

國立交通大學

資訊工程學系
碩士論文

以代理人為基礎的工作流程管理系統中
定義的一致性

Definition Consistency on An agent-based
Workflow Management System

研究生：李吉正

指導教授：王豐堅 教授

中華民國九十三年六月

以代理人為基礎的工作流程管理系統中定義的一致性

Definition Consistency on An Agent-based Workflow

Management System

研究生：李吉正

Student：Chi-Cheng Li

指導教授：王豐堅 博士

Advisor：Dr. Feng-Jian Wang



A Thesis

Submitted to Department of Computer Science and Information Engineering

College of Electrical Engineering and Computer Science

National Chiao Tung University

In Partial Fulfillment of the Requirements

For the Degree of Master

In

Computer Science and Information Engineering

June 2004

HsinChu, Taiwan, Republic of China

中華民國九十三年六月

以代理人為基礎的工作流程管理系統中 定義的一致性

研究生：李吉正

指導教授：王豐堅 博士

國立交通大學

資訊工程研究所

新竹市大學路 1001 號



一般來說，工作流程管理系統是被設計成以資料庫中的工作流程定義及文件定義來執行。然而工作流程定義的靜態特性可能會降低系統的實用性。要開發一個工作流程管理系統的代價很高，所以把彈性和再利用性考慮進來是合理的。本篇論文中，我們要呈現一個利用代理人動態地修正工作流程定義的方法。在我們以代理人為基礎的工作流程管理系統中，一個執行中的工作流程可以被視為是一個（群）代理人，根據工作流程定義，和角色間的互動。代理人本身也能對一個工作流程作修改。因此，代理人可以修改它們對使用者的導引，而系統在面對規格異動時也能更有彈性。

Keywords: 工作流程、工作流程管理、代理人、網際網路

Definition Consistency on An Agent-based Workflow Management System

Student: Chi-Cheng Li


Advisor: Dr. Feng-Jian Wang

Institute of Computer Science and Information Engineering

National Chiao Tung University

1001 Ta Hsueh Road, Hsinchu, Taiwan, ROC

Abstract



Workflow management systems (WfMSs) are usually designed to run with the workflow (process) definitions and artifact definitions from repository. The static characteristics for process definitions might reduce the practicability of the systems. The cost to develop a WfMS is so high that it is reasonable to take the flexibility and reusability of system architecture into account. In this thesis, we present an approach to modify the definition of workflow processes with agents dynamically. In our agent-based workflow system, the running of a workflow can be viewed as an (or a set of) agent interacting with roles based on the definition of the workflow. The definition of a workflow can be modified by the agent itself. Therefore, agents can modify their guidance for workers, and the system will be more flexible to meet the specification modification.

Keywords: Workflow, Workflow Management, Agent, Internet

誌謝

本篇論文的完成，首先要感謝我的指導教授王豐堅博士兩年來不斷的指導與鼓勵，讓我在軟體工程及工作流程的技術上，得到很多豐富的知識與實務經驗。另外，也非常感謝我的畢業口試評審委員楊鎮華博士以及楊仁達博士，提供許多寶貴的意見，補足我論文裡不足的部分。

其次，我要感謝實驗室的伙伴們，有博士班建偉學長督導我們寫論文，對論文給予了相當多的寶貴意見，而其餘幾位學長姐熱心地參與幫忙和討論，讓我學得許多論文技巧，得以順利的撰寫論文。當然，值得一提的是我們這屆畢業生瓊文、大立及祖年，在各方面彼此不斷的砥礪與照顧下，使得大家在各個領域的技術及理論上能有所成長。

最後，我要感謝我的家人，由於有你們的支持，讓我能心無旁騖地讀書、作研究然後到畢業，此外，也要謝謝女友筱晴的細心陪伴，在我遇到挫折時能互相勉勵。由衷地感謝你們大家一路下來陪著我走過這段研究生歲月。

Table of Contents

摘要.....	I
Abstract	II
誌謝.....	III
Table of Contents	IV
List of Figures	VI
Chapter 1. Introduction.....	1
Chapter 2. Background.....	4
2.1 Related Workflow Research.....	4
2.2 Related Agent Technology.....	6
2.3 Our Background System	7
Chapter 3. Changes on Process Definitions.....	10
3.1 Characteristics of the Process Definitions.....	11
3.2 Possible Phenomena and Corresponding Decision Policies.....	13
3.2.1 Changes on Static Definitions.....	13
3.2.2 Changes on Dynamic Definitions.....	18
3.3 Problem Solution to Policies.....	20
3.4 A WF-Diff Algorithm.....	23
Chapter 4. Changes on Role and Artifact Definitions.....	30
4.1 Changes on Role Definitions.....	30
4.2 Changes on Artifact Definitions.....	34
4.3 Compound Changes.....	38
Chapter 5. Comparison and Conclusion.....	42
5.1 Comparison.....	42

5.2 Conclusion.....42

References.....44



List of Figures

Figure 2.1	System internal view.....	8
Figure 3.1	Three kinds of changes on flow graph.....	16
Figure 3.2	Examples of changes on flow graph.....	17
Figure 3.3	Several basic nodes.....	24
Figure 3.4	Common flow structures we used.....	25
Figure 4.1	The example of case 1.....	40



Chapter 1

Introduction

A workflow management system (WfMS) is an internet application system that can define, control and manage workflow processes[1][17]. A workflow process can contain business logics within an enterprise, or between enterprises and their customers[11][13]. By system's development tools and design rules, the workflow process designer can clearly describe all process definitions, which are stored in the repository of the system for execution[15]. The enactment service instantiates a workflow instance with the workflow definition[5]. Each workflow instance is instantiated when the precondition is true, interacts with role and terminates when the responsibility completes. On the other hand, the workflow engine transfers the output data to the preconditions which in turn are used for enactment of another process. So the system can direct all involved workers to cooperate and complete their working responsibility[11]. The management part of a WfMS is a sub-system monitoring the enactment services, and the system administrators can supervise, manage, and configure the whole system through this sub-system[17].

Modern internet software are getting powerful and complex[8][16]. The complexity of WfMSs increases significantly, so does the cost and time of system development. While the requirement changes fast, the life cycle of these systems cannot follow up due to the development or modification capability and cost. It might be helpful to consider the adaptability of the enactment service for a WfMS. In general, the enactment service can assist workers to do their job according to the process definition. A case not considered in the definition may cause the WfMS collapsing[3]. For example, that an enacted workflow process delivers the artifact to a dismissed

person would cause the process to be stopped. This case is solved by allowing workflow process to detect the expiration time and deliver the artifact to another person instead. However, this is a pre-defined or static case. The cases which are not considered might cause exceptions dynamically and the process designer has to modify the static process definitions. To take all possible changes into account is impractical and sophisticates the process definition[12][14].

The thesis presents an approach to allow a running workflow instance to continue its work, when the system administrator modifies the process, artifact, or role definitions. Our approach, based on an agent-based WfMS[22], lets an agent which is in charge of controlling process have self-adaptation capability. That is, the agent can reason and modify its process definitions for dynamic changes, which in general can be classified into three categories according to the process-, artifact-, and role- submodels respectively. The accurate adaptation actions usually rely on the concrete process modeling[9][19], and the essential principles to work around the demands can be addressed feasibly. To realize the dynamicity of workflow processes, the software agents should be enhanced with the situation-aware rules and adaptation functions[18][20]. With the principles discussed in the content, our agents can: (1) infer differences between original and modified definitions, (2) identify the change type and reason whether to adapt itself for the changes, and (3) choose a proper adaptation function for the running process. In order to cope with the changes, a WF-Diff algorithm is also presented to identify the modified range that affects the process running afterward. With the identified range, the agent can further decide the rules to modify its behaviors such as plans or even goals. Besides, we also give appropriate examples corresponding to each change type to demonstrate the feasibility of our adaptation approach.

The remains of this thesis are organized as follows. Chapter 2 describes related

discussion and an overview of our background system. Chapter 3 discusses the solutions to changes on the process sub-models. The changes on the role and artifact sub-models are described in chapter 4. In addition, the composite changes which consist of two or more kinds of dynamic changes are also described in chapter 4. Then we give a comparison and conclude briefly in Chapter 5.



Chapter 2

Background

2.1 Related workflow research

Generally speaking, every change which is unexpected or isn't concerned in the process design phase is classified as a request to modify the definition at run time, for example, an error on operating system or the hardware malfunction[10]. Several approaches have been developed to make a WfMS behave more flexibly when an unexpected case occurs[3][9][19][20].

Cardoso[3] defines an *survivability architecture* distinguishing the exceptions in four layers. The adaptability on this architecture is achieved by the exception handling mechanism with case-based reasoning. The goal of the case-based reasoning is to derive an exception handler, based on the knowledge about the past experience. To cover a case before the system collects the experience, however, is not considered.

Jie and Stanley[5] proposed a dynamic workflow model (DWM) for inter-organization WfMSs, which is extended from the underlying meta-model of the Workflow Management Coalition's (WfMC's) workflow process definition language (WPDL). In this model, a task is encapsulated as an e-service. Within an organization, an e-service *adapter* is implemented as a wrapper of its service operations. The flexibility of the system is achieved according to the dynamic binding of the e-services, and the dynamic properties of the business process model are specified in terms of events, triggers, and rules. The activities inside this dynamic workflow model can post *events* to *trigger* business *rules* during the enactment of workflow processes. The design of the e-service adapter, however, are not described clearly, and the constraint-based description of the e-service is too simple to specify most business

tasks.

