

A Multilevel Secure Workflow Management System

Myong H. Kang,¹ Judith N. Froscher,¹ Amit P. Sheth,² Krys J. Kochut,²
and John A. Miller²

¹Information Technology Division

Naval Research Laboratory

²LSDIS Lab, Department of Computer Science

University of Georgia

<http://lsdis.cs.uga.edu>

Abstract. The Department of Defense (DoD) needs multilevel secure (MLS) workflow management systems to enable globally distributed users and applications to cooperate across classification levels to achieve mission critical goals. An MLS workflow management system that allows a user to program multilevel mission logic, to securely coordinate widely distributed tasks, and to monitor the progress of the workflow across classification levels is required. In this paper, we present a roadmap for implementing MLS workflows and focus on a workflow builder that is a graphical design tool for specifying such workflows.

1 Introduction

The constant aspect of today's military challenge is change due to the need for operational response 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 weapons of mass destruction and aggression against the United States. At no other time in the nation's history has the military relied so heavily on information technology (IT) for all of its operations, including command and control, logistics, surveillance and reconnaissance, personnel management, finances, etc. This dependence means that these systems must be easily configurable and secure.

The operational requirement for DoD is to pull together coalitions quickly and use US military systems as well as coalition partners' systems to achieve a common goal. Each mission has different mission logic and deals with different data sets. For example, the data for disaster relief are different from the data for biological weapons attack. To achieve the needed flexibility, the military should be able to react to different situations quickly and without procuring new IT resources for each crisis. Hence, there is a need to be able to specify the mission logic in terms of existing DoD

and coalition resources and applications, and enact that logic with the applications to achieve the mission. A workflow management system (WFMS) is a key enabler for such a capability.

A WFMS enables the automated coordination, control, and communication of tasks performed by people and/or computers. Although a majority of commercial off-the-shelf (COTS) WFMSs use a client server model, our requirements call for a WFMS that runs in a distributed, heterogeneous computing environment spanning one or more enterprises, as well as supporting integration with independent (legacy) software. We can view such a WFMS as a software layer above the user interface or application layer in the open systems model. For the commercial world, this is the business logic layer; for DoD, it is the mission logic layer.

However, current WFMSs lack capabilities that stem from the following unique operational requirements for DoD:

- The organizations that participate in dynamic coalition may be located in different security 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), in cooperation with the Large Scale Distributed Systems Lab at the University of Georgia, has embarked upon an R&D project to build a *multilevel secure (MLS) workflow* management system [1]. The goal of the project is to develop tools and security critical components that allow enterprises to harvest emerging COTS technology and still rely on legacy resources with reduced risk. Since our approach to solving the MLS workflow problem requires workflow interoperability, we have introduced extended workflow interoperability capabilities to our MLS WFMS. They are:

- A new workflow interoperability model (i.e., cooperative model),
- A mechanism to communicate to other independent workflows (i.e., synchronization nodes), and
- A new way to model workflow interoperability in a workflow design environment (i.e., foreign task).

This paper presents an approach for developing an MLS WFMS. In section 2, we briefly describe the overview of our 5-step strategy to implement an MLS WFMS. The detailed plan for the first two steps—choosing an MLS architecture and a strategy for dividing an MLS workflow into multiple single-level workflows are presented in section 3. Section 4 discusses the third step that involves executing single level workflows and reconstructing an MLS workflow from multiple single-level workflows. The tools that support building an MLS workflow using our approach are discussed in section 5. We conclude this paper by describing future work in section 6.

2 Technical Approach for an MLS WFMS

An MLS WFMS should support functionality equivalent to a single-level WFMS for users with different clearances but prevent unauthorized access to resources. Tasks that may be single-level individually but operate in different classification domains have to cooperate to achieve a higher-level mission.

Multilevel secure domains can be defined as follows. There is a lattice S of classification domains with ordering relation $<$. A classification domain S_i *dominates* a domain S_j if $S_i \geq S_j$. There is a labeling function L that maps each user, session, task, and data (object) to a classification domain. The classification levels a user can access are determined by his clearance and enforced by the underlying MLS architecture.

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

- High¹ 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.

The development of high-assurance software necessary to provide separation between the lowest security level (unclassified) and the highest security level (Top Secret) information has proven to be both technically challenging and very expensive through 20 years of computer security history. Today's fast paced advances in technology and the need to use COTS products make the traditional MLS approach untenable.

