A Pattern for Managing Distributed Workflows

Seung Il Lee, Dongsoo Han, and Dongman Lee
{gaialee, dshan, dlee}@icu.ac.kr
School of Engineering
Information and Communications University
P.O. Box 77, Yusong, Taejon 305-600, Korea

1. Introduction

Patterns in the domain of workflow management systems(WFMS) have been written. Meszaros and Brown wrote a pattern language for workflow systems[11]. They considered the workflow facility a component of a system and described the process for creating that system. Thus their pattern language falls short of describing workflow facilities managed by an independent WFMS with additional features: transaction, adaptiveness, error handling, etc. Moreover, it does not address issues associated with managing *distributed workflows* in which more than one department or enterprise are involved. Such distributed workflows are very common these days.

In this paper, we propose a domain-specific pattern for managing distributed workflows named **ManageDistributedWorkflows**. This pattern allows a workflow to be executed and managed easily in several workflow servers supporting *autonomous* task execution and *hierarchical* workflow management.

2. Workflow Management Systems(WFMS)

Workflow can be described as flow of information and control in a business process. In this respect, workflow management means an efficient management of the flow and a WFMS is a software system that automates workflows. The core component of WFMS for the automation is called *engine*, which can control the execution of a set of workflow process, maintain workflow control data and workflow relevant data, and handle an interface to invoke external applications. A workflow engine is involved in the *workflow enactment service* which provides the run-time environment where process instantiation and activation occurs utilizing one or more workflow management engines[1]. Although the workflow enactment service is considered as a single logical entity, it may be functionally distributed. So we define a physical system as a *workflow server* (or just use a server, if no confusion in the following) which provides the workflow enactment service.

A workflow can be grouped into two types according to the degree of task distribution within a workflow. One is a workflow within which all the tasks are executed at one location and the other is a workflow within which tasks are executed at more than one location. In this paper, we define the former as *centralized workflow* and the latter as *distributed workflow*.

As the Internet becomes more popular, business automation expands into the form of collaboration between physically distributed departments and even further to the extent of collaboration among companies and customers. This results in increasing prevalence of distributed workflows in business processes such as electronic commerce. Thus considerable research has been done for workflow models which can manage distributed workflows[1][2][4][7][8][9][10].

We propose a pattern that encompasses common key features of distributed workflow models. In the following sections, we describe the **ManageDistributedWorkflows** pattern that addresses the problem of managing distributed workflows in detail.

3. ManageDistributedWorkflows Pattern

Intent

The **ManageDistributedWorkflows** pattern allows distributed workflows to be executed and managed easily in several workflow servers supporting autonomous task execution and hierarchical workflow management in the domain of workflow management systems(WFMS).

Context

Consider an Internet bookstore that sells books through the Internet. The Internet bookstore in general requires two departments: an order department for receiving order from the Internet and a delivery department for delivering ordered books to its customers. These two departments can be located geographically in different areas to serve its customers efficiently. It is reasonable to place the order department in the urban area where its business activities are usually done. On the other hand, the delivery department can be located in the suburbs of the urban area since it requires a warehouse to store books.

The process of the Internet bookstore, stated here as an example, may follow the general sequence below.

1. First, a customer orders a book through the Internet.
2. And then the order department checks the customer's credit and the stock for the order.
3. After completing the checking process, the order department returns the order confirmation and the delivery date to the customer.
4. Then it sends to the delivery department a message requesting that it starts delivering process and providing the information corresponding to the order: delivery address, name of recipient, what book to deliver, and the delivery date.
5. The delivery department, then, will make an arrangement for the delivery and then ship the ordered book according to the schedule to the customer generally via a shipping company.
6. Once the delivery is done, the delivery department sends to the order department a message of delivery completion.
7. Finally, this process will end after the order department sends the delivery confirmation message to the customer.

Figure 1 shows this process in the form of an activity diagram.

Internet business process like the example described above is one of typical examples of distributed workflows. Thus if one is to automate distributed workflows, a WFMS should have the ability to handle distributed workflows in order to increase the efficiency of work processing.

