (to start the process of creating cheque instances) as well as a normal child.

Only an 'approve' sub-process can be created within the cheque process since 'approve' is the first child of 'cheque'. Assume that Sarah approves the cheque: P = $\{\chi, cheque_1, approve_1\}$, $P_{Hierarchy}$ = $\{<\chi, cheque_1$, propagating: false, first: true, final: false>, $<cheque_1, approve_1$, propagating: false, first: true, final: false>$\}$, P_{New} = $\{\chi, cheque_1, approve_1\}$, $P_{Completed}$ = $\{approve_1\}$ and $P_{Subject}$ = $\{<cheque_1$, John>, $<approve_1$, Sarah>$\}$. The $approve_1$ process is marked as completed because 'approve' does not have any children.

The final step is to issue $cheque_1$. The only person who can now issue $cheque_1$ is James since John and Sarah have already participated in the process. Note that $approve_1$ was marked as a new as well as completed process above. According to the semantics of the workflow model a process that is preceded by other processes (as is indicated by P_{Before}) can only be created if all those processes are new and completed. In this example an instance of 'issue' can be created because it is preceded by $approve_1$, which is new and completed. The result is: P = $\{\chi, cheque_1, approve_1, issue_1\}$, $P_{Hierarchy}$ = $\{<\chi, cheque_1$, propagating: false, first: true, final: false>, $<cheque_1, approve_1$, propagating: false, first: true, final: false>, $<cheque_1, issue_1$, propagating: false, first: false, final: true>$\}$, P_{New} = $\{\chi, cheque_1, issue_1\}$, $P_{Completed}$ = $\{approve_1, issue_1, cheque_1\}$ and $P_{Subject}$ = $\{<cheque_1$, John>, $<approve_1$, Sarah>, $<issue_1$, James>$\}$.

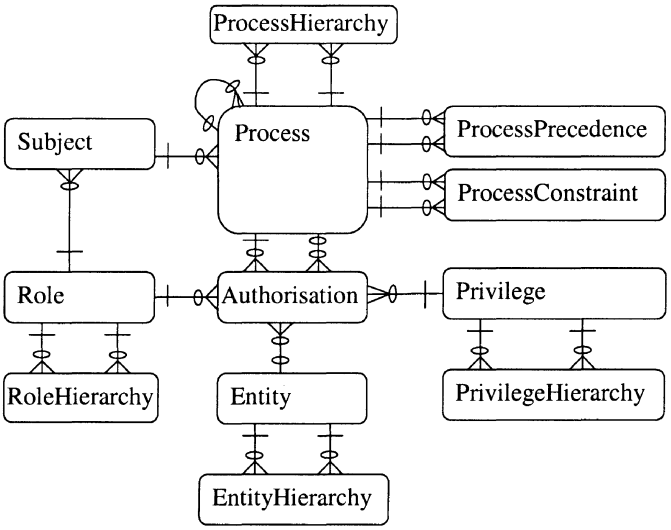

Figure 1. Data model of the data storage layer.

Note that $cheque_1$ is now completed because it was approved and issued. '$approve_1$' is further not new ('$issue_1$' was created after it) and $issue_1$ is new.

A bonus of the workflow model that was presented here is that the information above can be used as an audit trail of the actions that occurred in the system as well as who performed them. This concludes the discussion of the structure and semantics of the workflow model.

The following section contains a description of how DCOM and SQL server were used to implement the workflow model.

4. Workflow Implementation with DCOM

The workflow model was implemented by making use of a three-tiered approach [6]. According to this approach the functionality of a system is divided into three layers: a data storage, business logic and presentation layer. The data storage layer for the workflow model was developed in SQL Server. A data model was used whereby most of the tables correspond to the mappings and sets of Section 2. Figure 1 depicts the data model of the workflow database.

The business layer has been implemented as a DCOM component in Windows NT. The DCOM component was developed in C++ and uses Active Data Objects (ADO) to access the workflow database. The workflow component consists of a class called 'WorkflowEngine' that implements the workflow engine. It supports an interface called 'IWorkflow' with the following methods:

GetEntity, CreateEntity, CreateProcess and GetTask. Figure 2 shows the IDL (Interface Definition Language) definition of these methods.

```
HRESULT GetEntity([in] long Subject, [in] BSTR EntityType, [in] VARIANT Id, [in]
VARIANT Parent, [in] VARIANT IsType, [in] VARIANT TypeId, [in] VARIANT Completed,
[out] _Recordset** Entities, [out] long* Erc);

HRESULT CreateEntity([in] long Subject, [in] BSTR EntityType, [in] VARIANT Parent, [in]
BSTR Description, [out] long* NewEntity, [out] long* Erc);

HRESULT DeleteEntity([in] long Subject, [in] long Id, [in] BSTR EntityType, [out] long* Erc);

HRESULT CreateProcess([in] long Subject, [in] long ProcessType, [in] long Process, [in]
BSTR Description, [out] long* NewProcess, [out] long* Erc);

HRESULT GetTask([in] long Subject, [in] long Process, [in] VARIANT ProcessType, [out]
_Recordset** Task, [out] long* Erc);
```

Figure 2. Workflow DCOM component methods.

