A Strategy for an MLS Workflow Management System

Myong H. Kang, Judith N. Froscher, Brian J. Eppinger, and Ira S. Moskowitz
Information Technology Division
Naval Research Laboratory

{mkang, froscher, eppinger, moskowitz}@itd.nrl.navy.mil

Abstract

Current DoD information systems need to support many different missions through cooperation with different organizations and allies. In today's fast paced and dynamic environment, it is almost impossible to design and implement a different information system for each mission. Therefore, DoD needs MLS workflow management systems (WFMS) to enable globally distributed users and existing applications to cooperate across classification domains to achieve mission critical goals. An MLS WFMS that allows users to program multilevel mission logic, securely coordinate widely distributed tasks, and monitor the progress of the workflow across classification domains is required. In this paper, we present requirements for MLS workflow and a strategy for implementing it, especially the method for decomposing an MLS workflow into multiple single-level workflows

1. Introduction

The streamlining of today's business processes has brought about significant increases in productivity and the ability for US companies to compete in the global market place. These re-engineering activities have as their basis an even greater reliance on information technology and automation. As a result, software developers have been challenged to streamline the software development process and to produce software that can be reused, that separates concerns, that supports autonomy and heterogeneity in a distributed computing environment, that allows extensibility, etc. The information technology (IT) community has developed distributed object computing standards, like CORBA and DCOM, that provide a basic level of interoperability among distributed applications. The next step is to build application specific software architectures that encode business logic for coordinating widely distributed manual and automated tasks in achieving enterprise level objectives. To assist business process modeling, vendors have developed several automated tools that help manage dependence among activities and users. A WFMS makes these tools available to users and allows them to monitor the business processes. This technology manages activities within a distributed computing environment comprising of loosely coupled, heterogeneous, autonomous, new (and legacy) components which may include transactional systems (i.e., DBMS). It provides a capability for defining the:

- ♦ Business logic,
- ♦ Tasks that make up business processes, and
- ♦ Control flow and data dependence among those tasks.

The potential benefits of this technology are enormous because of its broad reach to manage business process productivity. Industry is beginning to turn to workflow technology in order to minimize its manpower needs, optimize IT investment, achieve good performance, use legacy systems, gain flexibility for supporting evolving business goals, as well as to capitalize on advanced technology. However, commercial WFMS do not support distributed mission critical applications. They do not ensure mission critical properties, such as recoverability nor do they enforce access control policies, and certainly not multilevel secure (MLS) access control policies. Even though there is a great need for this technology in DoD, DoD cannot rely on commercial WFMS to protect national security information and perform mission critical business processes.

Report Documentation Page	*Form Approved* *OMB No. 0704-0188*

1. REPORT DATE **1999**	2. REPORT TYPE	3. DATES COVERED **00-00-1999 to 00-00-1999**

4. TITLE AND SUBTITLE **A Strategy for an MLS Workflow Management System**	5a. CONTRACT NUMBER
	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **Naval Research Laboratory,Information Technology Division,4555 Overlook Avenue, SW,Washington,DC,20375**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES

14. ABSTRACT

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES **15**	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**			

The constant aspect of the military challenge is change. The challenge is to respond to new threats in completely different environments. For example, today's military supports disaster relief, drug interdiction, peace-keeping missions in worldwide regional skirmishes, treaty enforcement, as well as the traditional role of national defense against aggression and weapons of mass destruction. At no other time in the nation's history has the military relied so heavily on IT systems for all of its operations, including command and control, logistics, surveillance and reconnaissance, personnel management, finances, etc. Challenging requirements of those systems include,

- The organizations that participate in a dynamic coalition may be located in different classification domains.
- The guidelines for sharing and exchanging information among organizations in different classification domains are stricter than those for organizations in the same classification domain.

To address those problems, the Naval Research Laboratory (NRL) has embarked upon a research project to build an MLS WFMS. The goal of the project is to develop tools and security critical components that allow enterprises to harvest emerging commercial off-the-shelf (COTS) technology and still rely on legacy resources with reduced risk.

In short, MLS WFMS should support:

- Secure interoperability among tasks that reside in different classification domains, and
- Maximum use of commercial software/hardware.

The need for a WFMS that can manage MLS workflows is immediate and imposes an implementation strategy that allows operational users to exploit advances in COTS technology and also to satisfy their mission critical requirements. The multiple single-level architecture, described in [6,7,12], provides the foundation for enforcing information flow requirements. A WFMS comprises several tools and runtime enactment services. This paper examines what properties these components must satisfy in a multiple single-level distributed computing environment. The MLS workflow design tool is of particular interest because the commercial tool must be significantly changed to represent classification domains. While this does not fit the paradigm of using unmodified COTS products with high assurance security devices, finding a research team that is developing a WFMS that supports mission critical workflows makes it possible to develop an MLS WFMS.

In this paper, we present requirements for MLS workflows, tools for supporting MLS workflows, and a strategy for implementing them. We also examine an MLS workflow model that supports MLS cooperation, and describe the decomposition of an MLS workflow into multiple single-level workflows.

2. Tools to Support MLS Workflow

A WFMS consists of, in general, three components:

- Workflow design (build-time) tool,
- Workflow enactment (runtime) service, and
- Monitoring tool.

A workflow design tool is a distributed programming tool with a graphical user interface. Users can express data dependence and control flow among tasks using this tool. Once a user specifies the mission logic (i.e., distributed programming logic), code for workflow enactment can be generated. A workflow enactment service is responsible for task scheduling, enforcing dependence among tasks, passing data from one task to another, and error recovery based on the generated code. The workflow monitoring tool, in general, tracks and monitors the progress of execution.