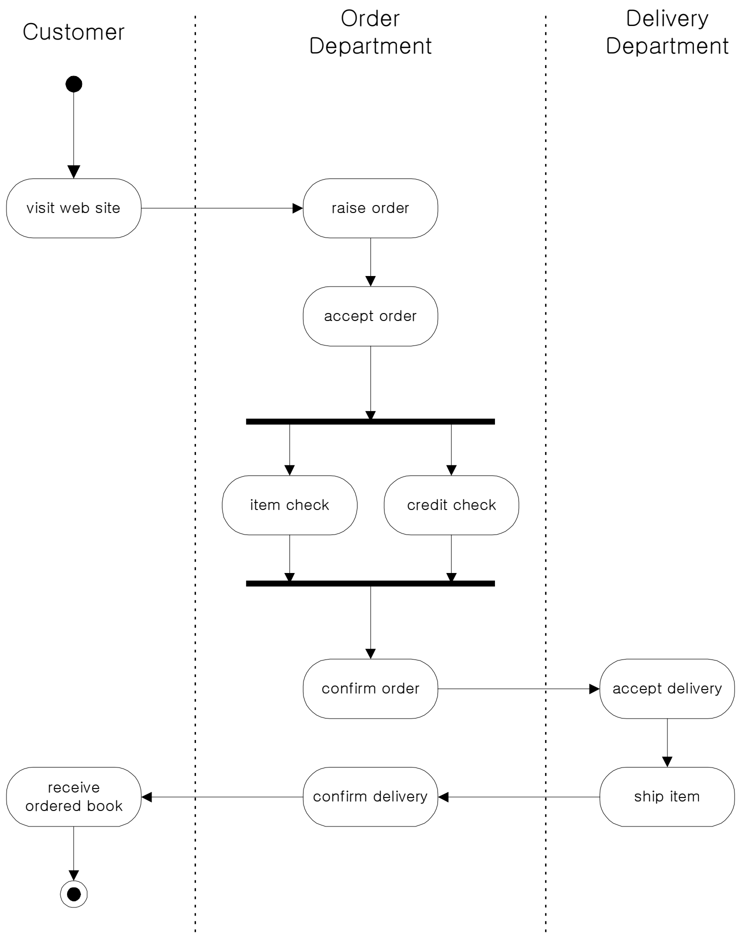

Figure 1. Activity diagram of an order process in an Internet bookstore

Problem

What should an architecture for workflow engines be like to manage distributed workflows effectively?

Forces

Suppose that a centralized workflow engine is used to handle distributed workflows. Information pertaining to enforcing inter-task dependencies, internal workflow management system state, and the organizational structure has to be stored in a centralized information repository. Thus the centralized architecture makes it easy to administrate and manage workflows. However, the architecture has two major problems in the automation of geographically distributed inter-department or inter-enterprise workflow processes. First one is frequent access to the workflow server that manages its repository. This is because workflow clients located not in the workflow server should send and receive messages to and from the repository to execute their assigned tasks. These frequent accesses might bring *heavy network traffic*. Second one is *processing load* in the workflow server. As the number of workflow processes grow, the processing burden in the workflow server increases because the overall workflow process is managed in the centralized repository. These two problems lead to severe performance degradation when the system is overloaded.

Instead, we choose a distributed architecture. Comparing to centralized workflow systems, distributed workflow systems involve data distribution, remote process creation, control or coordination of distributed processes running in different systems, etc. To fully get the benefits given by the distributed architecture, when designing a system handling distributed workflows, therefore, the following forces should be addressed:

- There should be no reliance on any centralized service that could limit the scalability of workflow processes.
- The network traffic and the processing load should be minimized when executing a workflow process.
- A workflow process should be created and managed flexibly at runtime.
- Engine components should run independently of the workflow clients. If not, the dependency between the engine components and the client makes the system more difficult to be maintained.

Solution

The essential components of a distributed workflow architecture are as follows.