For dynamic interoperation between WfMSs and dynamic integration of enterprise applications, Kwak and Han [8] proposed a framework including four main components: *Workflow engine*, *Adapter*, *Service Interface Repository*, and *XML message*. By integrating these four components, an external sub-process can be determined and bound to the main business process at run time. If needed, the local workflow system would ask the adapter to search the required service from other workflow systems and/or enterprise applications. The search and binding of services are transparent to the user, and they can improve the flexibility, scalability, and interoperability. The *multi-tiered state transition model* in this framework would cause unnecessary complexity when an exception occurs in a process.

Aalst distinguished the changes on process definition into two types, ad-hoc change and evolutionary change[19]. He presented a generic workflow model to tackle the problem of change and to get the management information. In the approach, the process instance whose definitions have been modified would be migrated between different members of the same process family, which is specified by a set of *inheritance diagrams*. Aalst's approach, however, is restricted to the changes on control flow.

There is an approach to make WfMSs more flexible by allowing the existence of inconsistent and incomplete models and involving users during the interpretation of the situated model [9]. In most workflow engines, the process models are activated by the system enforcing the "script." Unlike other workflow systems, this interaction framework allows users to resolve ambiguities in the situation that the computer can't interpret the process model definitely. Thus the process modeling language should be simple, user-oriented, extensible and adaptable. Briefly, the author proposed a new look at how models are interpreted, and this inspires us to have a new look at the

relations between the processes and artifacts.

Müller and Rahm[10] proposed a rule-based approach for dealing with *logical failures*. Their mechanism for adaptation decides which process has to be informed, and estimates the temporal and qualitative implications. By the *event-condition-action rules* and the negotiation among system components, the automatic handling of “legal failures” is possible.

Another rule-based, asynchronous approach is to enact a business process by the assessment of legal status and directives of the system, or the “Event-Condition-Obligation” style[20]. In contradiction with the classical approach, this approach facilitates the conflict detection and the corresponding resolution. The obligations, however, have to be designed with caution to avoid the divergence of the “triggering” mechanism.

2.2 Related agent technology

Several characteristics of agent technology are helpful for the workflow management application. With the pro-activeness of the agent, the workflow enactment mechanism could be sensitive to the changes on the static definition in repository. With the mobility, the agent could carry data to the destination to do the computation, and the network traffic and computation time could be reduced significantly[21].

Yan and Maamar[4] analyzed the way how the agent-enhanced and agent-based technologies to affect a WfMS. They listed the advantages and disadvantages brought by the agent technology. Several research issues, like system architecture, agent architecture, and negotiation, etc, are discussed in the paper.

In ADEPT environment[6], the agent offers the service, which corresponds to some units of the problem solving activity. Several modules are defined to provide the agent

with problem solving capability. All agents in ADEPT have the same architecture. The *agent head* of an ADEPT agent is composed of several modules to manage the agent's activities, and the *agency* presents distinct agents' capability. Although ADEPT model provides an exception handling mechanism in the *situation assessment module* for the service failure cases, it does not cover much on flexibility, and cannot react to the services changes dynamically.

2.3 Our Background System

In a mobile agent system, the agents can not only collaborate and communicate with each other, but also *migrate* to another place to accelerate the problem-solving process. The Object Management Group (OMG) had proposed a standard specification, *Mobile Agent System Interoperability Facility (MASIF)*[7]. This specification defines a set of interfaces and data types for the interoperability between those heterogeneous mobile agent platforms of different organizations. Organizations following these specifications can migrate their agents to another heterogeneous agent platform for work.

Our agents are based on an extended agent architecture from JAM[2]. We extended the JAM system by implementing several *primitive functions*. For an agent's mobility and negotiation, our system follows the MASIF specifications to implement the *agent-move* and *communicate* functions.

Figure 2.1 illustrates our system. All the workflow, artifact, and role definitions, named static definitions, are stored in the script repository and database. The Agent Manager is responsible to instantiate, manage and destroy all kinds of agents. The other components are based on the services supported by the Agent Manager. The Workflow Manager manages authority settings and monitors workflow instances. The

agents can request the Workflow Manager for various services, such as instantiating a agent, inquiry for system status, starting a workflow process, etc. The administrators can maintain the static process, role and artifact definitions by administration tools, and control the running workflow instances through the Workflow Manager. The client tools provide GUIs for users and can interact with his/her person agent.

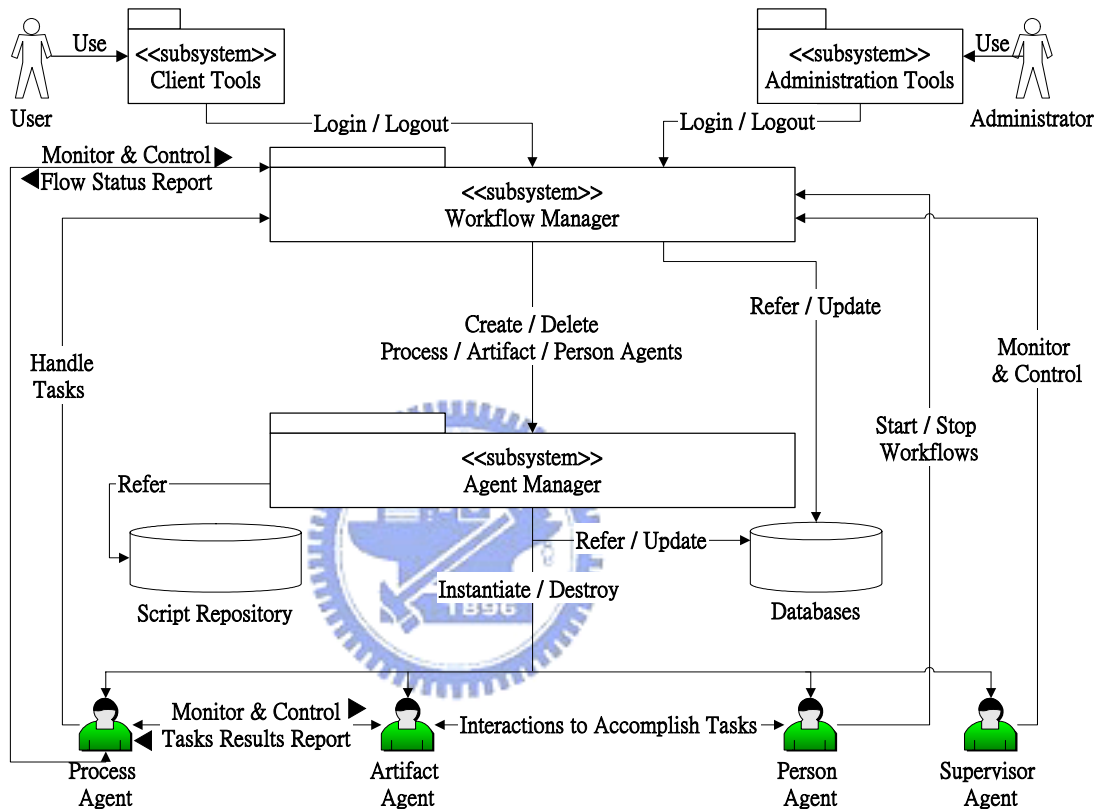


Figure 2.1: System internal view.

There are four kinds of agents which cooperate to accomplish a workflow (instance) in our system. When a workflow is initiated, a process agent will be instantiated with corresponding process definitions. The process agent then requests to create the artifact agents for required artifacts in the workflow. A process agent keeps a “flow graph” for each artifact agent, takes charge of task dispatching, and makes the routing decision for each artifact agent. An artifact agent carries the workflow relevant data (an e-form), keeps the state of these definitions, and communicates with the proper person agents assigned by the process agent. A person agent has the personal and role information of

the corresponding user, and it would present appropriate contents of the artifact to the user when he/she interacts with an artifact agent. The supervisor agent is a special person agent used by the administrator to interact with other agents.

Here is a possible scenario of the running system. Suppose that a user enters the workflow system with his/her client tool, and the client tool sends an instantiation request to the Workflow Manager. The Workflow Manager in turn requests the Agent Manager to instantiate a person agent, which carries the user's personal and role information. When user enacts a workflow with the client tool, his/her person agent is asked to send a request to the Workflow Manager. The Workflow Manager firstly checks if the user is authorized for the process. If the authentication is not passed, the request is rejected directly. Otherwise, the Workflow Manager asks the Agent Manager to instantiate a process agent which is in charge of the enactment of the workflow. This process agent then requests the Workflow Manager to instantiate one or more proper artifact agents, and makes the routing decisions for each artifact agent. Each artifact agent will communicate with the engaged person agents to perform (pre-defined) actions with the participants respectively. After completing one delegated work, the artifact agent reports the computation results to the process agent. Meanwhile, these reports and carried process definition are calculated by the process agent to make the next routing decision, and notifies the artifact agents of the decision. When the workflow process completes, the process agent will request to destroy all artifact agents and then itself be destroyed by the flow manager.

In the following sections we will distinguish the types of changes that may occur on the agents and the data carried by the agents, and discuss their solutions.

Chapter 3

Changes on Process Definitions

The processes, roles, and artifacts definitions, which are defined by the process designer in the design phase and stored in the repository, are the *static definitions*. When a workflow is enacted, a process agent is instantiated with the corresponding workflow definition. When an artifact is requested by a process agent, an artifact agent is instantiated with the artifact definition. When a user enters the system, a person agent is instantiated with his/her role definition. The definitions carried by these agents are called *dynamic definitions*.