DoD needs all the tools that single-level WFMS provides. However, DoD requires extra capabilities in those tools to support MLS workflow. We will examine the extra requirements for each tool.

Design Tool for MLS Workflow

A workflow design tool is a distributed programming tool with a graphical user interface that provides the global picture of the whole mission. MLS workflow designers should be able to specify (1) tasks in different classification domains and (2) data and control dependence (flow) among them. Based on this workflow design, a specification for workflow runtime can be generated. Final runtime code that will be securely executed on an MLS distributed architecture is generated, based on this specification. The runtime specification and code generation processes, in general, depend on the underlying MLS distributed architecture.

In MLS applications, each subtask may be located in a different classification domain. The design tools for single-level workflow do not provide a capability to specify classification domains or compartments. In other words, the whole drawing area of the workflow design tool belongs to one classification domain. It also does not generate runtime code that can be executed on the underlying MLS distributed architecture. What is needed is a design tool for MLS workflow that:

- Allows MLS workflow designers to divide a design area into multiple domains,
- Allows MLS workflow designers to specify information flow and dependence among tasks that are in the same or different domains, and
- Allows MLS workflow designers to specify dominance relationships among domains (e.g., Top Secret > Secret > Unclassified).

For example, a workflow designer should be able to specify an MLS workflow as in Figure 1 where ovals represent tasks and arrows represent control and data flow. In the figure, B (begin), S (success), and F (failure) represent special tasks.

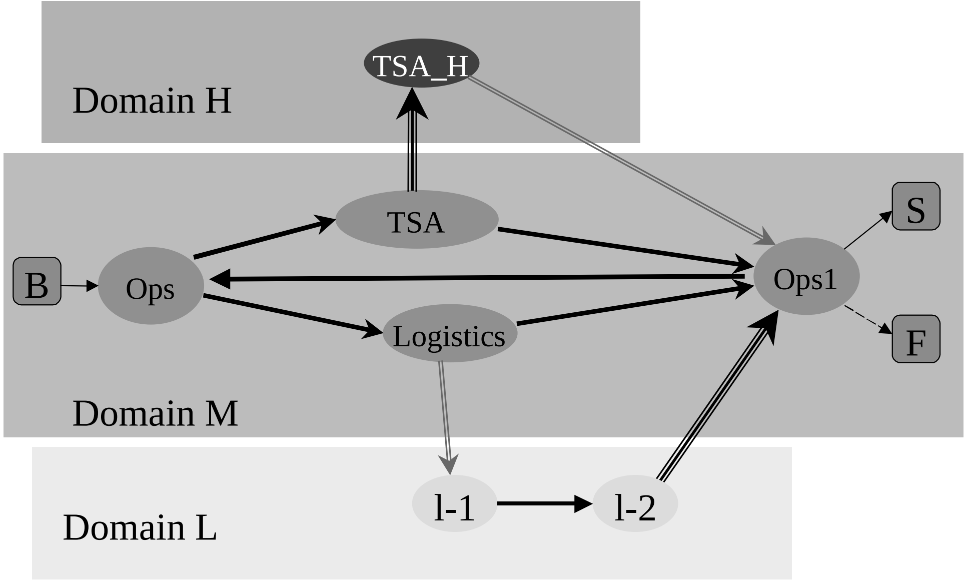

Figure 1: An Example of an MLS Workflow Design

Even though this tool allows workflow designers to specify information and control flow among tasks in different domains, the operational environment of the tool will be system-high (i.e., the workflow design tool neither accesses sensitive data in multiple domains nor passes it around). Hence, although this tool has to be trusted in the sense that it does what it is supposed to do, it can be run on a single-level system.

MLS workflow has another functional requirement. When an MLS workflow is designed, it must often interact with tasks in other domains about which the designer is not allowed to know the details. For example, when a secret workflow designer designs a workflow, the secret workflow may need to send a request to a top secret task. The secret designer is not allowed to know how the top secret task gets the answer, but he knows how to send a request and how to receive an answer. Hence, the top-secret task, in this case, is a task foreign to the secret designer (i.e., the top secret task does not belong to his workflow). In fact, the H workflow in figure 1 may be designed in a completely different organization from the M workflow and the L workflow. A workflow design tool for MLS workflow should be equipped with the capability to model interactions with a task for which only its interface specification is known. We believe this capability to model foreign tasks has broader application, even for single-level workflows (e.g., two cooperative workflows run by two different organizations).

Enactment Service for MLS Workflow

An MLS workflow enactment service is responsible for executing a workflow in a correct and secure manner. Hence, it depends on services in the underlying MLS distributed architecture to coordinate information and control flow among tasks in different classification domains. What we need is to make secure use of single-level COTS workflow enactment services with or without modifications. As we will explain in section 3, we plan to use multiple single-level workflow enactment services to execute an MLS workflow. Since there will be no direct communication among workflow enactment services at different classification domains, there are no special MLS requirements for a workflow enactment service itself. On the other hand, the underlying MLS distributed architecture and its security devices must provide the necessary assurance for multilevel security.

Monitoring Tool for MLS Workflow

When MLS workflow is executed, there are many automatic and human computer tasks that are executed in different classification domains. Workflow managers in different classification domains (there may be a workflow manager per classification domain) may have knowledge about tasks in their classification domain and other domains they are authorized to access. In other words, users of MLS workflow in different classification domains may have different views of the workflow they are running. Hence, an MLS WFMS should provide the ability to monitor activities in all domains the workflow manager is authorized to access. Monitoring may include when, where, and who performs the tasks in the case of human tasks. Because the expected, legal behavior of a workflow is specified, the workflow monitor can be designed to detect security critical events as well as unexpected behavior. Additionally, responses for security exceptions can be specified as part of the workflow design.