A workflow process is composed of more than one task and executed in the unit of a task which is a logical execution step. So we introduce a *task manager* that is responsible for processing a task and manages only one task within a workflow process. It contains all the information for executing its task and thus need not to send and receive any data to and from any other components of the workflow engine. Due to this *task execution independency*, a task manager can decrease the reliance on a centralized service and can be created in any workflow server.

Though a task manager can execute a task autonomously, however, it has two dependencies with other task managers to enable a workflow to be controlled and scheduled in the process. One is *control dependency* with the task manager that has the control of the next task in the workflow process. The other is *data dependency* which results from resource sharing between tasks. The former is represented in a form of notification given by a task manager to the next task manager. The latter is represented in a form of a physical transfer of relevant data from a task manager to another task manager. Thus these two inter-task dependencies can be handled by a task manager itself, not by another special control component in the workflow engine.

Though a task manager can control the transition of tasks in a workflow process, the transition control is very localized since a task manager only concerns transitions to and from adjacent task managers. Thus if a transaction over several task managers are required, you need to manage the task-related information somewhere outside a task manager. So we need an object to manage a workflow process in the overall perspective. To decrease the reliance on any centralized service, we introduce a *workflow manager* that is responsible for managing task managers within a workflow process. A workflow manager handles the control of the overall workflow process and the information in the workflow process level, not the task level: name of the workflow process, number of task managers, data of transaction, etc. A workflow manager is also independent of any other workflow manager and can be created in any workflow server in any department.

A task manager and a workflow manager are introduced to address the first force. These two components represent a workflow process in a WFMS. The details of where and how these two are created should be considered at runtime. So, we need other objects for creating task managers and workflow managers in the appropriate workflow servers. This is composed of two processes: determining object location and object creation.

A *local factory* object is responsible for creating a task manager and a workflow manager at runtime. Each workflow server should have at least one local factory. We need another object, a *global manager*, for deciding the location of the task managers and workflow managers. A global manager selects an appropriate workflow server and requests local factories to create task managers or workflow managers. A global manager makes the decision to put a task manager in an appropriate server, since the task managers in those servers are responsible for the tasks assigned to a department that owns that workflow server. In the case of workflow managers, a global manager chooses one of several servers which contain the task managers involved in the same workflow process as the workflow manager. If there is more than one server in a department, a global manager will select a server which has the fewest tasks assigned to it.

To allow the engine components to run independently of workflow clients, we introduce a *workflow requester*, which intermediates between a workflow participant and a workflow engine. Each workflow requester is for a single enactment, that is, *instance*, of a workflow process, and starts an instantiated workflow process. Thus, a workflow participant creates a workflow process and runs it through a workflow requester in the WFMS.

The instantiation of the order workflow process in an Internet bookstore shown in Figure 1, using the components stated above, looks like the one as shown in Figure 2.

The structure shown in Figure 2 is composed of three workflow servers: one main server, and two workflow servers for each department. The main server manages the metadata of WFMS and other workflow services: repository for workflow templates, repository for log data, etc.

