AN ONTOLOGICAL MODEL FOR WORKFLOW MANAGEMENT

by

JINGWEI XU

B.ECON., FUDAN UNIVERSITY, 2001

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE

in

THE FACULTY OF GRADUATE STUDIES
(Commerce and Business Administration)

We accept this thesis as conforming
to the required standard

THE UNIVERSITY OF BRITISH COLUMBIA

February 2004

Library Authorization

In presenting this thesis in partial fulfillment of the requirements for an advanced degree at the University of British Columbia, I agree that the Library shall make it freely available for reference and study. I further agree that permission for extensive copying of this thesis for scholarly purposes may be granted by the head of my department or by his or her representatives. It is understood that copying or publication of this thesis for financial gain shall not be allowed without my written permission.

Jingwei Xu
Name of Author *(please print)*

18/03/2004
Date (dd/mm/yyyy)

Title of Thesis: An Ontological Model For Workflow Management

Degree: Master of Science Year: 2004

Department of Commerce and Business Administration
The University of British Columbia
Vancouver, BC Canada

ABSTRACT

This thesis proposes a theory-based workflow model. It reviews workflow management basic concepts and discusses workflow generic constructs. It uses BWW Ontology as theoretical foundation and introduces an ontological process model. It derives a set of construct mapping rules that map ontological constructs to workflow management constructs, by extending PML-BWWP construct mapping rules which lead to the ontological workflow model by incorporating the concept of a controller and ontological analysis of workflow management using BWW process modeling. The model focuses on tasks, events and exceptions.

An ontological workflow model algorithm provides a procedure on how to transform a process model into a workflow model by investigating each task and each event following a recursive procedure. An example is used to illustrate the way in which the algorithm can be applied.

Finally, the thesis examines the contributions and limitations of the research work, and provides suggestions for future research.

TABLE OF CONTENTS

LIST OF FIGURES

Number Page

LIST OF TABLES

Chapter One

INTRODUCTION

Structuring and managing business processes with the aid of information systems has garnered much attention by organizations for decades. Since business processes are usually well-structured and repetitive, the use of information technology to automate them is possible. Workflow management (WFM), the main business process management tool, promises a solution to an age-old problem, which is the analysis, design, implementation, execution, controlling, monitoring, and supporting of business processes with the use of information technology. What is new about workflow management is the explicit representation of the business process logic, which allows for computerized support (van de Aalst, 1999). Workflow management aims at modeling and executing application processes in complex technical and organizational environments. According to Stark (1997), "workflow systems offer a new model for the division of labour between people and computers", and provide "a process control backbone for business processes by mediating the flow of responsibility in a process from person to person and from task to task".

1.1 Motivation

For several years workflow management systems (WFMS) have been announced as the next best-selling computer application (White and Fischer, 1994; Koulopoulos, 1995). "But up to now they have not attained the success of other packages such as productivity tools, e-mail systems, web-browsers and even groupware platforms", as observed by Agostini and De Michelis (2000). Why do workflow management systems remain in limbo? The essential reason is their lack of a formal theoretical foundation (Jablonski, 2000).

Since workflow is an explicit representation of the business process, we believe process modelling can be a promising solution to analyze this domain of interest and to design a workflow system. In this thesis, we choose a specific process modeling method, BWWP, because it is both comprehensive and simple.

1.2 Thesis Objectives

The purpose of the thesis is to discover the theoretical foundation for workflow management, and to investigate the procedures of building a theory-based workflow management model that could be easily used to guide the analysis and designing of workflow systems in practice. The thesis has two objectives:

1. Formalize a theoretical foundation for workflow management in two steps:

    - Associate workflow management with BWW Ontology (an ontology presented by Bunge and extended by Wand and Weber), and discover the construct mapping rules between the basic concepts of each other.
    - Follow the BWWP-WFM construct mapping rules and generate a theory-based workflow management model, which can be used for the analysis and design of a workflow system.