In order to implement the functionality of the workflow model, the workflow component interacts with the data storage layer (the SQL Server database), the operating system and the presentation layer. These interactions will be briefly described below.

The GetEntity method is used to retrieve information about all the tables in the database such as a process, entity, privilege etc. CreateEntity is used to create new records in the tables of the database such as new processes and entities while DeleteEntity is used to delete records. CreateProcess is used to create an instance of a process type within a process instance according to the semantics that were described in Section 3.

GetTask is used to retrieve the process types that are applicable to a specific process instance. For instance, for a newly created cheque process GetTask will retrieve the approve process type. The process types returned by GetTask for a process will change as sub-processes are created and completed within the process.

The workflow engine interfaces with the operating system to implement authorisations according to the workflow process. The Win32 API functions GetSecurityInfo, SetEntriesInAcl and SetSecurityInfo are used to grant privileges to and revoke privileges from operating system entities such as files [6]. The workflow engine of the prototype is, therefore, able to permit operating system users to access operating system entities in specific processes and later revoke their access as these processes complete.

In order to implement access control on the workflow component level it is necessary to know which subjects are calling the methods of the workflow

component. The workflow component's object context can be used for this purpose and it can be retrieved with the component's GetObjectContext method. The GetDirectCallerSID method of the object context's security property is used to retrieve the identifier of the user who called the component. Note that the object context differs across Windows NT and Windows 2000, making it difficult to deploy DCOM applications across both these operating systems simultaneously.

The presentation layer of the workflow prototype consists of a rudimentary Visual Basic front-end through which administrators and users can interact with the workflow engine. Users can use the front-end to view and select processes and to create sub-processes within the currently selected process.

This concludes the discussion of the implementation of the workflow prototype.

5. Conclusions

A workflow model was introduced in this paper that supports role-based access control and separation-of-duty. This model includes the semantics for the actual implementation of the model. A prototype of the workflow model was further implemented with Microsoft's SQL Server, DCOM and Visual Basic.

Further research is needed to refine the semantics of the workflow model. Currently the workflow model does not support the routing of work along different types of topologies such as sequential, parallel, circular and alternative routes, as is already supported by many commercial systems. Research is further needed for a more concise definition of the workflow model by making use of models such as process calculus and Turing Machines. Such a definition will make it possible to combine additional security requirements such as information flow with workflow.

References

[1] D. Box, K. Brown, T. Ewald and C. Sells, *Effective COM 50 Ways to Improve Your COM and MTS-based Applications*. Addison-Wesley, Reading, Massachusetts, 1999.

[2] L. Fischer (ed.), *The Workflow Handbook 2001*, published in association with the Workflow Management Coalition (WfMC) [9], Future Strategies, Lighthouse Point, Florida, 2000.

[3] R. Grimes, *Visual C++ 6 MTS Programming*, Wrox, Birmingham, U.K., 1999.

[4] W. Huang and V. Atluri, Analyzing the safety of workflow authorization models, in S. Jajodia (ed.), *Database Security, XII: Status and Prospects*, Kluwer, Dordrecht, The Netherlands, pp. 43-57, 1999.

[5] L.G. Lawrence, The role of roles, *Computers & Security*, Vol. 12(1), pp. 15-21, 1993.

[6] Microsoft Corporation, *Microsoft Developers Network (MSDN)*, Redmond, Washington, 2001.

[7] R.S. Sandhu, Separation of duties in computerized information systems, in S. Jajodia and C. Landwehr (eds.), *Database Security, IV: Status and Prospects*, North Holland, Amsterdam, The Netherlands, pp. 179-189, 1991.

[8] R.S. Sandhu, E.J. Coyne, H.L. Fernstein and C.E. Youman, Role-based access control models, *IEEE Computer*, Vol. 29(2), pp. 38-47, 1996.

[9] Workflow Management Consortium (Wfmc) http://www.wfmc.org.