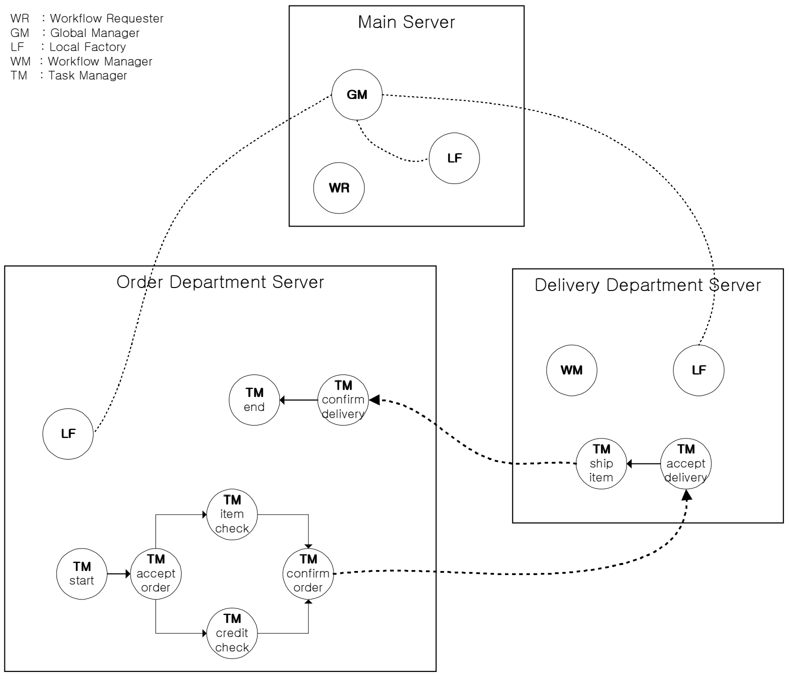

Figure 2. An instance of an order workflow process and other engine components

There are a global manager, a workflow requester, and a local factory in the main server. In the workflow server at the order department, there is a local factory for creating a workflow manager and task managers in response to the global manager’s requests. Note that for tasks assigned to the order department, task managers are instantiated only in the workflow server at the order department, which results in a decrease of network traffic. There is also a local factory in the workflow server at the delivery department. Like the order department, there exist task managers for tasks assigned only for the delivery department in the workflow server of the delivery department. As shown in Figure 2, there is also a workflow manager for this ordering book workflow process in the server at the delivery department. Actually this workflow manager could be instantiated in the main server or the order department server. But it is reasonable to create the workflow manager at the delivery department or the order department server due to the two reasons: reducing network overhead and resource balancing. The task managers concerning with the order process will be created in order department and delivery department servers. If the workflow manager is created in the main server, it has to communicate with two servers through the network. However, if the workflow manager is in one of the two workflow servers, it can reduce network overhead. Considering resource balancing, the workflow manager can be in any of the two workflow servers depending on the resource allocation status of each workflow server.

Structure

As shown in Figure 3, the **ManageDistributedWorkflows** pattern has five concrete classes. Among the five classes, WorkflowRequester, GlobalManager, and LocalFactory should be instantiated at system setup time. WorkflowManager and TaskManager are instantiated at runtime responding to the requests from the instance of WorkflowRequester.

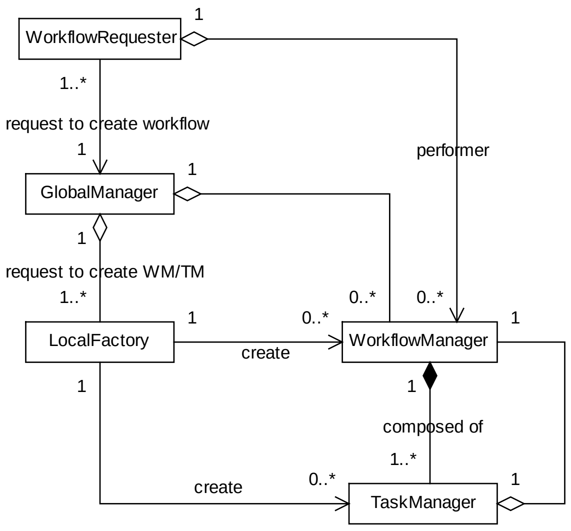

Figure 3. **ManageDistributedWorkflows** Pattern

Participants

✓ **WorkflowRequester(WR)**

- plays a role of an interface between a workflow participant and a workflow engine.
- is the requester in the workflow engine for creating a workflow instance and is the runner of the created workflow instance.

✓ **GlobalManager(GM)**

- manages the instances of LocalFactory and is an access point for instances of workflow processes.
- contains the references of instances of LocalFactory and delegates the creation requests of WorkflowRequester to an appropriate LocalFactory. Selection of a LocalFactory for specific instances of TaskManager or an instance of WorkflowManager is done with the information of the number of instances of TaskManager and WorkflowManager given by LocalFactorys.
- contains all the references of instances of WorkflowManager currently in the system. This makes GlobalManager the access point of instances of workflow processes.