2. Generate a method for using the model in business practices by following the mapping rules and WFM model. To be specific, the method involves transforming a process model, which is the result of analyzing the domain of interest by using any modeling language that is frequently used as well as BWW Ontological concepts, into a WFM model, which can be a formal representation of a workflow system.

1.3 Thesis Outline

This thesis consists of six chapters.

Chapter Two provides an overview of the concepts of workflow management. The workflow management system is also discussed. Generic constructs of workflow management and their relationship to each other are presented with the relationships between each other, followed by an investigation into the requirements of the workflow management model. Some related works are examined with respect to the requirements.

Chapter Three investigates the theoretical foundation of the WFM by looking at BWW Ontology concepts. An ontological process model is presented which will be extended in chapter four for the purpose of discussing workflow management in ontological context.

Chapter Four presents an ontological model specifically designed for workflow management. The model is an extension of the ontological process model in chapter three with additional workflow-related constructs. By analyzing how things communicate from an ontological perspective, a set of WFM-BWW mapping rules are found. The workflow management system is then introduced as a controller to monitor and control the execution of business processes. Based on the ontological

analysis of workflow constructs and the controller's controlling mechanism, the ontological workflow model is built.

Chapter Five focuses on the application of the ontological model for WFM. A conceptual modeling algorithm on how to operate the model in business practices is generated, and an example is given to illustrate the procedures of running the algorithm.

Chapter Six summarizes the thesis, including its contribution, limitations and future research.