3. A Strategy for MLS Workflow

An MLS WFMS should support functionality equivalent to a single-level WFMS from the perspective of MLS users who design and use multilevel workflows. Tasks that may be single-level individually but located in different classification domains, have to cooperate to achieve a higher level MLS mission.

To provide MLS services in a distributed and heterogeneous computing environment, the following information flow requirements must be enforced:

- High users must have access to low data and low resources,
- High processes must have access to low data, and
- High data must not leak to low systems or users.

An MLS WFMS should obey this MLS policy. Atluri *et. al.* investigated MLS workflow in general [13, 14]. There are two basic ways to enforce the MLS policy in MLS workflow systems:

- Build high-assurance MLS WFMS that will run on an MLS platform, or
- Build an MLS workflow by integrating multiple single-level workflows with an MLS distributed architecture.

The development of high-assurance software, necessary to provide separation between unclassified and TS/SCI information, such as MLS workflow systems, has proven to be both technically challenging and expensive. Today's fast paced advances in technology and the need to use COTS products make the traditional MLS approach untenable. Therefore, we have chosen the second approach for building MLS WFMS. It is more in line with the modern distributed computing paradigm than the first approach in terms of supporting autonomy and heterogeneity.

To implement an MLS WFMS using the architectural method, the following technical approach has been established:

- Choose an MLS distributed architecture where multiple single-level workflows can be executed.
- Choose a strategy for dividing an MLS workflow into multiple single-level workflows.
- Choose a single-level WFMS to execute single-level workflow in each classification domain.
- Implement the necessary tools for supporting MLS workflow.
- Extend the workflow interoperability model to accommodate the communication among workflows at different classification domains.
- Extend the single-level workflow enactment service to accommodate communication among tasks in different classification domains.

MLS Distributed Architecture

Composing an MLS workflow from multiple single-level workflows is the only practical way to construct a high-assurance MLS WFMS today. In this approach, the multilevel security of our MLS workflow does not depend on single-level WFMS but rather on the underlying MLS distributed architecture. Thomas and Sandhu have proposed task-based authorization for single-level workflows [15]. The MLS distributed architecture will:

- Host multiple single-level workflows to be executed and
- Provide conduits for passing information among tasks in different classification domains.

Our MLS distributed architecture is based on a security engineering philosophy: a few trusted devices in conjunction with information release and receive policy servers enforce the information flow policy of the classification domains, and single-level systems and single-level engineering solutions provide other functionality. A generic MLS distributed architecture is shown in Figure 2.

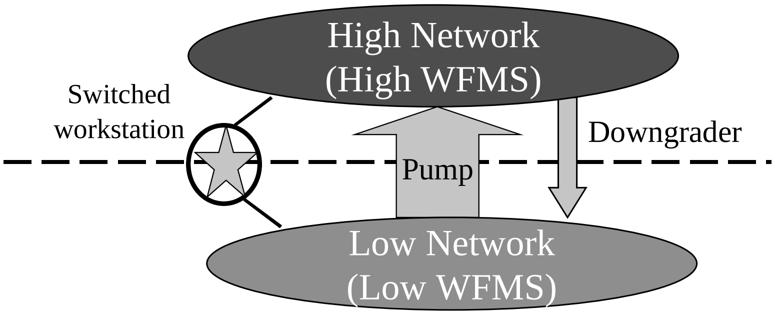

Figure 2: An MLS Distributed Architecture

In this architecture, switched workstations (e.g., "Starlight" [1]) enable a user to access resources in multiple classification domains and create information in domains that the user is authorized to access. One-way devices (e.g., a flow controller such as "A Network Pump" [5]) together with information release and receive policy servers provide a secure way to pass information from one classification domain to another. An information release policy server resides in a classification domain where the information is released, and an information receive policy server provides a secure way to pass information from one classification domain to another as shown in Figure 3.

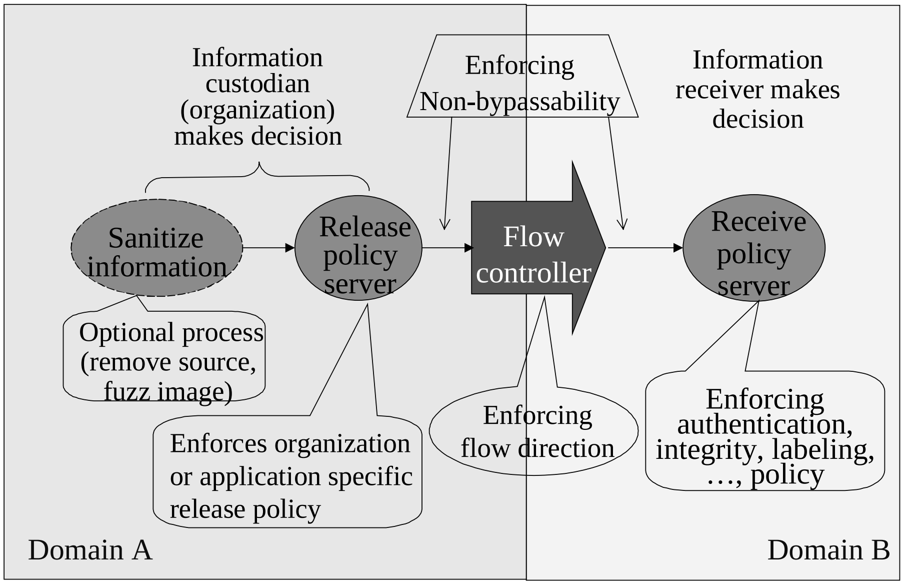
Information custodian (organization) makes decision
Enforcing Non-bypassability
Information receiver makes decision
Sanitize information
Release policy server
Flow controller
Receive policy server
Optional process (remove source, fuzz image)
Enforces organization or application specific release policy
Enforcing flow direction
Enforcing authentication, integrity, labeling, …, policy
Domain A
Domain B

Figure 3: Information Release and Receive Policies in Conjunction with a Flow Controller

In general, when COTS software passes data to a flow controller a *wrapper* translates the protocol of COTS software to that of the flow controller because COTS software and flow controllers communicate with other software through their own protocols. Hence, a wrapper can be considered a protocol translator. The detailed description of the architecture and the MLS services are in [7].