- ✓ **LocalFactory(LF)**

  - creates an instance of TaskManager or WorkflowManger whenever there are creation requests from a GlobalManager.
  - keeps the number of instances of TaskManager and WorkflowManager that it has create. This is used for a GlobalManager to decide the location of TaskManager or WorkflowManager instances.
  - At least one instance of LocalFactory should be located in each workflow server.

- ✓ **WorkflowManager(WM)**

  - is responsible for managing TaskMangers that are for tasks within a workflow process.
  - is concerned mainly with keeping the data in the workflow process level: name of the workflow process, date of start, date of limit, number of tasks, references of task managers, etc. It does not contain any task-specific data: data for executing a task, transition data, etc.
  - can contain the control data of a workflow process when there are transaction or dynamic reconfiguration features in the workflow engine.

- ✓ **TaskManager(TM)**

  - is responsible only for managing a task within a workflow process.
  - The instance of TaskManager is an autonomous object. That is, it contains all the necessary data required in the process of executing its managed task that is executed independently of any other tasks.
  - As stated above, a TaskManager has two dependencies with other TaskManagers due to the flow of a workflow process, not to the execution of its task: control dependency, and dataflow dependency.

Collaborations

- **Workflow instance creation** – As Figure 4 shows, a business workflow process is instantiated through the interactions of the participants. A global manager tries to find an appropriate workflow server after workflow requester requests the instantiation of a workflow process. Once appropriate workflow servers are chosen, the global manager requests the creation of a workflow manager for the requested workflow process to a local factory in the server. Then the selected local factory instantiates a workflow manager. After this, the workflow manager requests the global manager to create task managers for managing the tasks within the workflow process. The workflow manager obtains the tasks within the workflow process from a repository containing workflow templates that was defined and saved using a workflow modeling tool. The global manager tries to find appropriate workflow servers again, but this time for task managers. After selecting workflow servers, it delegates the creation of the task managers to local factories in the selected workflow servers. An instance of the requested workflow process is created in the workflow engine after all these processes have been completed.

- **Starting a workflow instance** – To start a workflow instance, the workflow requester sends a start message to the workflow manager that is responsible for the workflow process. After receiving a start message, the workflow manager redirects this message to the task manager that is responsible for the first task within the workflow process. And then the task manager starts its task, which means that the workflow process starts.

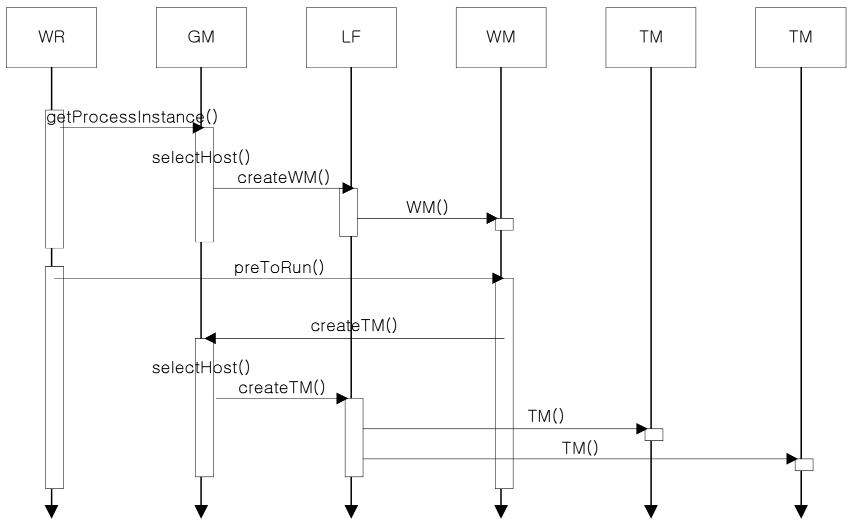

Figure 4. Participant interactions in the creation of a workflow instance

- **Removing a workflow instance** – Destruction of a workflow instance requires to log all the information occurred during the execution of a workflow instance and to release all the resources allocated for the workflow instance. Task managers and a workflow manager within a workflow process can save their log data independently into a repository. Each task manager or workflow manager returns its allocated resources independently as well. But during releasing resources, they also should send a delete message to the global manager and local factories that contain the references of them.