Chapter Two

AN OVERVIEW OF WORKFLOW MANAGEMENT

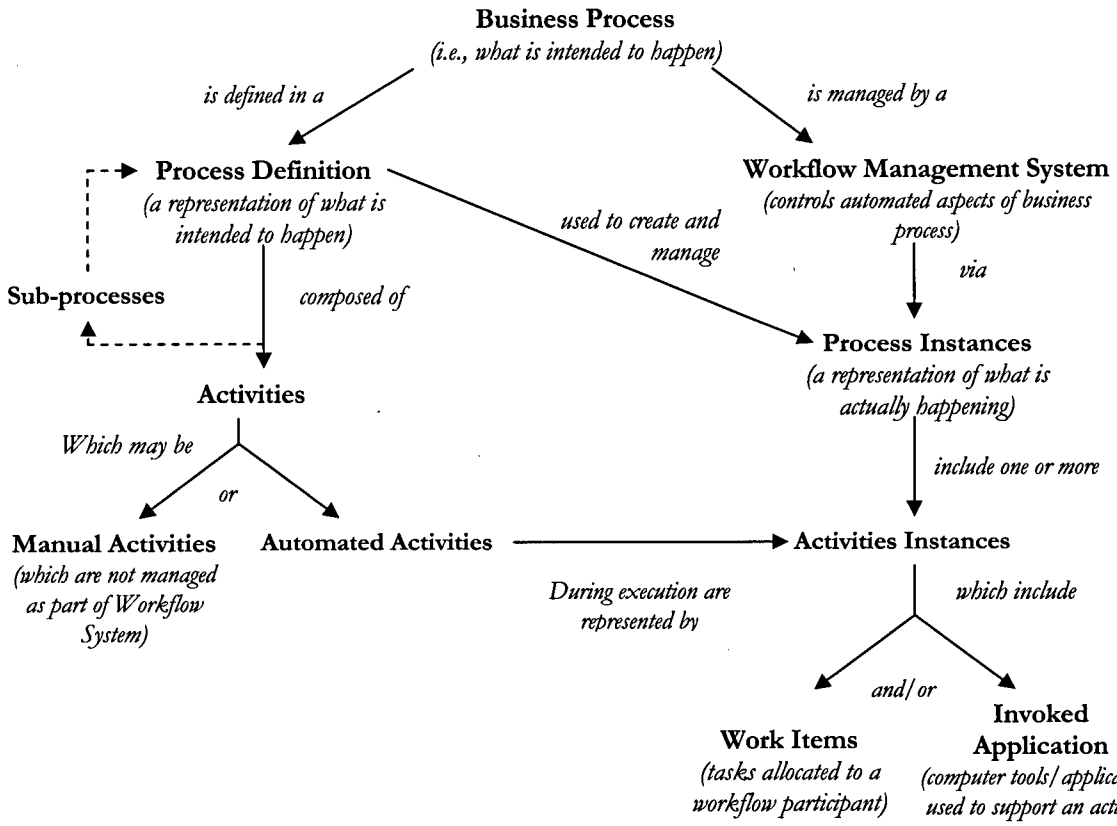
2.1 Overview of Workflow Management

Workflow is the automation of a business process, or a set of coordinated activities representing a business process within a company or an organization (Borghoff and Schlichter, 2000). Other definitions include, Jablonski and Bussler's, who state that workflow "typically consists of various activities, which have to be executed in some order involving multiple collaborating persons call agents in a distributed environment to fulfill a certain task in an organization" (1996).

Among popular terminologies is the one developed by the Workflow Management Coalition (WfMC) (WfMC, 1999). WfMC is a non-profit international organization of workflow vendors, users, analysts and university/research groups. Its mission is to "promote and develop the use of workflow through the establishment of standards for software terminology, interoperability and connectivity between workflow products". In this thesis, we follow WfMC's terminology unless otherwise stated.

According to WfMC, a **workflow** is defined as the automation of a business process, in whole or part, during which documents, information or tasks are passed from one participant to another for action according to a set of procedural rules. A **business process** is a set of one or more linked procedures or activities which collectively realize a business objective or policy goal, normally within the context of an organizational structure defining functional roles and relationships.

Figure 2-1 Workflow Terminology (adapted from WfMC (1999))



Because the terms "workflow" and "process" are so closely related, researchers sometimes use them interchangeably. In any case, a workflow is basically the execution of a business process. From a modeling point of view, a process model describes the structure of a business process in the real world, while a workflow model gives the representation of business processes for computer support in managing business processes (Leymann and Roller, 1999).

WfMC's terminology can be described in Figure 2-1. Other key terms defined by WfMC are:

• **Activity** - A description of a piece of work that forms one logical step within a process. An activity may be a manual activity, which does not support computer automation, or a workflow (automated) activity. A workflow activity requires a human and/or machine resource(s) to support

process execution; where a human resource is required, an activity is allocated to a workflow participant. An activity is also called a task, step, workflow element, or a process element.

• **Work Item** - The representation of the work to be processed by a workflow participant in the context of an activity within a process instance.

• **Worklist** - a list of work items associated with a given workflow participant (or in some cases, with a group of workflow participants who may share a common worklist).

It should be noted that WfMC further categorizes an activity as 'automated' or 'manual' and normally refers the term 'activity' to 'automated activity', while it regards 'manual activity' as something lying 'outside the scope of a workflow management system' (WfMC, 1999). We believe that 'manual activity' should be 'inside' rather than 'outside' the workflow management systems scope, as the aim of a workflow management system is not necessarily to automate all the tasks of a workflow process. Some tasks might continue to involve humans; and even for automated tasks, the determination of when to initiate them and/or determining whether such automated tasks have been successfully completed might be left to humans. The emphasis is much more on automating the tracking of workflow task states, and allowing the specification of preconditions to decide when tasks are ready to be executed (inter-task dependencies) and of information flow between tasks (Mohan, 1997).