Workflow Interoperability

As we mentioned earlier, our strategy for implementing an MLS workflow is through combining single-level workflows on an MLS distributed architecture. Workflows in different domains may be heterogeneous and autonomous. Hence workflow interoperability is an important requirement for the approach that we have taken to implement an MLS workflow. Two important aspects of workflow interoperability are:

- Interoperability protocol among independent WMFS.
- The ability to model interoperability in a workflow process definition tool (i.e., workflow designer).

A standard body such as Object Management Group (OMG) can handle the first aspect (e.g., jFlow [4]). However, the second aspect should be handled by each WFMS.

OMG's jFlow introduces two models of interoperability. They are nested sub-process and chained processes as shown in Figure 4-a and 4-b respectively.

In nested sub-process workflow structures, a task in workflow A may invoke workflow B as the performer of a task and then wait for it to finish. Hence, the task in workflow A is a requester, and the task that is realized by the sub-processes can serve as the synchronization point [11] for interaction of the two workflows.

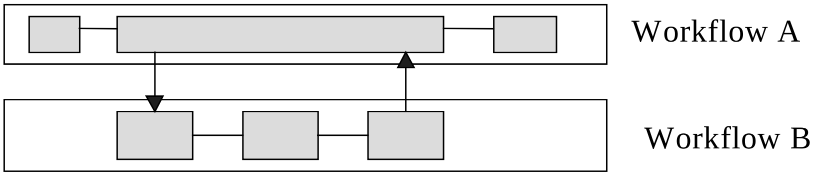

Figure 4-a: Nested Sub-Process

In chained workflow structures, one task may invoke another, then carry on with its own business logic. The workflows terminate independently of each other; in this case, the task registered with the sub-process would be another entity that is interested in the results of the sub-process.

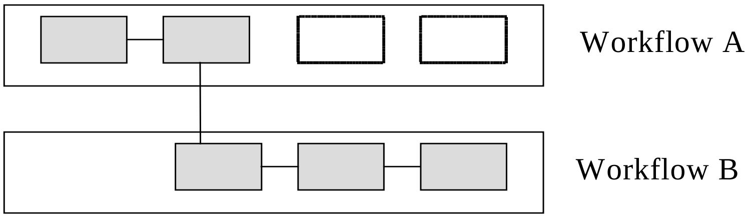

Figure 4-b: Chained Processes

The above two models provide powerful mechanisms for interoperability. However, we would like to extend these models to support an additional interoperability model, *cooperative processes*. Consider two independent autonomous workflows that need to cooperate. Let us assume that there are two cooperating organizations. Organization A is in charge of workflow A and Organization B is in charge of workflow B. Tasks in workflow A and workflow B can communicate and synchronize with each other as shown in Figure 5. In this example, two workflows may have independent starting and ending points.

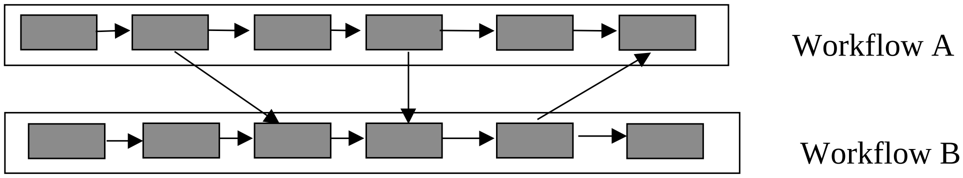

Figure 5: Cooperative Processes

There is another situation that we want to support in the context of cooperative processes. In general, the designer of workflow A does not need to know the structure of workflow B and vice versa. In conjunction with MLS principles, the designer of a workflow may not be allowed to know the detailed workflow structure of a higher level workflow. For example, in Figure 1, the designer of the workflow, whose classification domain is M, may not be allowed to know the workflow structure that contains the TSA_H task.

However, there is a minimal set of information that is required for communication and synchronization among tasks in cooperating autonomous workflows. This includes:

♦ Where and how to send/receive requests (i.e., the location and invocation method of tasks) and
♦ How and where to receive replies (i.e., expected output and the return location).

Therefore, the above specification has to appear in the workflow design so that the proper runtime code can be generated. Hence, we need a primitive that represents this situation in the design tool. This is the reason that we have introduced foreign tasks in the workflow model (section 4).

4. An MLS Workflow Model

Our strategy for implementing an MLS workflow is to combine single-level workflows on an MLS distributed architecture. We have chosen the METEOR WFMS [2, 3, 8, 10] as our single-level WFMS because it is a CORBA compliant, recoverable, and distributed WFMS (i.e., ORBWork [10] is a specific version of METEOR). METEOR also supports legacy tasks. It is an important feature for DoD because DoD has legacy applications that are costly to replace all at once. Hence, METEOR is a good starting point for extending capabilities to support MLS workflow.

To accommodate MLS workflow, the METEOR model has been modified. We summarize only the small subset of the new MLS METEOR model necessary for understanding this paper. A detailed description of the METEOR model can be found in [11].