Consequences

The **ManageDistributedWorkflows** pattern allows a workflow manager to be independent of any other workflow manger. It also allows an autonomous task execution. These two functionalities can make workflow managers and task managers to be created and executed in any workflow servers. As a result, this pattern can give three benefits as follows, which are closely related to each other.

- **Decreasing communication load**
  There are two ways of decreasing network communication overload. One is to let a global manager decide the position of a workflow manager in one of the workflow servers at the departments which are involved in the workflow process assigned for the workflow manager.

The other is to make a global manager to decide the position of a task manager in a workflow server at the department responsible for the task. For example, in Figure 2, a workflow manager is created in the delivery department and each task manager is scattered across two department workflow servers according to its assigned department. Consequently, this decreases network overhead.

- **Resource balancing**
  Resource balancing can be achieved through the location selection of workflow managers or task managers. As described above, a workflow manager will be created in one of the workflow servers at the departments which are involved in the workflow process. Then for resource balancing, we can let a global manager set the position of the workflow manager not in an overloaded workflow server. For example, as shown in Figure 2, the number of task managers in the server of the order department is larger than that of the delivery department. If a workflow manager for the workflow process is to be created at the order department, the system performance could decrease because it might overload the server. In this example, it is better to create the workflow manager in the delivery part for the entire system performance. Resource balancing through the location selection of task mangers can be done alongside with processing load balancing, which is described in the following.

- **Processing Load balancing**
  The main processing load is the execution of tasks since executing a workflow process means executing each task within the workflow process. Thus we should consider how to distribute task managers across several workflow servers. There are two ways of balancing processing load. One is to let a global manager select the positions of task managers at the department level when the task is to be performed to decrease communication load. The other is to let a global manager select the positions of task managers evenly across several workflow servers in distributed manner. For example, suppose that there are several customers who intend to buy books through the Internet within very short time interval. Several workflow instances with the same type of a workflow process become present in the server. As the number of customers grows, so does processing load in the server. To lessen the processing load in the workflow server, we can simply install additional servers. If there are more than one server, we can let a global manager decide the positions of task managers evenly across several servers at the department, which can balance processing load between several workflow servers and balance system resources as well.

The **ManageDistributedWorkflows** pattern allows the information of a workflow instance to be managed hierarchically across a global manager, a workflow manager, and task managers, which increase the scalability of the system.

- **Hierarchical management**
  The information of a workflow process generated during its execution is managed hierarchically. A global manager has only references of workflow managers for each workflow process instantiated and running in the system. Thus the global manager plays a role of the access point to the management of workflow processes. The overall information of a workflow process is managed in a workflow manager and the information of each task is

managed in the corresponding task manager[see Figure 5]. For example, the state information of a task in a workflow instance is obtained as follows. First, a client can find out the reference of the workflow instance through the global manager. Now it can access to the workflow instance. Next, it finds out the reference of the task manager through the workflow manager. After obtaining the reference, it finally can query and get the state information of the task.

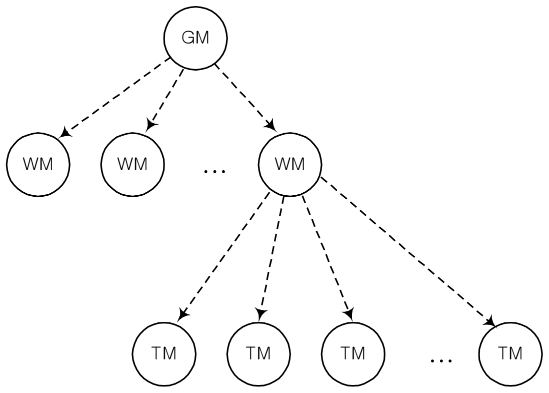

Figure 5. Managing workflow information in the hierarchical structure
A dashed line indicates the reference of the object in the head of an arrow

The pattern also has disadvantages as follows.