2.2 Workflow Management System

A **workflow management system (WFMS)** is a system that defines, creates and manages the execution of workflows through the use of software, running on one or more workflow engines,

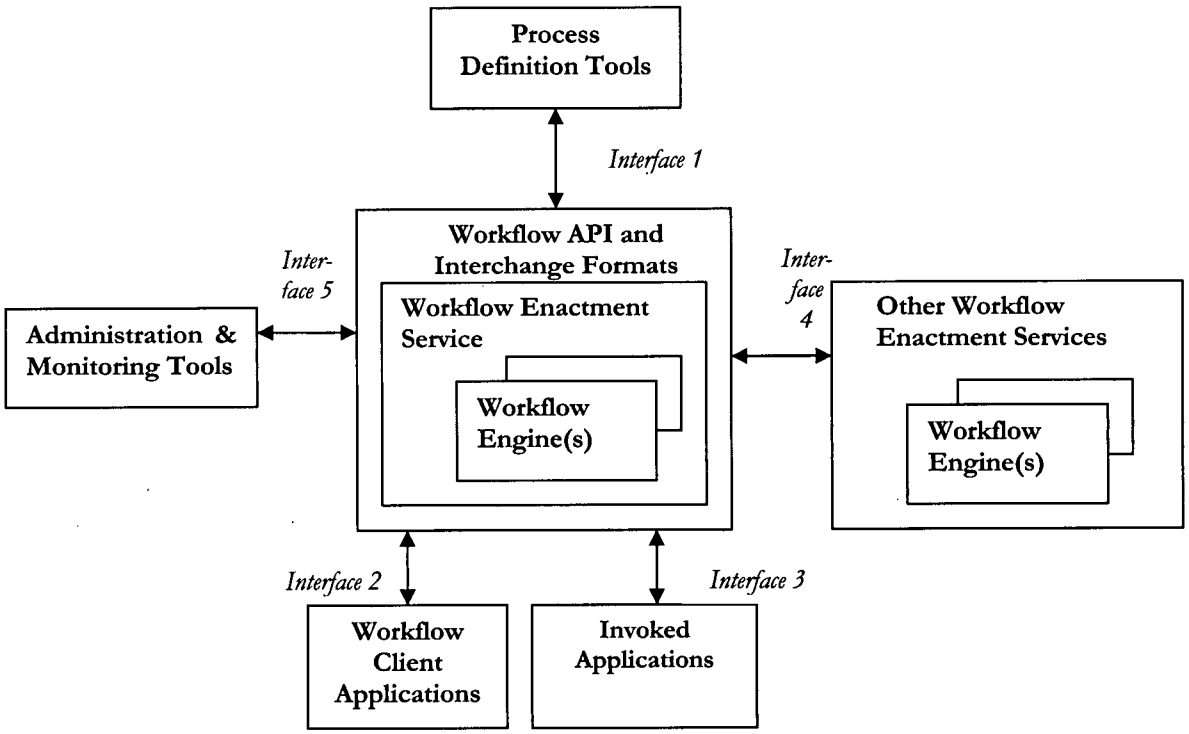
which is able to interpret the process definition, interact with workflow participants and, where required, invoke the use of IT tools and applications (WfMC, 1999).

Despite the differences of modeling and design among different researchers and software vendors, generally speaking, all workflow management systems should support three main functions (Divitini et al., 2001):

- The definition of workflows;
- The execution of workflows;
- The administration of workflows and the monitoring of their execution.

To standardize a common architecture for their functions, the Workflow Management Coalition has developed a Workflow Reference Model from the generic workflow application structure by identifying the interfaces within this structure that enable products to interoperate at a variety of levels (WfMC, 1995) as shown in Figure 2-2.

Figure 2-2 Workflow Reference Model (adapted from (WfMC 1995))



The Workflow Reference Model consists of six components:

• **Workflow Enactment Service** – A workflow system's core component, which supports the execution of workflow processes by providing a run-time execution environment. The workflow Enactment Service is defined as a software service that may consist of one or more workflow engines that create, manage and execute workflow instances. Applications may interface to this service via the Workflow Application Programming Interface (WAPI).

• **Process Definition Tools (Interface 1)** - Used to analyze, model, describe and document a business process; such tools can be either informal or highly formalized and may be supplied as part of a workflow product or as a separate toolset.