To implement an MLS workflow management system using the architectural approach, the following technical approach has been established:

1. Choose an MLS distributed architecture where multiple single-level workflows can be executed.
2. Choose a strategy for dividing an MLS workflow into multiple single-level workflows.
3. Select a single-level workflow management system to execute single-level workflow in each classification domain and devise a way to glue together single-level workflows to provide multilevel functionality.
4. Implement the necessary tools to support MLS workflow.
5. Extend the single-level workflow enactment service to accommodate communication among tasks in different classification domains.

¹ The term high[low] user/process/data stands for high[low]-level cleared/classified user/process/data.

3 MLS WFMS through Multiple Single-level WFMSs

In this section, we explain in detail the approach, outlined in section 2, for implementing the first two steps of our strategy for supporting MLS workflow.

3.1 MLS Distributed Architecture

Our approach depends on an MLS architecture to separate multiple single-level WFMS to achieve a multilevel secure WFMS. Therefore, multilevel security does not depend on individual WFMS but rather on the underlying MLS distributed architecture.

The MLS distributed architecture:

- Consists of physically or logically separated multiple single-level networks, each containing information at a given level and below.
- Hosts single-level applications and workflows that access information at that level and below, and
- Provides conduits to pass information among tasks in different classification domains.

The MLS distributed architecture is based on a security engineering philosophy: A few trusted devices in conjunction with information release and receive policy servers to enforce the information flow policy among the classification domains, and single-level systems and single-level engineering solutions to provide other functionality, including single-level security services.

In this architecture, switched workstations (e.g., Starlight [2]) 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 an NRL Pump [7]) together with information release and receive policy servers provide a secure way to pass information from one classification domain to another domain. A detailed description of the multilevel infrastructure and services can be found in *Towards an Infrastructure for MLS Distributed Computing* [8].

The workflow specification at each classification level is derived from the MLS workflow specification that a workflow designer provides through an MLS workflow builder. The next subsection describes how an MLS workflow specification can be decomposed into multiple single-level workflow specifications.

3.2 MLS Dependency and MLS Workflow Decomposition Strategy

An MLS WFMS should support the same kind of intertask dependencies as in a single-level WFMS.² However, some dependencies in an MLS workflow may be

² To enable the use of different WFMS products at different classification levels, restricted communication along the line of SWAP or OMG/jFlow standards may be adopted.

specified across classification boundaries; these are called MLS dependencies. In other words, 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 through information release and receive policy servers, and high-assurance flow controllers (e.g., Pump or downgrader). We can decompose a transition between two tasks in different classification domains into many transitions. Consider a transition from a task in domain 1 (T_{D1}) to a task in domain 2 (T_{D2}). This transition will be decomposed into transitions from T_{D1} to P_{D1} , P_{D1} to P_{D2} , and P_{D2} to T_{D2} as in Figure 1. Note that there is a flow controller between P_{D1} and P_{D2} where P_{D1} and P_{D2} are proxies that combine the function of flow controller wrappers and policy servers. Flow controller wrappers take care of any protocol translation between an application and a flow controller, and policy servers determine whether the information should be released to or received from another domain. For example, 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. Note that domain policies may not allow flow controller wrappers and information policy servers to be combined. In that case, we can decompose transitions further.

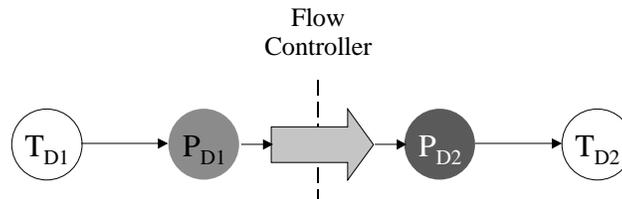


Fig. 1. Indirect transition through a flow controller and policy servers

In this way, an MLS workflow will be transformed into multiple single-level workflows. The single-level workflows neither communicate directly nor recognize single-level workflows from other classification domains. The minimum number of single-level workflows that will be generated is equal to the number of classification domains in the MLS workflow. An example of an MLS workflow design and its decomposition are illustrated in section 5.1.

4 A Single-level WFMS and an MLS Workflow Model

A centerpiece of our strategy for implementing an MLS workflow is to coordinate single-level workflows in the MLS distributed architecture that we described in section 3.1. Therefore, achieving MLS functionality largely depends on interoper-

ability among single-level workflows. In this section, we present an interoperability model that we plan to support as well as an MLS workflow model and primitives that we are implementing to support MLS workflow. In this section, we focus on the third item of our technical approach.