The changes occurring at run time might work either on the static definitions or dynamic definitions. The changes on the static definitions might occur at the modification by the process designer, participants, or the other workflow instances. The modification on the static definitions can also change the dynamic definitions of the agents, if the administrator enforces to apply the modification immediately. The dynamic definitions can also be changed when some participants request to modify the workflow instances.

Basically, our system adapts to the changes in three steps:(1) Recognize: Firstly the process agent identifies whether there is a change and the type of the change. (2) Modify: After figuring out the change, the process agent will decide when to halt the workflow instance, how to modify the carried dynamic definition, and then apply the modification. (3) Resume: Finally the system records the system status and the process agent resumes the halted workflow instance. The third activity keeps the history of the changes and depends on the system implementation. So we will skip the discussion of the resuming action.

To identify whether there is a change, it is appropriate to periodically check the difference between the static definitions in the repository and the dynamic definitions carried by the process agent. If the process designer makes an urgent change on the static definitions, the workflow manager can actively notify the related agents of the changes. Besides, the process agent can be aware of the inherent exceptions, such as the inconsistent definitions caused by the process designer, during the enactment of the definitions. With the pre-defined rules, it can actively modify its carried definitions to continue the enactment, or even ask the workflow manager to update the static definitions and notify the other running instances. If the exceptions cannot be solved, the process agent might halt the workflow instance to wait for the administrator's manipulation.

When the process agent notices the differences between the static and dynamic definitions or the alert of changes from the workflow manager, it could adopt the proper actions according to the changes types and the status of the workflow instance. The rest of this chapter distinguishes the changes on the process sub-models.

3.1 Characteristics of the process definitions

The changes on the static process definitions could be resulted from the modification by the process designer, the system administrator, or the participants, as described above. To change the dynamic process definition, a user must have the authority to make the local change, like the insertion of a sub-flow. As the process agent detects an inherent exception, it can halt the workflow instance and notify the administrator to check out the static definition.

The following describes the typical definitions which can be carried by a process agent and the potential changes on these definitions. The further solutions are presented in the next section.

1. Workflow requester's information: If the system administrator modifies the authority setting of a workflow process, it may cause some contradiction. For example, a user may not be permitted to initiate the workflow instance that he/she has initiated.
2. Workflow identifier (unique name, or serial number, etc): The information is used to recognize the type of workflow instance. When the static definition of a workflow is changed, we can exactly recognize those workflow instances whose corresponding static definitions are just modified by compare their workflow identifiers.
3. Artifact identifier: The process agent can identify if a new artifact is inserted in the static process definition, or if an obsolete artifact is deleted from the static process definition, by the number of artifact identifiers. The change of an artifact's content cause no addition or deletion of artifact identifier in the process definition. This case will be discussed in next chapter.
4. Flow graph of an artifact: When the system administrator modifies the flow graph of an artifact, he/she changes the static artifact definitions. The process agent can distinguish the difference between the modified flow graph and the carried flow graph. Then, with the status of the workflow instance, the process agent can make a correct routing decision for the artifact agents.
5. Authority setting of local change: A local change on the dynamic process definition means that the change is only applied to the process definition carried by the process agent. A local change on a workflow instance can be the insertion of a new sub-flow or the deletion of an original sub-flow. If a person agent wants to make a local change to a workflow instance, it must be authorized to change the dynamic definition of the workflow instance.

3.2 Possible phenomena and corresponding decision policies

The way to adapt to the changes is case by case. Now we state the phenomena and the corresponding decision policies in our system, and illustrate the examples of the change types described above.

3.2.1 Changes on static definitions

- 1 **Changes on authority settings:** Only the process designer and the system administrator can modify one process's authority settings. If the process designer or the administrator takes off a user's authority to initiate a workflow process, the following requests of instantiation of this workflow process from the user will be rejected by the workflow manager. When a process agent notices the authority change of the user who is interacting with the artifact agent of the workflow instance, the process agent has to decide to continue or stop according to business policy. If the process agent decides to stop, it would cause the destruction of its artifact agents after their current tasks and notify the process requester of the situation. Then it would request to be deleted. If the process agent decides to continue, it would finish the rest of the process as the change on authority setting does not happen.

For example, every employee can initiate a workflow for applying money. Now, the system administrator modifies the authority setting to allow that in a department, the manager is the only person who can initiate the workflow for employee. In other words, the authority to initiate the workflow is changed. During the time of authority change, all the unfinished applying processes with the illegal process requesters will be stopped by their process agents. After notifying their requesters, these process agent and related artifact agents will be

destroyed by the workflow manager.

- 2 **Changes on artifact settings:** The process designer and the system administrator can add or delete an artifact to or from a workflow process. So do a participant with the authority of local change and we'll describe it in next section. After the modification on the static artifact settings, the process agent would request the workflow manager to create or destroy the corresponding artifact agents, and lead the rest artifact agents to complete the process. Although destroying an artifact agent will not affect the other artifact agents of the same workflow process, the process agent has to modify its process definition to make the correct routing decision for the rest artifact agents. For example, the process agent can directly lead the other artifact agents to the next site without waiting for the task report of the destroyed artifact agent. When the process designer or the administrator modify the artifact setting to add a new artifact into the workflow at run time, the process agent request the workflow manager to create an artifact agent with the corresponding artifact definition, and directs the artifact agent according to its flow graph. The process agent would make the routing decisions for the original artifact agents simultaneously. For consistency, an artifact agent should not be aware that itself is inserted for a dynamic change. For example, suppose that there is a new artifact agent added in a workflow instance, which had only one artifact agent before, and these two artifact agents shares the same one flow graph. In other words, the process agent has to route the two artifact agents at the same time. When one artifact agent is ready to enter a site, the process agent has to check if another artifact agent had arrived already. If not, the process agent further checks if another artifact agent had passed the site or not. Then the new artifact agent can finish the early task and catch up another artifact agent.

For example, the system administrator modifies the artifact settings of the

subvention recreation application process, and the workflow requesters are now asked to provide an additional manifest, like a voucher. All the unfinished workflow instances would be halted by their process agents. The process agent requests the workflow manager for a new artifact agent for the manifest. The new artifact agent will interact with the person agent of the process requester for the manifest's content, and follow the process agent's routing decision to the site the halted artifact agent stays. Eventually all artifact agents would get together and the process agent would make the routing decision for them.

- Changes on flow graphs:** The process designer and the system administrator can modify the flow graphs in the static definitions. A normal participant cannot change the flow graph of an artifact in the static definition, but a participant with authority of local change can modify the flow graph in the dynamic definition. The change on the dynamic definition will be described in the next section. After noticing the modification on the static flow graph, the process agent must modify the dynamic definition (the flow graph). Then according to the modified flow graph and the process status, the process agent will make the decision of how to adapt to the change. There are three cases. In the first case, the artifact agent has not moved in the modified region of the flow graph yet. The process agent just need to update the carried definition and will make the routing decision by the updated definition in the future. The "modified region" of the flow graph is a continuous segment within a flow graph, which includes all the nodes affected by a dynamic change. During execution time, it can also be deemed as an area of the flow graph starting from the first affected node to the last affected node. There will be a possible algorithm to find out the modified region in the later section. In the second case, the artifact agent has already gone through the modified region. There are two ways for the process agent to handle this kind of case. One is to

ignore the modification and continue to finish the remains of the workflow. Another is to route the artifact agent backward to the starting node of the modified region and start over again. In the third case, the artifact agent is inside the modified region. In this case, the process agent must make the artifact agent restart from the starting node of the modified region. Figure 3.1 illustrates these three kinds of cases.