In the METEOR model, a *task* represents an abstraction of an activity. A task can be regarded as a unit of work which is performed by a variety of processing entities, depending on the nature of the task. A task can be performed by (*realized by*) a human or by a computerized activity that executes a computer program, a database transaction, or possibly a network (workflow or subworkflow) of interconnected tasks. Hence, a task provides one level of abstraction (view) and its realization provides a lower level of abstraction (view). This also directly maps to the nested sub-process concept of jFlow. Since the realization of a task may contain many tasks at different levels of abstraction, a task is a recursive reference in the METEOR model.

In this paper, we categorize tasks into two types:

- *Foreign task:* A task whose realization (i.e., strategy for implementation) is unknown to the workflow designer. It represents a task that is a part of cooperating independent autonomous workflow. It is required for a designer to declare a foreign task explicitly to provide a hint to the METEOR runtime code generator. A foreign task should have a minimal information set that we specified in section 3 (e.g., invocation, output, where to send the request).
- *Native task:* A task for which the realization is known or the realization will be provided before the runtime code generation (i.e., all other tasks except the foreign tasks).

A *network task* represents the core of the workflow activity specification. Since a network task is one of the realizations of a task, it is always associated with a task called its *parent task*. A single network of tasks defines a relationship among workflow tasks, transferred data, exception handling, and other relevant information. It is a collection of either foreign or native tasks and transitions from one task to another. Figure 6 shows a simplified version of two levels of abstractions (views) where Task2 is the parent task of the projected workflow W_i which contains tasks 4, 5, 6, and 7, and transition t_j represents a transition from Task1 to Task2. In Figure 6, Task1, Task2, and Task3 may belong to different classification domains. Hence, the MLS METEOR model can be thought of as follows: along the *xy*-plane, there are tasks in different domains and along the *z*-axis, there are different levels of abstraction.

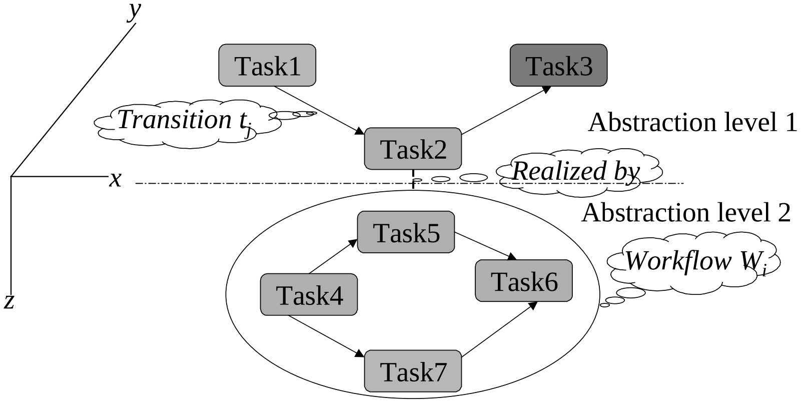

Figure 6: MLS METEOR Model

A task may play the role of a source task or a destination task (e.g., Task1 is the source task and Task2 is the destination task of the transition t_j in Figure 6) for a number of transitions. All of the transitions for which a task is the destination task are called the *input transitions* for that task (e.g., transition t_j is an input transition for Task2). Likewise, all the transitions for which a task is the source task are called its *output transitions* (e.g., transition t_j is an output transition of Task1). A transition may have an associated Boolean condition called its *guard*. A transition may be activated only if its guard is true. When there is a transition from task T_i to task T_j, where T_i and T_j are in different classification domains, we call this an MLS *transition* from T_i to T_j .

An *external transition* is a special type of a transition in which the two participating tasks (source and destination) are not in the same workflow (i.e., transition to and from a foreign task). An implied external transition may lead to a start task of another workflow. Similarly, an implied transition leads from the final task and is used to notify the external entity that the network has terminated. Note that an MLS transition is turned into an external transition when an MLS design is divided into multiple single-level workflows for runtime.

External transitions are also used to specify synchronization points with some external events. Typically, external transitions may be used to specify communication and synchronization between two independent workflows. Here, an external transition leading into a task in the workflow is assumed to have an implied source task (outside the workflow). Similarly, an external transition leading out of a task in the workflow is considered to have an implied destination task (outside the workflow). External transition is a cornerstone of our strategy to support MLS workflow.

The classes (i.e., types of objects) that are associated with an input transition to a task are called the task's *input classes*, and those appearing on an output transition are called *output classes* of that task. A task's output class, which is not its input class, is *created* by the task. Specifically, an object instance of the specified class is created by the workflow runtime. A task's input class, which is not its output class, is *dropped (consumed)*. When input classes are unused by the task, they are transferred to the task's successor(s).

A group of input transitions is called an *AND-join* if all of the participating transitions must be activated for the task to be *enabled* for execution. An AND-join is called *enabled* if all of its transitions have been activated. All the input transitions of a task may be partitioned into a number of AND-joins. A group of input transitions is called an *OR-join* if the activation of one of the participating transitions enables the task.

A group of transitions is said to have a *common source* if they have the same source task and all lead either from:

- Its success state or
- Its fail state.

A group of common source transitions may form either:

- *AND-split*: Each of the transitions in the group has the condition set to `true`. This means that all of the transitions in the group are activated once the task is completed.
- *OR-split* (selection): An ordered list of transitions where all but the last transition may have arbitrary conditions (i.e., the last transition on the list has the condition set to `true)`. The first transition whose condition is satisfied will be activated.
- *Loop*: A special case of an OR-split, where the list is composed of exactly two transitions: loopback and continue. Loopback implies branch taken and continue implies branch not taken (i.e., fall through).

All tasks that we define in this paper are single-level tasks. What we mean by single-level is that the task receives input from one classification domain and produces output at the same classification domain. There are four special tasks: *begin, success, failure,* and *synchronization*. The synchronization tasks represent external transitions to and from other workflows. In general, workflow designers do not manipulate synchronization nodes directly. They are automatically generated by the system based on the specification of foreign tasks and input and output transitions to and from the foreign tasks.