4.1 An MLS Workflow Model

We have chosen the METEOR system [4] as a starting point to build our MLS WFMS. The METEOR Enterprise Application Suite (EAppS) consists of four components: `EAppBuilder`, `EAppRepository`, `EAppEnactment`, and `EAppManager`. `EAppEnactment` includes two services: `ORBWork` and `WebWork`. Both `ORBWork` and `WebWork` use a fully distributed open architecture. *WebWork* [9] is a comparatively light-weight implementation that is well-suited for traditional workflow, help-desk, and data exchange applications. *ORBWork* [10] is better suited for more demanding, mission-critical enterprise applications requiring integration with legacy applications and data, high scalability, robustness and dynamic modifications. These features make the METEOR system with `ORBWork` enactment service a good starting point for extending capabilities to support MLS workflow.

To accommodate MLS workflow and other capabilities such as adaptive workflow, the earlier METEOR model [3] has been significantly modified. Some of the revised features have been influenced by our experience with building realistic workflows with our industry partners, while others have been influenced by other relevant research, including ADEPT [11]. We summarize only the small subset of the new METEOR model that is necessary for understanding the rest of the paper.

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 performing a computerized activity through executing a computer program, a database transaction, or possibly by 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 (see section 4.2). 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 and provide a hint to the METEOR runtime code generator. A foreign task should have a minimal information set that we will specify in section 4.2 (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 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 2 shows a simplified version of two levels of abstractions (views) where *Task2* is the parent task of the workflow W_i which contains tasks 4, 5, 6, and 7, and transition t_j represents a transition from *Task1* to *Task2*. In Figure 2, *Task1*, *Task2*, and *Task3* may belong to different classification domains. Hence, the MLS METEOR model can be thought of as follows: along the xy -surface, there are tasks in different domains and along the z -axis, there are different levels of abstraction.

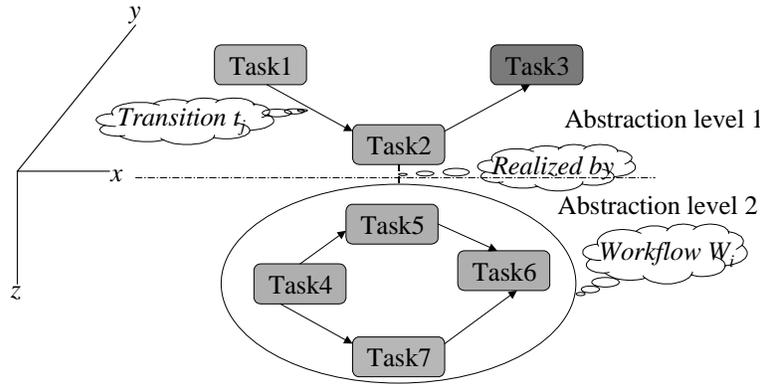


Fig. 2. MLS METEOR model

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 2) 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 of 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 decomposed 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 of the workflow). Similarly, an external transition leading out of a task in the workflow is considered to have an implied destination task (outside of 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*). Note that some input classes may be unused by the task. They are simply 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 from either its success state or its fail state. A group of common source transitions may form either an *AND-split*, *OR-split* (selection), or *Loop*.

All tasks that we define in this paper are single-level tasks. What we mean by single-level task is that it receives input from one classification level and produces output at the same classification level. 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.

An MLS workflow that is the realization of task T_i where $L(T_i) = S_a$ must obey the following constraints:

- The *begin*, *success*, and *fail* nodes of the MLS workflow must be $L(\textit{begin}) = L(\textit{success}) = L(\textit{failure}) = S_a$ and
- It may have tasks in other classification domains; however, if the $L(T_j) = S_b$ 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.

If the workflow designer creates an MLS workflow from the highest classification level with a complete view of the workflow being designed, then the complete MLS workflow with realizations of all its tasks can be specified. However, if the workflow designer creates an MLS workflow that requires input from (output to) higher classification levels, then he may only know the interfaces to the tasks at the higher levels but not the detailed workflow process at that level. Hence, in such cases, foreign tasks can be used to define communication and synchronization with a task at higher classification levels.

4.2 Workflow Interoperability

There are two aspects of workflow interoperability:

1. The interoperability protocol between independent WFMSs.
2. The ability to model the interoperability in a workflow process definition tool (i.e., workflow builder).