- **Implementation complexity**
  The pattern is based on a distributed system. Due to the separation property that a distributed system inherently has, additional considerations should be given such as consistency management for duplicated data and concurrency control for shared data between several engine components.

- **Message overhead**
  In the pattern, all the engine components are scattered across workflow servers and most of the system operation is based on the message transfer between the components. Most of these messages go through the network and this can be a system overhead compared to the centralized system.

Implementation

There are several issues that must be considered when implementing the **ManageDistributed Workflows** pattern.

- **Components as distributed objects**

  The **ManageDistributedWorkflows** pattern represents an architecture that inherently supports distributed environment through the global manager as a locator and the local factory as a factory. Thus using a distributed object model, a distributed workflow system might be rather easily constructed. To construct each component of the **ManageDistributedWorkflows** pattern as a distributed object, it is desirable to use standard middlewares such as CORBA and DCOM, that provide fundamental services for distributed objects such as naming service, transaction service, concurrency control, persistence service, etc.

- **Locator**

  To create a task manager or a workflow manager in an appropriate workflow server, a global manager should be able to obtain the information for selecting workflow servers.

  Two major things should be considered to create a workflow manager in the appropriate server: the location information of workflow servers in each department and the information of the number of workflow manager and task manager managed in each server. The former is required for reducing network overhead and the latter for resource balancing. If we use a middleware for distributed objects such as CORBA[12], the location information of workflow server in each department can be obtained transparently, which is possible to bind the department name to a local factory using a naming service. The information concerning the latter is obtained easily if a local factory provides interfaces by which a global manager can obtain the number of workflow managers and task managers in that server. Based on these two kinds of information, a global manager requests the creation of a workflow manager to a local factory being in the workflow server in the department responsible for the workflow process with plenty of resources

  Using the same information as those used for creating workflow manager, task managers can be created in the appropriate workflow servers. Using the former information, we can create task managers in the workflow servers that are in the department responsible for the tasks. This reduces the network traffic as in the case of creating a workflow manager. When there are more than one workflow servers at the department, we can balance processing load by creating task managers evenly across several servers using the latter information.

- **Autonomous Task execution**

  It is important to make a task manager autonomous in that execution since autonomous execution makes the distributed architecture different from the centralized one. A task manager must have all the metadata needed for executing its assigned task. The metadata contains information for executing the assigned task: general information of a task, information of participant for the task, reference of a repository that contains data for the task, information of applications used for the task, etc.

Known Uses

- **jFlow**[2]
  jFlow is the standard model about workflow management system, adopted in 1999 by OMG. To address the problem of managing distributed workflows, this model uses the **ManageDistributedWorkflows** pattern, though it does not explain explicitly and describes only interfaces. WfRequester, WfProcessMgr, WfProcess, and WfActivity interfaces in jFlow correspond functionally to WorkflowRequester, GlobalManager, WorkflowManager, and TaskManager respectively in the **ManageDistributedWorkflows** pattern.

- **Nortel**[7]
  This is a model about WFMS that was researched by The University of Newcastle upon Tyne, UK, partly sponsored by Nortel Technology. This model was also submitted to OMG's request for proposals for a workflow management facility and competed with jFlow. In the Nortel's model, a workflow is divided into more than one logical unit, that is a task, to manage the workflow efficiently. Nortel uses two engine components to manage and execute tasks within a workflow process. Task component is used for only executing its assigned task. TaskController component is used for scheduling and controlling a Task component. These two components are tightly coupled with each other and operate upon a task autonomously. As the functions of each components imply, these two components can be considered as one logical component. Comparing with the **ManageDistributedWorkflows** pattern, Nortel only divides TaskManager functionally into the above two components. And in Nortel' model, the functionalities of GlobalManager and WorkflowManager components are implicitly contained in Execution facility.