An MLS *workflow* is a network of interconnected single-level (foreign or native) tasks from more than one classification domain. Note that we call a task single-level from one particular level of abstraction (view). Since a single-level task may be realized by an MLS workflow at a lower level of abstraction, it may have side-effects on different classification domains at lower abstraction levels. Hence, our distinction between single-level and multilevel is purely from the perspective of a specific abstraction level.

Let CL(T_i) represent the classification domain of task T_i. An MLS workflow that is the realization of task T_i where CL(T_i) = S_a must obey the following constraints:

- The *begin*, *success*, and *fail* nodes of the MLS workflow must be CL(*begin*) = CL(*success*) = CL(*failure*) = S_a.
- It may have tasks in other classification domains; however, if the CL(T_j) = S_b where S_a does not dominate S_b, then T_j must be a foreign task. In other words, only tasks in S_c where $S_a \geq S_c$ may have realizations.

For example, the workflow in Figure 1 is designed at domain M; thus special nodes are located in domain M. If the workflow designer creates an MLS workflow from the highest classification domain with a complete view of the workflow being designed, then the complete MLS workflow with realization of all its tasks can be specified. However, if the workflow designer creates an MLS workflow that requires input from (output to) higher classification domains, then he may only know the interfaces to the tasks at the higher levels but not the detailed workflow process at those levels. In such cases, foreign tasks can be used to define communication and synchronization with a task at higher classification domains.

5. MLS Dependence and MLS Workflow Decomposition

As we mentioned briefly in section 2, an MLS workflow design tool allows MLS workflow designers to:

- Divide a design area into multiple domains,
- Drop tasks in different domains, and
- Specify data and control flow among them.

Once the design of an MLS workflow is completed, the MLS workflow has to be divided into multiple single-level workflows to be executed on the underlying MLS distributed architecture that was described in section 3.

MLS Dependence

An MLS workflow design tool should allow the same kind of intertask dependence as a single-level workflow design tool (e.g., guards, input and output classes). However, some dependence in an MLS workflow may be specified across classification boundaries (i.e., MLS dependence). In other words, workflow state information and some values may have to move across classification boundaries during workflow execution. Hence, it is important to understand what the vulnerability is, whether it is easily exploitable and how to reduce it.

In our MLS WFMS, all information that has to move across classification domains must go though information release and receive policy servers and a high-assurance flow controller (e.g., Pump or downgrader [9,16]). We divide a transition between two tasks in different classification domains into a series of transitions. For example, if there is a transition from a task in domain 1 (T_{D1}) to a task in domain 2 (T_{D2}) as in Figure 7, then

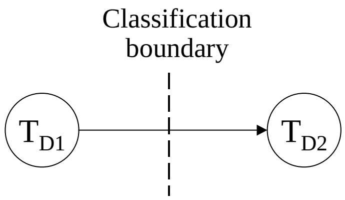

Figure 7: MLS Transition Across a Classification Boundary

this transition will be divided into transitions from T_{D1} to P_{D1}, P_{D1} to P_{D2}, and P_{D2} to T_{D2} as shown in Figure 8. Note that there is the flow controller between P_{D1} and P_{D2} where P_{D1} and P_{D2} are proxies that combine flow controller wrappers and policy servers (i.e., P_{D1} combines a flow controller wrapper and information release policy server, and P_{D2} combines a flow controller wrapper and information receive policy server).

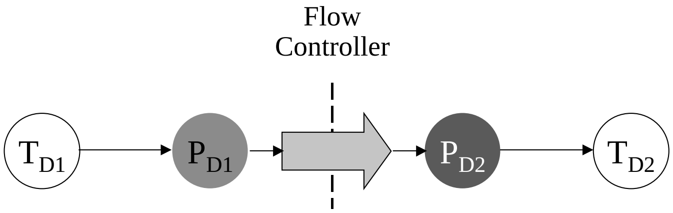

Figure 8: Indirect Transition Through a Flow Controller and Policy Servers

Note that domain policies may not allow combining flow controller wrappers and information policy servers. In that case, we have to break down transitions further. Also note that since T_{D1} and T_{D2} belongs to two different workflows, the P_{Di} serve as synchronization points that we discusses in section 4.

Decomposition of an MLS Workflow

Using the method that we described above, an MLS workflow will be separated into multiple single-level workflows. The single-level workflows neither communicate directly nor recognize single-level workflows from other classification domains. The number of single-level workflows that will be generated is equal to the number of classification domains in the MLS workflow (except the domains that contain only foreign tasks). When the breakdown is performed, direct transition (see Figure 7) becomes indirect transition as in Figure 8, and security depends on the underlying MLS architecture. Before we show an example of the division from an MLS workflow to multiple single-level workflows, we will formalize the MLS workflow model and the decomposition of an MLS workflow.

Formalism

An abstraction represents how we view a workflow. At the highest level of abstraction a workflow should be just one network task. However, as we traverse down the hierarchy of abstractions, workflows become more and more concrete with both network and simple tasks making up connected components via the flows. At the bottom of this hierarchy of abstractions, there should be only simple tasks.

There exists a partially ordered set of classification domains $\{D_i, \leq\}$ and a function $CL:\{\mathit{Tasks}\} \cup \{W\} \rightarrow \{D_i\}$.