A standards body such as OMG (e.g., jFlow [5]) can handle the first aspect. However, the second aspect should be handled by each WFMS.

OMG's jFlow introduces two models of interoperability: nested sub-process and chained processes. 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 complete. 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 for interaction between the two workflows. 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.

These two models provide powerful mechanisms for interoperability. However, we would like to extend them to support a richer interoperability model: *cooperative processes*. Consider two independent autonomous workflows that need to cooperate. Let's assume that there are agreements among organizations that participate in the cooperation. 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 3. In this example, two workflows may have independent starting and ending points.

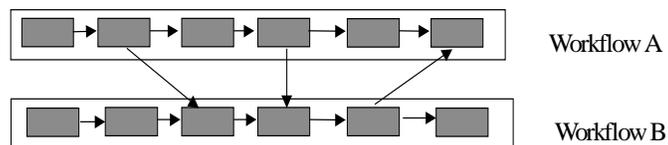


Fig. 3. Cooperative processes

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. This may be because organization A does not want organization B to know the structure of its workflow process 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, the designer of the workflow, whose classification level is M, may not be allowed to know the workflow structure that is in domain H where H dominates M.

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

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

Therefore, the above specification has to appear in the workflow design so that proper runtime code can be generated. Hence, we need a primitive that represents this situation in the design tool. The foreign task, introduced in section 4.1, can accommodate this need.

5 Workflow Tools for MLS Workflow

As we mentioned in section 4.1, a workflow management system consists of, in general, four components. The `EAppBuilder`, sometimes called a workflow process definition tool, is a distributed programming tool with a graphical user interface. Users should be able to express mission logic in terms of input, output, and external transitions, guards, input and output classes, and conditions for enabling each task, etc. Once a user specifies the mission logic, the runtime code for the `EAppEnactment` can be generated. The `EAppEnactment` is responsible for task scheduling, enforcing dependencies among tasks, passing data from one task to another, and error recovery based on the generated code. The workflow monitor that is a part of `EAppManager` is a convenient tool to track and monitor the progress of work.

An MLS workflow needs all the tools that a single-level WFMS provides. However, an MLS workflow requires extra capabilities in those tools. We will examine the extra requirements and the capability we plan to support for each tool.

5.1 A Builder for MLS Workflow

An MLS workflow designer should be able to specify MLS mission logic graphically using this tool. In other words, this tool should provide:

- A global picture of the MLS workflow process (mission logic);
- Appropriate views for different users at different levels of abstractions;
- A way to express input and output classes, a guard for each transition, the structure of input and output transitions (e.g., AND-join, Loop) among tasks in the same and different classification domains; and
- Capabilities to define domains and to specify dominance relationships among domains (e.g., Top Secret > Secret > Unclassified).

Each task that will be specified in the tool may be either a foreign or native task. If it is a native task, it has a realization as described in section 4.1. This tool provides the capability to expand a task whose realization is a workflow to see the detailed specification.

To support the design of information flow among classification domains, this tool allows a workflow designer to divide the design region into many classification domains. It allows users to add tasks to a domain. Once a task is added to a domain, it recognizes the classification of the domain and associates that classification level to the task. In our MLS workflow builder, all tasks are single-level tasks at this abstraction level. However, a task in a classification domain may be realized by an MLS workflow at a lower abstraction level. If a user wants to see other levels of abstraction, he can do so by expanding a specific task that was realized by a workflow. The reasons for allowing only single-level tasks in our MLS workflow process definition tool are as follows:

- An MLS task can be decomposed into single-level tasks,
- Each task that was not realized by a workflow must run at a single host and site that are single-level.

Hence, it is more natural to map real-world tasks into single-level tasks in a workflow than to map them into multilevel tasks.

The MLS workflow builder also allows users to add transition arcs between tasks where the tasks may be in the same domain or in two different domains. If there is a transition arc from a task in one classification domain to a task in another classification domain, then there is an information flow between two classification domains. During the runtime code generation stage, the code generator recognizes information flow across classification domains and generates the special code that was described in section 3.2.

A designer of an MLS workflow, working at level S_a , often has a need to specify an interaction with a task T_i at S_a to another task T_j at S_b where S_a does not dominate S_b . Since S_a does not dominate S_b , the designer is not allowed to know the detailed description of T_j . For example, when a secret level workflow designer designs a workflow, the secret workflow may need data from a top secret level task. The secret level designer may not be allowed to know how the top secret task generates the answer, but he knows how to send a request and how to receive an answer when the top secret task sends information. Hence, the top secret task is a foreign task to the secret level designer. Even if S_a dominates S_b in the above example, the MLS workflow designer at S_a may not wish to specify (or does not know the details about) the workflow at S_b , and therefore, may treat T_j at S_b as a foreign task.