- **ORBWork**[8]
  ORBWork is a CORBA based workflow enactment system for the METEOR2 workflow model, which was developed in Georgia College. In the METEOR2 model, as in Nortel's model, the workflow system's runtime is divided into two types of components: task managers and tasks. The purpose of a task manager is to control or schedule the execution of a task within a workflow process. Thus the structure for managing distributed workflows in METEOR2 is much the same as that of Nortel's model.

Related Patterns

- **Factory Method**[3]
  The **ManageDistributedWorkflows** pattern uses the FactoryMethod to instantiate a task manager or a workflow manager. As shown in Figure 3, a global manager delegates the creation of a workflow instance to local factories and then a local factory instantiates task managers or global managers.

- **Composite Pattern**[3]
  There is a task that is a type of a workflow process and is called a *subprocess* since it is an element of a workflow process. In the example of this paper, the credit check task can be a subprocess since the process of checking credit of a person could be composed of more than one task. To represent subprocess, the **ManageDistributedWorkflows** pattern makes it possible for a task manager or a workflow manager to recursively compose each other in a treelike manner as shown in Figure 3. This means that the Composite pattern is used in the **ManageDistributedWorkflows** pattern.

4. Conclusion

Several researches on workflow model for managing distributed workflows have revealed a pattern that a workflow engine divides a workflow process into logical steps, tasks, and execute autonomously the workflow in the unit of task. The **ManageDistributedWorkflows** pattern described in this paper was obtained in the process of careful examining and developing of this pattern.

In this paper, we developed only a pattern concerning the architecture of a workflow engine for managing distributed workflows. However, there are several more patterns in the domain of WFMS. We plan to write more patterns and develop a pattern language in this field.

Acknowledgements

We want to thank Alan O'Callaghan for shepherding our paper. He gave us an insight into our pattern which shows that our pattern can be divided into a few small patterns and becomes a small pattern language. We also thank all the workshop members for their invaluable comments on this pattern.

References

[1] Workflow Management Coalition, *Workflow Management Coalition The Workflow Reference Model*, Document number TC00-1003, January 19, 1995.

[2] OMG BODTF RFP #2 Submission, *Workflow Management Facility*, Revised Submission, OMG Document Number: bom/98-06/07, July 4, 1998.

[3] E. Gamma, R. Helm, R. Johnson, J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, 1995.

[4] D. S. Han, J. Y. Shim, *Connector Oriented Workflow System for the Support of Structured Ad hoc workflow*, in Proceedings of The Thirty-third Annual Hawai'i International Conference on Systems Sciences(HICSS-33), January 4-7, 2000, Maui, Hawaii(CD-ROM).

[5] A. P. Sheth, W. van der Aalst, I. B. Arpinar, *Processes Driving the Networked Economy*, IEEE Concurrency, 1999, pp. 18 ~ 31.

[6] M. Anderson, R. Allen, *Workflow Interoperability – Enabling E-Commerce*, http://www.aiim.org/wfmc/mainframe.htm.

[7] Nortel, *Workflow Management Facility Specification, Submission*, supported by University of Newcastle upon Tyne, OMG Document Number bom/98-03-01, 1997.

[8] S. Das, K. Kochut, J. Miller, A. Seth, and H. Worah, *ORBWork: A reliable distributed CORBA-based workflow enactment system for METEOR2*, Tech. Report No. UGA-CS-TR 97-001, Dept. of Computer Science, University of Georgia, 1997.

[9] S. Paul, E. Park, and J. Chaar, *RainMan: a Workflow System for the Internet*, in Proceedings of USENIX Symposium on Internet Technologies and Systems, December 1997.

[10] D. S. Han, J. Y. Shim, and C. S. Yu, *ICU/COUWS: A Distributed Transactional Workflow System Supporting Multiple Workflow Types*, to appear July 2000 in IEICE Transaction on Information and Systems.

[11] G. Meszaros, K. Brown, *A Pattern Language for Workflow Systems*, in Proceedings of 4th Pattern Languages of Programs(PLoP) Conference, September 1997.

[12] *CORBAservices:Common Object Services Sprcification*, the complete formal/98-12-09, http://www.omg.org/corba/sectran1.html.