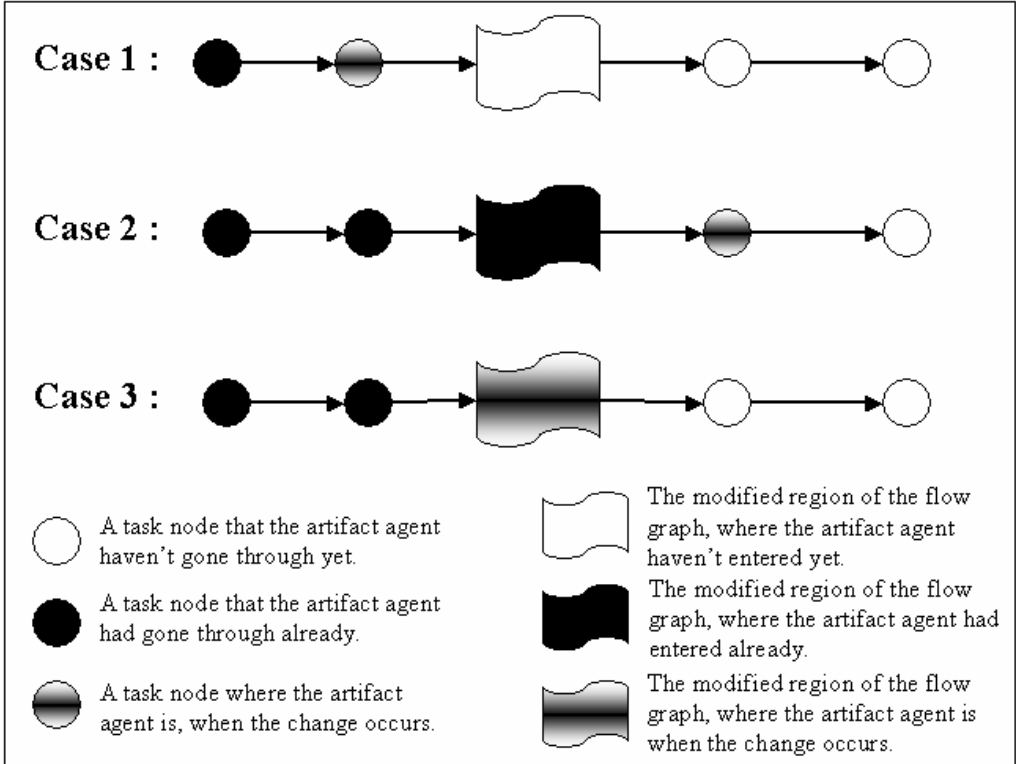
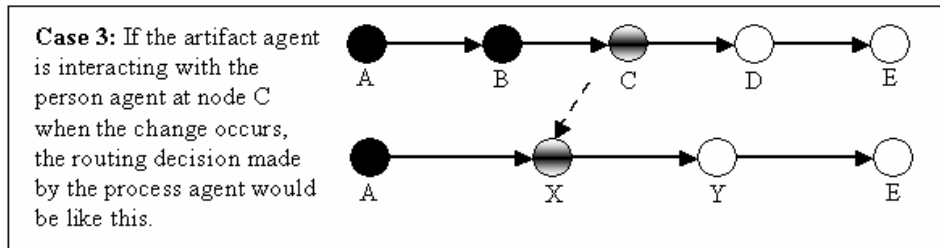
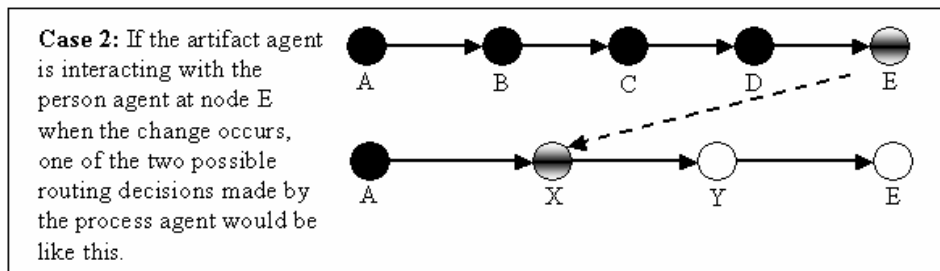
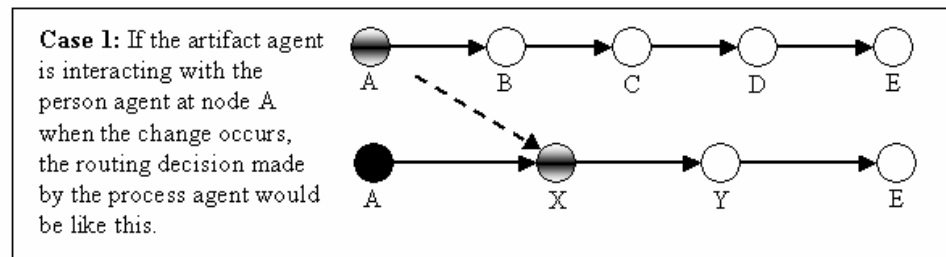
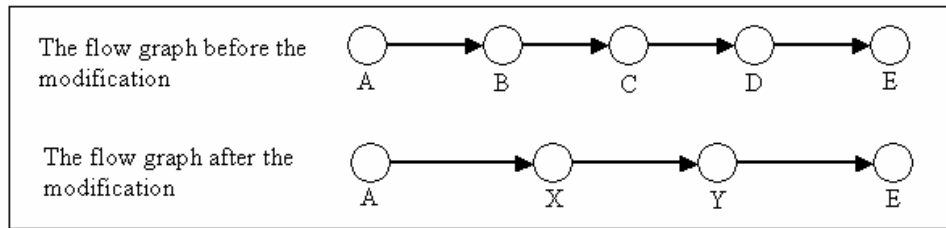


Figure 3.1: Three kinds of changes on flow graph

For example, in Figure 3.2, the original flow graph of an artifact was $A \rightarrow B \rightarrow C \rightarrow D \rightarrow E$. After the modification, the flow graph is changed to $A \rightarrow X \rightarrow Y \rightarrow E$. After the identifying and updating its definition, if the artifact agent is now at the node A, the process agent will make the routing decision by the updated flow graph from now on. If the artifact agent is at the node E, the process agent may continue making the routing decision by the original flow graph or route the artifact agent backward to the node X, depending on the

company's policy. If the artifact agent is now at the node C, B, or D, the process agent has to ask the artifact agent backward to the node X.



← - - - - The routing decided by the process agent

Figure 3.2: Examples of changes on flow graph.

For another example, originally, the flow graph of an artifact was like $A \rightarrow B \rightarrow C$, and now it is changed to $A \rightarrow X \rightarrow C$. The process agent has to ask the artifact agent backward to the node X. Although this case is a kind of the transference of the employee or the role, it also can be classified as a case of changing the flow direction. We will present an algorithm to infer whether the

artifact agent is in the modified region of the flow graph in the later section.

3.2.2 Changes on dynamic definitions

In previous section, we have described the phenomena about the changes on the static definition. Although the changes on the static definition can cause the corresponding change on the dynamic definition carried by the agents, the changes on the dynamic definition might not have to be applied to the static definition. This kind of change is called local change. The person agent with the authority of the local change can modify the dynamic definition carried by the process agent at run time. In most companies' business processes, only the managers have this authority. The process designer and the system administrator can also apply a local change to one of the running processes.

Although the process agent keeps up its workflow identifier and process requester's information, it is nonsense for a participant to modify the identifier or to disable the requester's authority. If the administrator want to modify the workflow process's identifier or take off one role's authority to initiate the process, it would be more convenient for him/her to directly modify the static definition as described in previous section.

When one of the participants requests a local change on the flow graph of an artifact agent, there are two kinds of the cases. In the first case, a new flow will be inserted after the node the artifact agent stays in the flow graph. The dynamic process definition would be modified by the person agent, so that afterward the process agent can correctly make the routing decision for the artifact agents to pass the inserted sub-flow. In the second case, a segment of the original flow graph after the node the artifact agent stays will be deleted. The process designer has to pay more attention on the authority settings and the structure of the workflow process when designing the

process and making the local change. Consider the authority settings, an employee can hardly make an artifact agent skip a sub-flow of the artifact's flow graph, if the sub-flow includes a node representing his/her manager. The system will be more powerful and reasonable with this consideration. Removing a sub-flow from the flow graph may also cause the synchronization problems. For example, there is an artifact delivered to two managers concurrently. After both managers finished their tasks, the president of the company signs it. Finally the artifact will be delivered to the accountancy division for a record. Because the two managers have to check the artifact at the same time, the process agent will duplicate the artifact agent and let both artifact agents interact with the two managers' person agents. If one manager requests for the local change and makes an artifact agent skip the next node. That is, the corresponding artifact agent would move directly to accountancy division. This is an inconsistency problem for the process agent.

After either the addition or the deletion of a sub-flow, the dynamic process definition carried by the process agent will differ from the static definition of the workflow in repository. Thereafter the process agent won't distinguish the difference between the dynamic and static definition. Similarly, the system administrator can make a local change on the flow graph of an artifact agent of a workflow instance. When the administrator makes a local change on a workflow instance, he/she only can change the region where the artifact agent of the workflow instance has not passed. When the local change is applied to one of the workflow instances, its process agent would modify its dynamic definitions and won't compare with the static definitions anymore.

A participant might request an additional artifact to follow with the original artifact. This is a local change on the artifact setting of the process definition. When the system administrator or a participant with authority of local change request for an additional

artifact, he/she has to define the artifact's flow graph and authority settings of fields definitely. When the local change is requested and the related artifact is defined well, the process agent would halt the original artifact agents and modify its dynamic definitions. After requesting the workflow manager for the new artifact agent, the process agent would make the routing decision for all the artifact agents. To delete an artifact from a running process is another local change on the artifact setting. The process agent would modify its artifact setting of the carried definition and make the correct routing decision for the rest artifact agents.

3.3 Problem solution to policies

In previous section, we analyzed the changes and phenomena that could happen on the process sub-model. In this section, we will present the design concept of each agent to show how our system adapts to the changes in every case.

1 Process agent:

i. In the case that the starting authority of process is changed.

- 1 Identify the change on authority either by receiving the alert from the system administrator or by actively noticing it.
- 2 Halt all the artifact agents after their current tasks.
- 3 Manipulate depending on the new authority setting and the company policy:
 1. If the process requester is still legal, nothing would happen.
 2. If the requester is no longer legal and the policy is stopping all these workflow instances, request the workflow manager for the destruction of the artifact agents.
 3. If the requester is no longer legal and the policy is continuing all

these workflow instances, ignore the authority setting afterward.

- 4 If the requester is legal, finish the process as normal. Otherwise, request the workflow manager for destruction and notify the requester of the situation.

ii ‧ In the case that the artifact setting of process is changed.