Let us give a concrete example that involves cooperative processes and foreign tasks in an MLS workflow design as in Figure 4 where multi-lined arrows represent information flow across classification boundaries, ovals represent tasks, and B (begin), S (success), and F (fail) are special nodes.

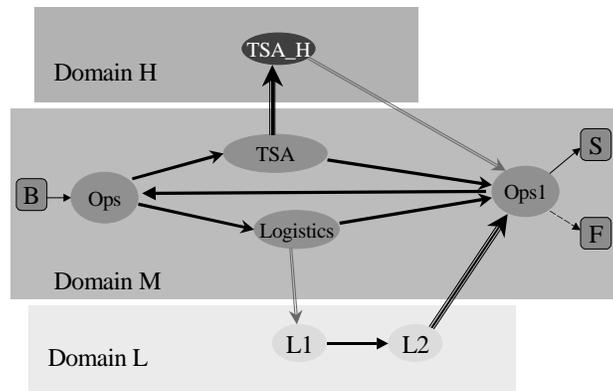


Fig. 4. A workflow that has a cooperative process and a foreign task

Logistics starts a workflow in domain L and *Ops1* receives information from the workflow in domain L. If the designer of the workflow in domain M does not want to specify the details of the workflow in domain L, he can declare *Logistics_L*, which is the combination of tasks *L1* and *L2*, as a foreign task.

Since this particular workflow is designed in domain M, all tasks in 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, he can declare the *TSA_H* as another foreign task. The transitions, *TSA* to *TSA_H*, *TSA_H* to *Ops1*, *Logistics* to *L1*, and *L2* to *Ops1*, and input and output classes that are associated with each transition define when and what kinds of data will be passed to other workflows at different classification domains. The specification of foreign task expresses interfaces (i.e., invocation methods and outputs) and where to send requests. The runtime code

generator uses that information (i.e., specification of foreign tasks and transition specification to and from other tasks) and generates two single-level workflows as in Figure 5 using the principles that were presented in section 3.2. Hence, shaded proxies in Figure 5 represent the combination of policy server, flow controller wrapper and synchronization nodes that represent external transitions. No code will be generated for the workflow in domain H because TSA_H is a foreign task.

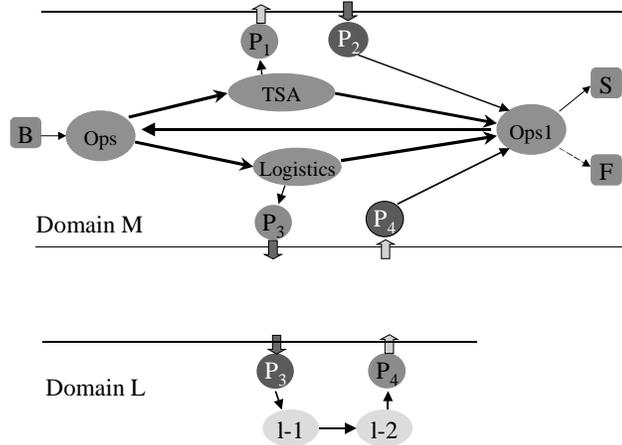


Fig. 5. An outcome of code generation from a workflow design specification

Note that 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., workflow builder neither accesses sensitive data in multiple domains nor passes it around). Hence, although this tool has to be trusted in the sense that the tool does what it is supposed to do, it can be run in a single-level platform.

5.2 Enactment Service for MLS Workflow

An MLS workflow enactment service is responsible for executing a workflow in a correct and secure way. As we presented earlier, our approach depends on:

- The services in the underlying MLS distributed architecture to coordinate information and control flow among tasks in different classification domains and
- Secure use of multiple single-level COTS workflow enactment services with or without modifications.

Since there will be no direct communication among workflow enactment services at different classification levels, there is no special MLS requirement for a workflow enactment service itself. On the other hand, the underlying MLS distributed architec-

ture and its security devices must provide the necessary assurance for multi-level security. However, our approach depends on workflow interoperability among multiple single-level workflow enactment services to achieve MLS workflow. This is why we extended the workflow interoperability model, introduced external transitions in the MLS METEOR model, and supported them in the METEOR enactment service.