There exists a set *Tasks.* (To distinguish the elements from the set itself, we capitalize the set name.) The set *Tasks* can be distinguished into the disjoint union of *Network Tasks* and *Simple Tasks*. The set of *Simple Tasks* can be further decomposed into $TT \amalg NT \amalg HT$. A simple task can be thought of as the smallest unit of work from the workflow's point of view because it is only associated with only one level of abstraction. TT represents transactional tasks, NT are non-transactional tasks, and HT are human tasks. One can think of a network task as a convenient way to group tasks that are associated with multiple levels of abstractions.

There also exists a set *Flows*. A flow can be thought of as the order of program execution (control and data flows). We are interested in tasks and flows that can be formed into directed graphs with tasks being the vertices and flows being the edges.

A **workflow** W is a directed graph made up of tasks and flows. {W} is the set of workflows. The goal of this section is to formalize the ideas about what workflows represent in the rest of this paper, thus ensuring rigor. In this section the concept of "MLS" is implicit in a workflow or tasks. We express this by the following definition.

There exists a set of *Abstractions* $\{z_1, z_2, \ldots, z_n\}$, the abstraction levels, with an ordering $\leq$, such that $z_1 < z_2 < \cdots < z_n$.

A workflow is always associated with an abstraction level z_j. The notation W^j signifies this association. The tasks of W^j are $\{N^j_1, N^j_2, \ldots, N^j_{\alpha 1}, S^j_1, S^j_2, \ldots, S^j_{\beta 1}\}$ where N^j_i is a network task of W^j and S^j_k is a simple task of W^j. For j = n, W^n consists of a single network task N^n_1 and For j = 1, W^1 consists only of simple tasks.

For each abstraction z_j, $j > 1$ there is a projection function π_{j-1} which takes an N^j_i of some W^j and gives a workflow W^{j-1} (associated with z_{j-1}). Furthermore we have the following:

Non-Increasing Security Projection Condition: $CL(\pi_{j-1}(N^j_i)) \leq CL(N^j_i)$, and this comparison is well-defined.

In other words the projection of a network task of a workflow will never give a workflow of a higher classification domain. Note that there is no concept of projection of a simple task because there is no lower level of abstraction for the simple task.

Starting with a workflow W^j (associated with z_j) we may form the *family of* W^j, $\mathcal{F}(W^j)$. The family of W^j gives a view of the workflow at its present, and all lower, levels of abstraction. Before making this definition precise we first discuss some other concepts.

Given W^j we define $P_{j-1}(W^j)$ as
$P_{j-1}(W^j) = \{\pi_{j-1}(N^j_i), \forall N^j_i \in W^j\}$, we may then form $P_{j-2}(W^j)$ as
$P_{j-2}(W^j) = \{\pi_{j-2}(N^{j-1}_i), \forall N^{j-1}_i \in P_{j-1}(W^j)\}$; thus we may recursively define
$P_{j-3}(W^j), \ldots, P_1(W^j)$. Now we are ready to define the family of W^j.

$\mathcal{F}(W^j) = \{W^j, P_{j-1}(W^j), \ldots, P_1(W^j)\}$

Given any classification domain D_k we may form the filter function F_{D_k} as:
$F_{D_k}(\mathcal{F}(W^j)) = \{\forall N^j_i \in W^j \ni: CL(N^j_i) = D_k\} \cup \{\forall N^{j-1}_i \in P_{j-1}(W^j) \ni: CL(N^{j-1}_i) = D_k\} \cup \cdots$
$\cup \{\forall N^1_i \in P_1(W^j) \ni: CL(N^1_i) = D_k\}$

We may also view the filter as a vector $\mathbf{F}_{D_k}(\mathcal{F}(W^j))$ where the (j-h)-component is
$\{\forall N^{j-h}_i \in P_{j-h}(W^j) \ni: CL(N^{j-h}_i) = D_k\}$.

We have the following obvious result: $\bigcup_k F_{D_k}(\mathcal{F}(W^j)) = \mathcal{F}(W^j)$.

Similarly, we may also form F_{bD_k} and $\mathbf{F}_{bD_k}$ by using network tasks whose classification is equal to or below D_k, instead of simply equal to D_k. Note that it is possible that some components are empty. The filter function provides a way to decompose multilevel workflow into single-level workflows, or a specified classification domain and below. Hence, the filter function provides a different view of multilevel workflows for users at different classification domains. For example, the view of a multilevel workflow at the unclassified level and the view of the same multilevel workflow at the secret level are different.

An Example Decomposition

Let us consider the simple MLS workflow shown in Figure 1 with classification domains L < M < H. Since this particular workflow is designed at domain M (special nodes, B, S, and F signify where the particular workflow was designed), all tasks in domain M and L, which is dominated by domain M, may be native tasks. Since the designer of the workflow in domain M may not know the detailed structure of the workflow in domain H, which dominates M, he can declare the TSA_H a foreign task. The specification of foreign task expresses interfaces (i.e., invocation methods and outputs) and where to send requests. The transitions, TSA to TSA_H, TSA_H to Ops1, Logistics to l-1, and l-2 to Ops1, and input and output classes that are associated with each transition define when and what kind of data will be passed to other workflows at different classification domains. After applying filter functions F_M and F_L, the runtime code generator generates the two workflows as shown in Figures 9-a and 9-b. The shaded proxies in Figure 9 represent the combination of policy server, flow controller wrapper and synchronization nodes that represent external transitions.