- 1 Identify the addition or deletion of the artifact identifiers either by receiving the alert or by actively noticing it.
- 2 Halt all the original artifact agents after their current tasks.
- 3 Request the workflow manager for the destruction of the obsolete artifact agents, if needed.
- 4 Update its definition of the artifact setting.
- 5 Request the workflow manager for the instantiation of the new artifact agents, if needed.
- 6 Follow the respective flow graphs to make routing decision for the rest and the new artifact agents.

iii ‧ In the case that the flow graph of an artifact agent is changed.

- 1 Identify the changes on the flow graph either by receiving the alert from the system administrator or by actively noticing it.
- 2 Decide the modified region with a WF-Diff algorithm.
- 3 Halt all artifact agents after their current tasks.
- 4 Manipulate in accordance with the process status:
 1. If the artifact agent has not entered or had passed the modified region, update its definition of the corresponding flow graph.
 2. If the artifact agent is within the modified region, update its definition of the corresponding flow graph, and set itself and the artifact agent to the state before the starting of the modified region.

5 Follow the new flow graph and make the routing decision as normal.

2 **Artifact agent:**

i 、 In the case that the starting authority of process is changed.

- 1 Receive the process agent's halting command after reporting the task's result.
- 2 Be destroyed by the workflow manager, if needed. Otherwise, receive the next routing decision and move to the destination site.

ii 、 In the case that the artifact setting of process is changed.

- 1 Receive the process agent's halting command after reporting the task's result.
- 2 Be destroyed by the workflow manager, if needed. Otherwise, receive the next routing decision and move to the destination site.
- 3 The new artifact agent would receive the routing decision and move to the first site after created.

iii 、 In the case that the flow graph of an artifact agent is changed.

- 1 Receive the process agent's halting command after reporting the task's result.
- 2 Receive the command of change state and change the artifact's state to a suitable state, if needed.
- 3 Receive the next routing decision and move to the destination site.

3 **Person agent:**

i 、 In the case that the starting authority of process is changed.

- 1 For the requester's person agent, it may receive a halting notification from the process agent since the requester becomes illegal to initiate the process. The user will know the authority change.
- 2 Afterward, the following request to initiate the corresponding process

will be rejected by the workflow manger.

3 The other person agents have nothing to do with this case.

ii 、 In the case that the artifact setting of process is changed.

1 For the person agents of the sites where the original artifact agents had passed, they may interact with the new artifact agents.

2 For the person agents of the other sites, they may interact with the new and original artifact agents.

iii 、 In the case that the flow graph of an artifact agent is changed.

1 For the person agents of the sites within the modified region, they might interact with the artifact agent again.

2 For the person agents of the other sites, nothing would happen.

3.4 A Supporting WF-Diff algorithm

To adapt to the dynamic changes as described in previous section, we need an algorithm to check whether the artifact agent is inside the modified region of the flow graph, and to verify if the changes affect the process. We will present a representation for the flow graph firstly. Then we will show the feasibility of this algorithm illustrated by the representation covering all kinds of flow graph styles.

With the derivation rules like that in a programming language, the whole flow graph of an artifact agent would be derived from a basic flow graph, or a starting symbol. By several common, basic blocks and the derivation rules, all kinds of flow graph structures can then be represented. In addition, by storing the progresses of derivation for each flow graph, we can figure out the differences between two flow graphs. We use this mechanism to infer whether an artifact agent is currently within the modified region of a flow graph.

We firstly define four simple symbols in figure 3.3 as the basic nodes in the later illustrations.

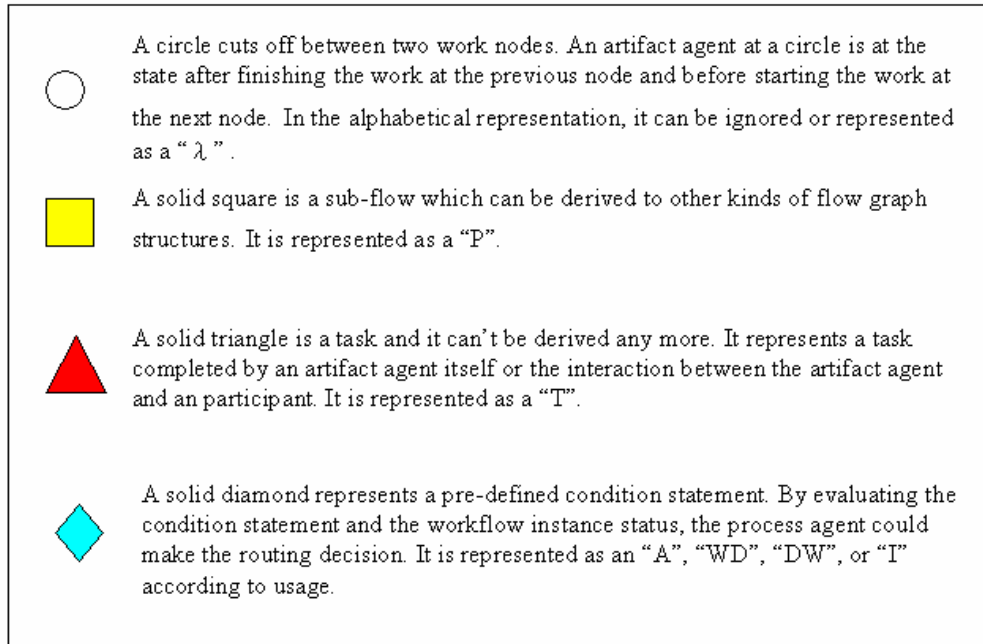


Figure 3.3: Several basic nodes.

Using these basic nodes, we list several typical flow structures, or blocks, which are common in most workflow processes. They are illustrated in figure 3.4.

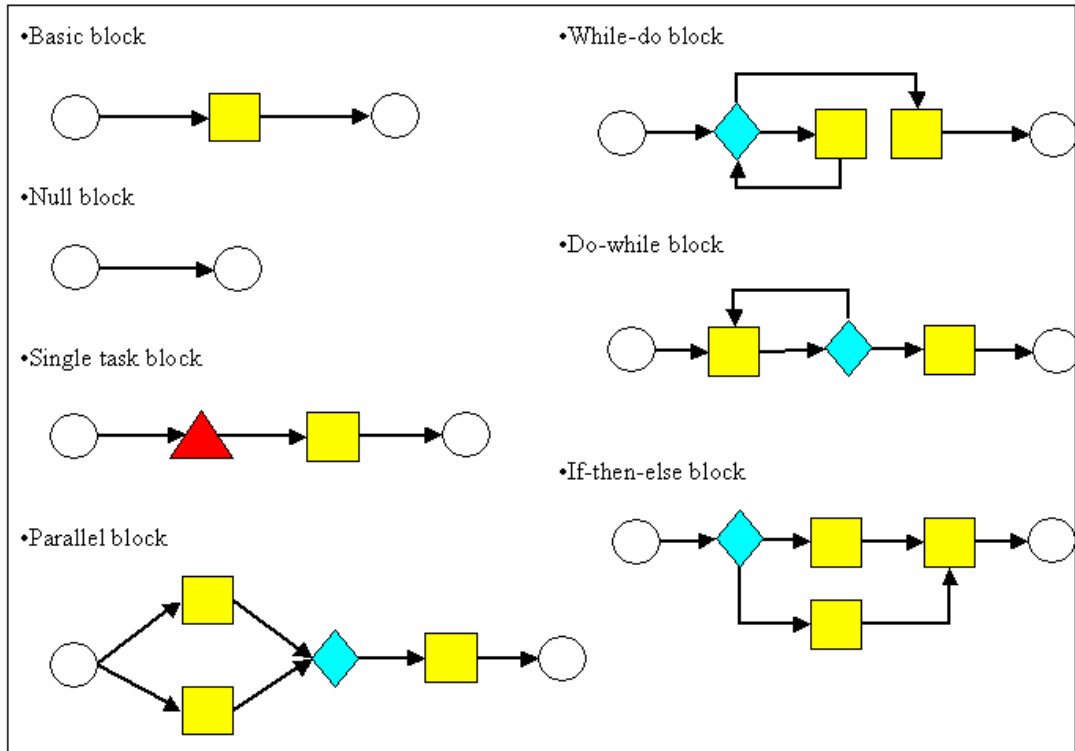
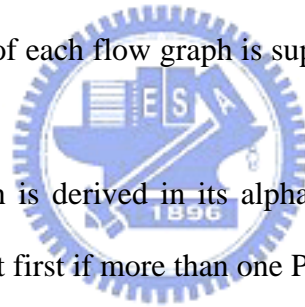


Figure 3.4: Common flow structures we used.

It is trivial for human to judge the difference between the flow graphs, but it's difficult for a program to check the diagram directly. So we replace the illustrations with the alphabetical representations and make the checking with the simple literal comparison rule in implementation. The *basic block*, which is deemed as the starting symbol in the derivation of a flow graph, can derive to other blocks, and is represented as "P" in the alphabetical representations. The *null block* can simplify the illustration of the flow graph in derivation, and it will be the last block to be derived when the derivation is completed. We represent the null block as " λ ". The *single task block* is used to derive a sub-flow with single task, represented as "TP". The *parallel block* shows the concurrent flow graphs, and it can be used to derive an OR-Join or AND-Join structure by a well-defined condition statement. The parallel block in the figure is a two branch case, represented as " $((P)(P))AP$ ", and the more "(P)" can be inserted as more branches are needed. These parentheses distinguish the different blocks

and are used as the borders in our alphabetical representation. The *while-do* and *do-while blocks* are another two optional flow structures used in business, and represented as “WD(P)P” and “(P)DWP”, respectively. The *if-then-else block* shows a simple conditional branch, represented as “I((P)(P))P”. Then the concepts of the derivation rules follow:

1. All the flow graphs can be derived from the basic block, P.
2. The λ can be omitted in the alphabetical representation. Similarly, in the illustrations of derivation, those adjacent circles can be combined.
3. The basic block, P, can be derived to other kinds of blocks, like $P \rightarrow TP \rightarrow T \lambda \rightarrow T$, representing the derivation of a flow graph that has only one task. The above list showing how P is derived to T is called a *derivation list* of this flow graph. The derivation list of each flow graph is supposed to be included in the workflow definition.
4. When a flow graph is derived in its alphabetical representation, the leftmost P should be derived at first if more than one P appears in the representation.