One question that arises from our approach is “can we use other COTS WFMS enactment services to achieve MLS workflow function?” As long as a WFMS understands the concept of external transitions, we can in principle use any COTS WFMS at each classification level. In that case, the METEOR design tool can be used as an integration tool for designing a workflow comprising several independent workflows.

5.3 Workflow Monitor for MLS workflow

When an 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 (assuming a workflow manager per classification level) may have knowledge about tasks in their classification domain and other domains that they are authorized to access. In other words, users of an MLS workflow in different classification domains may have different views of the workflow that they are sharing. Hence, an MLS WFMS should provide the ability to monitor activities in all domains that the workflow manager is authorized to access.

Monitoring may include when, where, and who performs the tasks in the case of human tasks. Since the workflow designer specifies the expected behavior of a workflow, the workflow monitor can be designed to detect security critical events as well as unexpected behavior. For example, system failure or communication failure can be reported to a workflow manager through a workflow monitor. Also, if a task has not completed within a given time (i.e., deadline), those anomalies can be reported.

A WFMS that runs at each classification level is a single-level WFMS in our strategy for MLS WFMS. A single-level workflow monitor cannot provide all the capabilities that are desired for an MLS workflow monitor. The MLS workflow monitors at different classification levels should have different views of the workflow depending on the dominance relationship among classification domains. Our strategy for an MLS monitor is to send lower level status information (i.e., workflow control data) from the monitor at a lower classification level to a monitor at a higher classification level. The higher level monitor can present a unified execution status of the workflow in its classification domain and other domains that it is authorized to view.

Another natural question in the context of heterogeneous workflow is “what if the COTS WFMS monitor is not equipped to send status information to the outside?” In that case, we can create dummy tasks at higher classification levels to receive status information from lower-level tasks. Higher level workflow managers can monitor lower level activities through those dummy tasks.

6 Conclusion and Future Work

MLS workflow is a new research area that combines workflow and security technology. In this paper, we presented a technical approach to MLS workflow and the necessary techniques for our approach. We introduced a cooperative model and foreign task for workflow interoperability. We also described a new METEOR workflow model and the focus of our current development effort, a new MLS *EApp* \blacktriangleright *Builder*. The current builder saves design in the form of XML [6]. These XML files are used by *EApp* \blacktriangleright *Enactment* to generate runtime code and by *EApp* \blacktriangleright *Manager* to monitor and manage applications. Our immediate future work includes the modification of *EApp* \blacktriangleright *Enactment* to fully support a new MLS METEOR model and to develop graphical user interfaces for workflow monitors.

References

1. Atluri, V., Huang, W-K., and Bertino, E.: An Execution Model for Multilevel Secure Workflows. 11th IFIP Working Conference on Database Security (August 1997)
2. Anderson, M., North, C., Griffin, J., Milner, R., Yesberg, J., Yiu, K.: Starlight: Interactive Link. 12th Annual Computer Security Applications Conference, San Diego, CA (1996)
3. Krishnakumar, N., Sheth, A.: Managing Heterogeneous Multi-system Tasks to Support Enterprise-wide Operations. Distributed and Parallel Database Journal, 3 (2) (April 1995)
4. METEOR project home page. <http://lsdis.cs.uga.edu/proj/meteor/meteor.html>
5. OMG jFlow submission. <ftp://ftp.omg.org/pub/bom/98-06-07.pdf>
6. Extensible Markup Language (XML) 1.0. World-Wide-Web Consortium. <http://www.w3.org/TR/1998/REC-xml-19980210.html>
7. Kang, M., Moskowitz, I., Lee, D.: A Network Pump. IEEE Transactions on Software Engineering, Vol. 22, No. 5 (1996) 329 - 338
8. Kang, M., Froscher, J., Eppinger, B.: Towards an Infrastructure for MLS Distributed Computing. 14th Annual Computer Security Applications Conference, Scottsdale, AZ (1998)
9. Miller, J., Palaniswami, D., Sheth, A., Kochut, K., Singh, H.: WebWork: METEOR's Web-based Workflow Management System. Journal of Intelligent Information Systems, Vol 10 (2) (March/April 1998)
10. Kochut, K., Sheth, A., Miller, J.: 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. Reichert, M., Dadam, M. and P.: ADEPT_{flex} - Supporting Dynamic Changes of Workflows Without Losing Control. Journal of Intelligent Information Systems, Vol. 10 (2) (March/April 1998)