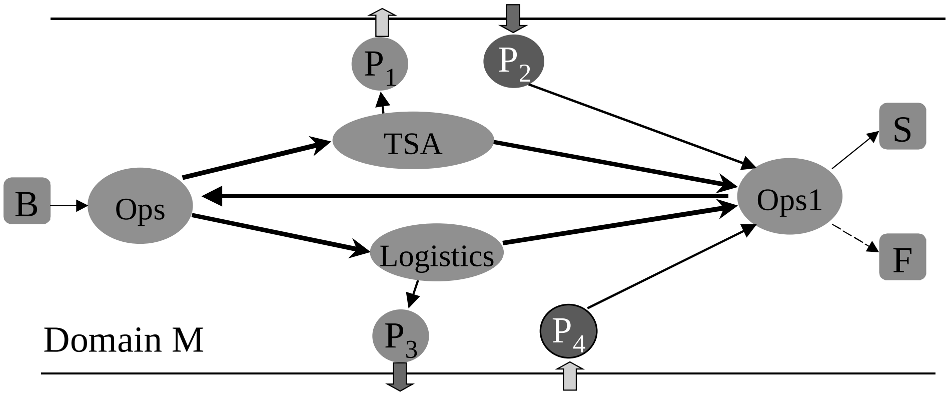

Figure 9-a: An Outcome of the M workflow after applying filter functions F_M

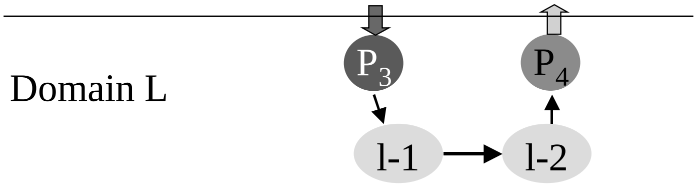

Figure 9-b: An Outcome of the L workflow after applying filter functions F_L

Note that workflows in domains M and L are two independent workflows after the filter function has been applied. The concept of cooperative processes and the synchronization node that we defined in sections 3 and 4 are useful for interoperability between these two independent workflows. Also note that applying the filter function to the original MLS workflow does not generate the workflow in domain H because there is only a foreign task in domain H.

6. Conclusion

Today's military is required to respond to constantly changing threats and cooperate with allies and different organizations. This dynamic environment and the military's dependence on IT systems require an MLS WFMS that allows

- Rapid specification and construction of mission specific IT systems,
- Maximum use of existing or commercial software/hardware, and
- Secure sharing and exchanging information among organizations in different classification domains.

In this paper, we presented requirements for an MLS workflow and tools needed to support an MLS WFMS. An MLS workflow designer should be able to specify different classification domains and tasks in those domains. He also should be able to specify control and data flows among tasks in different classification domains. To accommodate this need, we introduced an MLS workflow model and an MLS workflow design tool.

MLS workflow may be realized in many different ways. However, composing an MLS workflow from multiple single-level workflows is the only practical way to construct a high-assurance MLS WFMS today. Hence, we presented a strategy for implementing an MLS workflow by composing multiple single-level workflows on a multiple single-level architecture. When independent multiple single-level workflows work together to achieve a higher mission, workflow interoperability is a vital element. We introduced an extended workflow interoperability model for that purpose. We then presented a method for decomposing an MLS workflow design into multiple single-level workflows for runtime.

References

1. M. Anderson, C. North, J. Griffin, R. Milner, J. Yesberg, and K. Yiu, "Starlight: Interactive link," 12th Annual Computer Security Applications Conference, San Diego, CA, 1996.

2. N. Krishnakumar and A. Sheth, "Managing Heterogeneous Multi-System Tasks to Support Enterprise-Wide Operations," *Distributed and Parallel Database Journal*, 3 (2), April 1995.

3. METEOR project home page, http://lsdis.cs.uga.edu/proj/meteor/meteor.html.

4. OMG jFlow submission, ftp:///fpt.omg.org/pub/bom/98-06-07.pdf.

5. M. Kang , I. Moskowitz, and D. Lee, "A Network Pump," *IEEE Transactions on Software Engineering*, Vol. 22, No. 5, pp. 329 - 338, 1996.
6. M. H. Kang, J. N. Froscher, and I. S. Moskowitz, "An Architecture for Multilevel Secure Interoperability," 13th Annual Computer Security Applications Conference, IEEE Computer Society, San Diego, CA, 1997

7. M. H. Kang, J. Froscher, and B. Eppinger, "Toward an Infrastructure for MLS Distributed Computing," 14th Annual Computer Security Applications Conference, Scottsdale, AZ, 1998.

8. J. Miller, D. Palaniswani, A. Sheth, K. Kochut, H. Singh, "WebWork: METEOR's Web-based Workflow Management System," *Journal of Intelligent Information Systems*, Vol 10 (2), March/April 1998.

9. L.W. Chang and I.S. Moskowitz "Bayesian Methods Applied to the Database Inference Problem," 12th IFIP WG 11.3 Working Conference on Database Security, Chalkidiki, Greece, July 15-17, 1998.

10. K. Kochut, A. Sheth, and J. Miller "ORBWork: A CORBA-Based Fully Distributed, Scalable and Dynamic Workflow Enactment Service for METEOR," UGA-CS-TR-98-006, Technical Report, Department of Computer Science, University of Georgia, 1998.

11. M. Reichert M. and P. Dadam, "ADEPT$_{flex}$ - Supporting Dynamic Changes of Workflows Without Losing Control," *Journal of Intelligent Information Systems*, Vol 10 (2), March/April 1998.

12. Security Architecture for the AITS Reference Architecture, http://web-ext2.darpa.mil/iso/ia/Arch/SecArch1/SecArch1.htm

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

14. V. Atluri, W-K. Huang and E. Bertino, ``A Semantic Based Execution Model for Multilevel Secure Workflows," *Journal of Computer Security*, To appear.

15. R. Thomas & R. Sandhu "Task-based Authorization Controls (TBAC): A Family of Models for Active and Enterprise-oriented Authorization Management" 11th IFIP Working Conference on Database Security, August 1997.

16. I. S. Moskowitz and L. Chang, The Rational Downgrader, Proc. PADD'99, London, UK, April 1999.