These concepts can be organized as the following grammar. The *PROCESS* and *task* used here refer to the part and subpart of the process definitions. We only consider the flow graph structures of derivations. More definitions and concerns are needed in implementation.

Grammar = {NONTERMINAL , TERMINAL , PROCESS ,
PRODUCTION-RULE }

NONTERMINAL = { *PROCESS* }

TERMINAL = { *task* , *parallel* , *while-do* , *do-while* , *if-then-else* , (,) , λ }

PRODUCTION-RULE = {

PROCESS $\rightarrow \lambda$;

$$\begin{aligned}
PROCESS &\rightarrow \mathbf{task} \text{ } PROCESS; \\
PROCESS &\rightarrow (\{ (PROCESS) \}^+) \mathbf{parallel} \text{ } PROCESS; \\
PROCESS &\rightarrow \mathbf{while-do} (PROCESS) \text{ } PROCESS; \\
PROCESS &\rightarrow (PROCESS) \mathbf{do-while} \text{ } PROCESS; \\
PROCESS &\rightarrow \mathbf{if-then-else} ((PROCESS)(PROCESS)) (PROCESS) \\
&\}
\end{aligned}$$

With the above grammar, we can describe abstractly most flow graph styles in our alphabetical representation. The terminal symbol **task** can be generalized to needed task's definition, and the **parallel**, **while-do**, **do-while**, and **if-then-else** symbols are conditional statement used in various situations. Note that the non-terminal symbol *PROCESS* is the starting symbol of this grammar, corresponding to the P of the alphabetical representation. So far, we have shown that our alphabetical representation of the flow graph covers the typical flow graph structures. Next, we'll present the feasibility of our WF-Diff algorithm with the alphabetical representation.

When the process agent notices the changes on the flow graph of an artifact agent, it should distinguish the difference between the flow graphs in the dynamic and static definitions. The following WF-Diff(A, B) algorithm can differentiate two flow graphs, A and B, by comparing their derivation lists. Flow graph B is supposed to be the obsolete flow graph and we want to find which segment in the new flow graph A differ from the flow graph B. In other words, we would like to find out the modified region in the flow graph B. After the algorithm finished, the process agent would have sufficient information to reason whether the artifact agent is before, after, or inside the modified region. Note that the inputs of the algorithm are the derivation lists, not themselves, of the flow graphs.

WF-Diff(A, B)

input: A and B are the modified and obsolete derivation lists of the flow graphs, respectively.

output: return the result of the Decide() function which use the difference between A and B.

```
1 pA ← Head of A ; pB ← Head of B
2 while both pA and pB are not NULL
3     do if pA.Content == pB.Content
4         then pA ← pA.Next ; pB ← pB.Next
5         else startA ← pA ; startB ← pB ; break
6 if pA and pB are not set yet
7     then return NO_CHANGES
8 pA ← Tail of A ; pB ← Tail of B
9 end ← ReverseCompare(pA.Content, pB.Content)
10 return Decide(startA, startB, end)
```

- On line 3, the comparison must be very complicated in implementation. Besides the alphabetical comparison, the semantic meaning of the symbols should be taken into account also. For example, two “T” can represent two different participants or two distinct tasks, two “C” represents two diverse condition statements, or two “A” represents two dissimilar parallel structure, such as one is an AND-Join action and another is an OR-Join action. All the above should be treated as the situations of comparison failure. So is the comparison in the function **ReverseCompare()** on line 9.
- The pA and pB on line 4 are the starting points of the difference of the two derivation lists, respectively. When the artifact agent needs to restart from the

node before the modified region, it must be set back to the state as if it just finishes the task at the node pA.Prev.

- On line 9, the **ReverseCompare()** function would make the comparison in the alphabetical representations in the reverse order to find the ending points of the difference of the two derivation lists, respectively. This ending point of the derivation list of A indicates the last node in the modified region.
- No matter if we find out the starting or the ending point, when the comparison fails on one character, the comparison fails on the smallest structures, which includes that character, in the alphabetical representation. For example, when we fail the comparison on the first T of $((T)(T))AT$, we also fail the comparison on this parallel block.
- Ultimately, the process agent could identify the difference between the two derivation lists. With the workflow instance status, it would reason whether the artifact agent is inside the modified region in the **Decide()** function.

Summarily, we show a representation for the flow graph and its derivation, and present an algorithm to identify the modified region within the flow graph with the representation in this section.

Chapter 4

Changes on Role and Artifact Definitions

In this chapter, we present the types of changes on the role and artifact definitions, respectively. Some noteworthy considerations when implementing the corresponding solution are provided. Since the changes on one sub-model usually cause the changes on another sub-model, we would show the way how our system adapts to these compound changes.

4.1 Changes on Role Definitions

A person agent in the WfMS corresponds to an employee in the company. After the workflow manager instantiates the employee's person agent with the static definition, the person agent would carry the employee's personal and role information. When the company makes the transference of roles, the static definition of some person agents would be modified by the system administrator. Then the WfMS shall actively replace the related person agents with modified definition. Although the employee cannot modify his/her role definition, he/she might want to change the personal data by requesting a workflow process. After the process finished, the person agent would be replaced with the employee's new personal information. In other words, the dynamic definition of the person agent should not be modified directly.

While involved in a workflow process, the participant's person agent would interact with the artifact agent. A new person agent might be improper to interact with those artifact agents of current definition. For example, different roles have different capability, i.e., different person agents. When a person agent is replaced with new

definition, the running interaction might be different. Even more, in a case that the new role has the same person agent or the person agent of the same authority as the old one, the execution results of the process agent will be as expected. In a case that the new role has higher or lower power of authority, this role owns various capabilities, e.g., it might own or lack of the authority to write a field of the artifact. For the process agent whose artifact agent had passed the site of the new person agent, there would be no change on flow graph of the artifact agent. For the process agent whose artifact agent has not entered the site yet, its artifact agent might interact with a different person agent instead. Therefore, when a role is changed, different process agents might have different modification on the flow graphs of their artifact agents.

Similarly, the change on the role definition usually causes the change on the access authority of the artifact content. Since the artifact agent may interact with a different person agent when a role is changed, the authority setting of the artifact content has to be modified. The change on the artifact definition will be described in next section.

Summarily, the change of an employee's role definition usually causes the changes on the other definitions, especially the flow graph and the authority settings of the artifacts. So the process designer or the administrator has to check and modify the affected definitions in the repository after modifying the role definition. Then the workflow manager would notify the process agent of the changes on the flow graph and the authority setting of the artifact. After the process agent updated its dynamic definition, the workflow manager would replace the person agents with the new definition. Then, the process agent would make the correct routing decision. Note that since the change on process definition might be the side effect of the changes on role definition, the process agent actually can adopt the solution described in 3.3 to adapt to the "ripple effect" of the changes.

For example, suppose that the manager B is replaced by the employee X and a

workflow graph is changed correspondingly from $A \rightarrow B \rightarrow C$ to $A \rightarrow X \rightarrow C$. That is, the person agent of the node B now has lower power of authority to the artifact agent of this workflow, and the person agent of the node X has higher one. When a process agent whose artifact agent had passed the node B notices the changes, it may continue as the change never happens. In the same case but the artifact agent has not passed the node B (or, node X now), its artifact agent shall move to the site of X after the site of A.

The following is the problem solution to the policies described above.

1 **Process agent:**

i · In the case that the change on role definition causes the change on flow graph.

1 Identify the changes on the flow graph either by receiving the alert from the system administrator or by actively noticing it.

2 Decide the modified region with a WF-Diff algorithm.

3 Halt all artifact agents after their current tasks.

4 Manipulate in accordance with the process status:

1. If the artifact agent has not entered or had passed the modified region, update its definition of the corresponding flow graph.

2. If the artifact agent is within the modified region, update its definition of the corresponding flow graph, and set itself and the artifact agent to the state before the starting of the modified region.

5 Follow the new flow graph and make the routing decision as normal.

ii · In the case that the change on role definition causes the change on authority setting of the artifact content.

1 Identify the changes on the authority setting by receiving the alert from the workflow manager.

- 2 Halt all artifact agents after their current tasks.
- 3 Notify the related artifact agents of the modification.

2 **Artifact agent:**

i 、 In the case that the change on role definition causes the change on flow graph.

- 1 Receive the process agent's halting command after reporting the task's result.
- 2 Receive the command of change state and change the artifact's state to a suitable state, if needed.
- 3 Receive the next routing decision and move to the destination site.

ii 、 In the case that the change on role definition causes the change on authority setting of the artifact content.

- 1 Receive the process agent's halting command after reporting the task's result.
- 2 Receive the new authority setting of the artifact content and apply the modification on the setting carried.
- 3 Receive the next routing decision and move to the destination site.

3 **Person agent:**

i 、 In the case that the change on role definition causes the change on flow graph.

- 1 Replace the obsolete definition with the new one received from the workflow manager.
- 2 For the person agents of the sites within the modified region, they might interact with the artifact agent again.

ii 、 In the case that the change on role definition causes the change on authority setting of the artifact content.

- 1 Replace the obsolete definition with the new one received from the workflow manager.

4.2 Changes on Artifact Definitions

The artifact agent is an artifact carrier in a workflow instance. All of the artifact definitions, including the artifact formats and the read/write authority settings, are stored in repository. When the process agent requests for an artifact with pre-specified definition, the artifact agent would be instantiated respectively. It then interacts with the process agent to get the routing decision and move to the next site. The artifact agent modifies the carried data after interacting with the proper person agent or the automatic manipulation. Then it reports the result to its process agent and waits for the next routing decision again.

Generally speaking, whenever the static definition in database is altered, the process agent would be notified of this change. Based on the change, the process agent is defined to ask the artifact agent to modify the carried data, or the dynamic definitions, after its current task. The information which an artifact agent carries for modification includes the followings.

- 1 Artifact identifier: Similar to the workflow identifier, when the static definition of an artifact agent is modified by the administrator, the process agent can find out the target agent right away, according to the artifact's identifier.
- 2 The names and the value types of the fields: When some fields of an artifact are renamed, the artifact agent can modify the content carried respectively. When the value types of fields are changed, the artifact agent modifies the data carried. Based on the status of the workflow instance, the process agent decides whether the artifact agent has to request new value from the person agent or not. For

example, let the social security ID field (and its value type) of the artifact definition be modified as the birthday field (and DATE). When the process agent is notified of this change (instruction), it would make the artifact agent to adapt to the change after finishing the current task. After applying the modification to the data carried, the artifact agent has to request the birth date from the proper person agent.

In the case of adding a field, the attributes of the new field should be well-defined in the static definition. After the artifact agent modified the data carried, the process agent might ask the artifact agent interact with the person agent to get the new field's value if the artifact agent had passed the site of the person agent. On the other hand, the deletion of a field would be trivial at omitting the deleted field thereafter, and the artifact agent would not reveal it to the later person agents. The careless deletion, however, might cause the contradiction of the process agent. For example, the artifact agent may interact with a person agent which cannot read any data, because all the fields which the person agent can access were deleted previously. To avoid this contradiction, the administrator has to check the whole process definition before deleting a field. If the process definition are modified corresponding to the deletion of the field, such as changing the flow graph of the artifact agent, the process agent might change its dynamic definition as described in 3.3, in addition to notify the artifact agent of the deletion.

- 3 Read-write authority for each role: The read/write access control for each field of the artifact should be set correctly. These authority settings decide what will be shown or can be done by a person agent. After the changes of the read-write authority settings in the static definitions, the process agent would be notified and thus the artifact agents for the authority. Similar to the change of a role definition,

if the authority setting for a person agent is changed and the process definitions are not modified correspondingly, the artifact agent may become illegal to interact with that person agent. Thus, the workflow process might never be completed. In fact, the case of deletion of a field in previous paragraph is a special case of the change of the read-write authority. If all the person agents involved in this workflow have no authority to access a field, the field seems to be deleted already.

The following is the problem solution to policies described above.

1 **Process agent:**

i 、 In the case that the names and value types are changed.

- 1 Receive the change instruction from the workflow manager.
- 2 Decide whether to ask the artifact agent to request for a new value with a WF-Diff algorithm.
- 3 Halt all artifact agents after their current tasks.
- 4 Ask the related agents to modify carried artifact contents, and to request for a new value, if needed.

ii 、 In the case that the read-write authority for one role is changed.

- 1 Receive the change instruction from the workflow manager.
- 2 Halt all artifact agents after their current tasks.
- 3 Ask the related agents to modify access authority setting.

iii 、 In the case that adding a field causes the change on the flow graph.

- 1 Receive the change instruction and identify the changes on the flow graph from the workflow manager.
- 2 Decide the modified region and whether to ask the artifact agent to request for a new value with a WF-Diff algorithm.

- 3 Halt all artifact agents after their current tasks.
- 4 Ask the related agents to add a field into the artifact content.
- 5 Manipulate in accordance with the process status:
 - 1 If the artifact agent has not entered the modified region, update its definition of the corresponding flow graph.
 - 2 If the artifact agent had passed the modified region, update its definition of the corresponding flow graph and ask the artifact agent to request for a new value.
 - 3 If the artifact agent is within the modified region, update its definition of the corresponding flow graph, and set itself and the artifact agent to the state before the starting of the modified region.
- 6 Follow the new flow graph and make the routing decision as normal.

2 **Artifact agent:**

- i 、 In the case that the names and value types are changed.
 - 1 Receive the process agent's halting command after reporting the task's result.
 - 2 Receive the new content of the artifact and modify the artifact content by the notification.
 - 3 Request the corresponding person agent for a new value by the process agent's instruction, if needed.
 - 4 Receive the next routing decision and move to the destination site.
- ii 、 In the case that the read-write authority of a field for one role is changed.
 - 1 Receive the process agent's halting command after reporting the task's result.
 - 2 Receive the new authority setting and modify the authority setting of field.

- iii ․ In the case that adding a field causes the change on the flow graph.
 - 1 Receive the process agent's halting command after reporting the task's result.
 - 2 Receive the new content of the artifact and modify the artifact content carried by adding a new field.
 - 3 Request the corresponding person agent for a new value by the process agent's instruction, if needed.
 - 4 Receive the command of change state and change the artifact's state to a suitable state, if needed.
 - 5 Receive the next routing decision and move to the destination site.

3 **Person agent:**

- i ․ In the case that the names and value types are changed.
 - 1 For those person agents of the sites where the artifact agent had passed, they might be asked for a new value by the artifact agent.
 - 2 For the person agents of other sites, nothing would happen.
- ii ․ In the case that the read-write authority for one role is changed.
 - 1 The person agent has nothing to do with this case.
- iii ․ In the case that adding a field causes the change on the flow graph.
 - 1 For the person agents of the sites within the modified region, they might interact with the artifact agent again.
 - 2 For those person agents of the sites where the artifact agent had passed, they might be asked for a value by the artifact agent.
 - 3 For the person agents of other sites, nothing would happen.

4.3 Compound Changes

In previous section, we have described the possible single changes on the

components in the system, and presented the corresponding solution discussions. But in reality these changes appear little alone. For example, the changes on some processes' definitions usually come along with the change on one role's definition. In this section, we would discuss how the system adapts to a compound change, by examples.

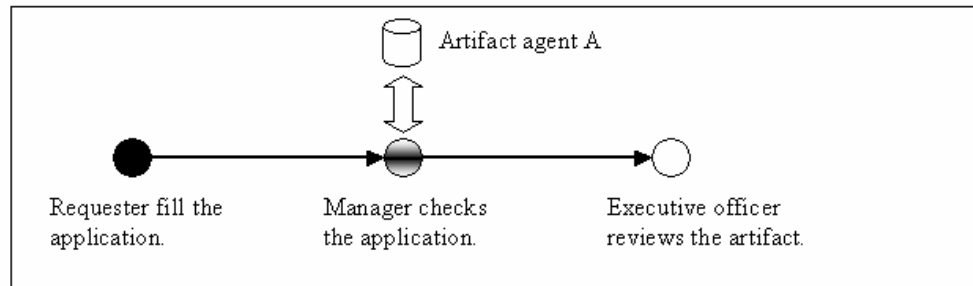
1. *Case 1: The flow graph of an artifact is changed, and the artifact contents are modified.*

- Case description: Suppose that a workflow process for a new project has a static definition change. With the workflow definition, the process requester fills in the application artifact before the manager checks it. Then, rather than being reviewed by executive officer as usual, the application artifact has to be audited by the accountancy assistant first.

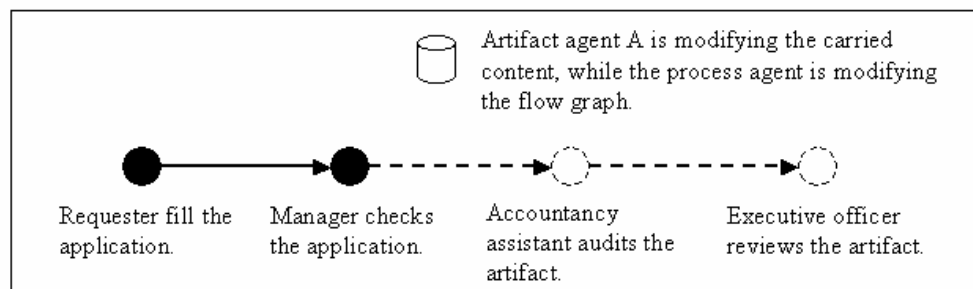
- Solution: According to the modification, a node representing the accountancy assistants is inserted between the nodes of the manager and executive officer in the flow graph of the application artifact agent. This is the change on the process definition. Besides, the fields of checking result and suggestion for the accountancy assistant are added to this artifact, and the authority settings of these fields are set for the assistant. These are changes on the artifact definition. After the changes on the static definition, if artifact agent A is interacting with the manager's person agent, the process agent would apply the modifications on the flow graph and ask the artifact agent A to modify the contents carried after it leaves the node of manager. Then the process agent would make the routing decision with the updated process definition. These are illustrated by figure 4.1. When artifact agent B is interacting with the executive officer's person agent, the process agent still has to notify artifact agent B to apply the modifications on the artifact contents, because change of the routing decision later. After the

modifications, artifact agent B will follow normal routing decision.

After the modification on the static definitions ...



After the task with the manager's person agent ...



After the modification on the dynamic definitions ...

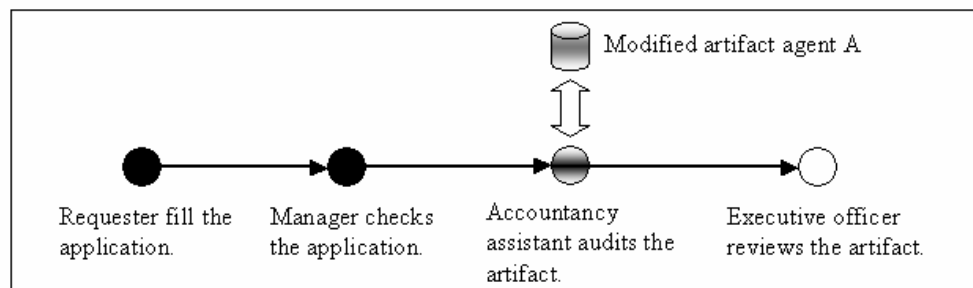


Figure 4.1: The example of case 1.

2. *Case 2: The authority setting of one role to request a workflow has been changed, and several additional artifacts have been added into that workflow.*
 - Case description: Suppose that the workflow process of requesting a new project can be started only by the department manager now. The report of the last project of the department should be provided also, as well as a budget table to estimate the cost of the new project.
 - Solution: Firstly, the authority setting of this workflow is modified so that only the department managers can request this workflow process with their client tool.

This is a change on the process definition. Secondly, two artifact identifiers are added into the process definition for the project report and the budget table. This is another change on the process definition. After the modification on the static definitions, the process agents would be notified of this change. If the administrator orders that all the related workflow processes which are requested by illegal person agents have to be stopped, those process agents will make their artifact agents stop after their current task. Then the starters of those workflow processes will receive the notification from the process agents. Other process agents would only be notified of the addition of artifacts for the project report and the budget table. The process agents will request for the instantiation of the two artifact agents, and make the routing decision for them. Suppose that the two new artifacts and the original artifact share the same flow graph. The original artifact agent would be stalled after the current task, while the other two artifact agents follow the flow graph to catch up the original artifact agent. Afterward, they would move together by the process agent's routing decision. On the other hand, if the administrator allows that all the running workflow processes can be completed, their process agents would only be notified of the addition of the artifacts, and behave as described as their starters are legal.

Chapter 5

Comparison and Conclusion

5.1 Comparison

To adapt to the changes occurring at run time, most current WfMSs adopt the version control mechanism. A tiny modification, however, might produce a new version of the workflow, and several workflow instances of various versions might run simultaneously. Stanley's work [5] provides the *adapter* mechanism to allow each organization maintaining its *services*, or, the process definitions it involved. They may replace the version control mechanism with the dynamic binding of the services. Compared with their models, the advantages of our system are the adaptation capability, where the process agent of each workflow instance can adapt to the changes at run time, and the practicability, where the most business logics can be enacted by our system. Rather than being restricted to the control flow as Aalst's model [19], we cover the changes both on data flow and control flow. In our system, the user will not be aware of the agent system, and the detection and resolution of the dynamic changes are transparent to the user, similar to Kwak's framework [8]. As described in chapter 2, the obligation rules of Abrahams' approach [20] have to be defined carefully to avoid the divergence of the number of the rules. It is not necessary in our system to set the rules to trigger each other rules for the flexibility. By the support of the underlying agent system, we can easily achieve the flexibility by the agent's reasoning mechanism.

5.2 Conclusion

The more quick Internet technologies evolve, the more complex the Internet

software grows. Because requirements are changed rapidly, most software costs arise from re-developing. To reduce costs and unnecessary reworking, software flexibility and adaptability are now significantly considered. In this paper, we discuss the modifications of a workflow and present an approach to increase adaptability on an agent-based WfMS. An intelligent agent can maintain its knowledge and react to variant environmental changes along the workflow. Adopting software agents helps the WfMS create, execute, and manage workflow processes more flexibly. We have classified all kinds of dynamic changes on process definitions, which are analyzed and categorized according to role, process, and artifact sub-models. Furthermore, solutions to adapting each change type are proposed respectively. An efficient algorithm to differentiate two process definitions is also well-defined. The designed agents therefore can react to these changes at runtime.

Environmental changes include not only those on user requirements, but several kinds of resource exhaustion or unexpected hardware/software failures. Some of them are not caused by humans and beyond our scope in this paper. However, more consideration and designs for handling unexpected events may bring a software system more solid and stable. We'll advance the adaptability of the WfMS and take more factors into account in the future.

References

- [1] Gregory Alan Bolcer and Richard N. Taylor. *Advanced Workflow Management Technologies*. Journal of Software Process Practice and Improvement, 1999.
- [2] Marcus J. Huber. *JAM : A BDI-theoretic Mobile Agent Architecture*. Proceedings of the third annual conference on Autonomous Agents. Seattle, United states, 1999.
- [3] Jorge Cardoso, Zongwei Luo, John Miller, Amit Sheth and Krys Kochut. *Survivability Architecture for Workflow Management Systems*. Proceedings of the 39th Annual ACM Southeast Conference, Athens, Georgia, 2001.
- [4] Yuhong Yan, Zakaria Maamar, and Weiming Shen. *Integration of Workflow and Agent Technology for Business Process Management*. Computer Supported Cooperative Work in Design, The Sixth International Conference on, 2001.
- [5] Jie Meng, Stanley Y.W. Su, Herman Len and Abdelsalam Helal. *Achieving Dynamic Inter-Organizational Workflow Management by integrating Business Processes, Events and Rules*. Proc. of the 35th Hawaii International Conference on System Sciences. 2002
- [6] N.R. Jennings, P. Faratin, T.J. Norman, etc. *ADEPT: Managing Business Processes Using Intelligent Agent*. Proc. of BC Expert Systems 96 Conference. 1996
- [7] GMD FOKUS and IBM Corporation. *Mobile Agent System Interoperability Facilities Specification*. Nov. 1997 Standard proposal.
- [8] Myungjae Kwak, Dongsoo Han and Jaeyong Shim. *A Framework Supporting Dynamic Workflow Interoperation and Enterprise Application*

- Integration*. Proceedings of the 35th Hawaii International Conference on System Sciences. 2002
- [9] Håvard D. Jørgensen. *Interaction as a Framework for Flexible Workflow Modeling*. Proceeding of the 2001 International ACM SIGGROUP Conference on Supporting Group Work.
- [10] Müller, Robert and Rahm, Erhard. *Dealing with Logical Failures for Collaborating Workflows*. Etzion, O.; Scheuermann, P. (Eds.): Proceedings CoopIS 2000, Eilat, Israel, September 6-8. LNCS 1901: 210-223.
- [11] Workflow Handbook 2003. Edited by Layna Fischer. Published by Future Strategies Inc. with WfMC. 2004. ISBN 0-9703509-4-5
- [12] Paul Dourish. *Developing a Reflective Model of Collaborative Systems*. ACM Transactions on Computer-Human Interaction. Mar 1995.
- [13] Sea Ling and Seng Wai Loke. *Advanced Petri Nets for Modeling Mobile Agent Enabled Interorganizational Workflows*. 9th Annual IEEE International Conference and Workshops on the Engineering of Computer Based Systems (ECBS'02).
- [14] Gary J. Nutt, Scott Brandt, Adam Griff, and Sam Siewert . *Dynamically Negotiated Resource Management for Virtual Environment Applications*. IEEE Transactions on Knowledge and Data Engineering. Jan/Feb 2000.
- [15] Ray J.R. Lin. *Enacting a Software Development Process*. Jun 1996.
- [16] Gary J. Nutt. *The Evolution toward Flexible Workflow Systems*. Distributed Systems Engineering, Dec 1996.
- [17] Workflow Management Coalition. *Workflow Management Coalition terminology and glossary* (WFMC-TC-1011). Technical report. Workflow management coalition. Feb 1999.
- [18] Anind K. Dey, Gregory D. Abowd, and Andrew Wood. *Cyberdesk: A*

Framework for Providing Self-Integrating Context-aware Services. ACM Symposium on User Interface Software and Technology. 1997.

[19] W.M.P. van der Aalst. *Generic Workflow Models: How to Handle Dynamic Change and Capture Management Information?* International Conference on Cooperative Information Systems. 1999.

[20] Alan Abrahams, David Eyers, and Jean Bacon. *An Asynchronous Rule-Based Approach for Business Process Automation Using Obligations.* Third ACM SIGPLAN Workshop on Rule-Based Programming. 2002.

[21] Weishuai Yang, Shanping Li and Ming Guo. *Mobile agent: Enhancing Workflow Interoperability.* International Conference on Info-tech and Info-net. 2001.

[22] Chiung Wen, Chang and Feng Jian, Wang. *Using Software Agents to Design a Modern Workflow Management System.* Master thesis. Dept. of CSIE, National Chiao Tung University. 2004.