• **Workflow Client Applications (Interface 2) and Invoked Applications (Interface 3)** – work together to enable end-users to interact with the workflow systems or third-party applications, e.g., ERP systems, to perform a task.

• **Administration and Monitoring Tools (Interface 5)** – Enables administrators to manage, monitor and analyze actual processes.

• **Other Workflow Enactment Services (Interface 4)** – Those from other workflow systems and are connected via interface 4 for inter-operations.

To make a WFMS capable of supporting these functions and interfaces, some essential features are taken into account. With these functions, we can list the requirements from a WFMS as follows:

Flexibility -- Supporting changes. One of the most significant challenges of a WFMS is to respond adaptively to changes. Changes may vary from ad-hoc modifications of the process for a single customer to a complete restructuring for the workflow process to improve efficiency (van der Aalst, 1999). The complication of workflow management systems affects their adaptability and changeability. It is claimed that the lack of adequate flexibility can be considered the primary reason why workflow management systems have been so awkward in dealing with changes (Agostini and De Michelis, 2000).

Flexibility -- Supporting exception handling. Exception handling mechanisms are basic features of workflow management systems. Because of the dynamic and ever-changing nature of business, a process description can rarely capture all the possibilities of what will occur. Exceptions are inevitable in the real world. Unfortunately, they are not adequately addressed. Generally either

they are too restrictive with respect to the needs of their users or they are too complicated both with respect to the performances of the workflow management systems and the usability of workflow models. We believe that the lack of flexibility is the main reason for this situation.

Interoperability. Workflow engines rarely work alone. Interoperability is one of the key objectives for workflow system design. The five interfaces defined in Workflow Reference Model (Figure 2-2) are the most visible standards set up by WfMC. The WfMC's Interoperability White Paper identified eight levels of interoperability from "No Interoperability" to "Common Look and Feel Utilities", and had the tests for workflow vendors to demonstrate the interoperability of their products. Since different workflow engines are founded on different conceptual models and supported by varying technologies, they have different behavioural characteristics and capabilities that affect the way in which they can support interoperability with other workflow engines. There is a need for workflow models to have a common theoretical foundation in order to communicate well.

Comprehensiveness. There are several dimensions covered in the workflow schema. These dimensions are widely known as workflow perspectives (Jablonski and Bussler, 1996). While the set of perspectives depends on the particular requirements of the application, the following perspectives are present in almost all workflow applications: The *functional* perspective specifies what has to be done within a workflow. The *operational* perspective determines how it is done, i.e., which programs are used to perform particular workflow activities. The *behavioral* perspective defines when and under which conditions a workflow is executed. Start condition or transition conditions are typical language constructs to specify the behavioural perspective. The *informational* perspective specifies the data objects that are manipulated during workflow executions and the

flow of data between workflow activities. Data flow connectors and parameters can be used to specify the information perspective. Finally, the *organizational* perspective describes the roles and personnel that are involved in workflow executions. It is argued in this thesis that workflow management systems should incorporate different perspectives of workflow to support application integration and the various roles of the actors involved.

With respect to these features, most existing workflow management systems appear to be inadequate. As Agostini and De Michelis noted, "it is difficult to interrupt them, to exit their normal flow and then re-enter it. They are based on complex and sometimes multiple process models (integrating data models, normal and exceptional flows, role descriptions) whose changes and exceptions need careful and time consuming analysis. They need to be designed by expert programmers, introducing a time delay between process and workflow changes. They do not support multiple viewpoints on the process, corresponding to the various actors with the different objectives and roles they support (the managers and those who have to control and evaluate the process execution, the task performers and the customers waiting for its outcomes" (2000).

Other observers have also argued that most workflow management systems make business processes too rigid, not allowing their users to react freely to the breakdowns that occur during their evolution (Bowers et al., 1995). Some seem to blame this rigidity on the use of formal theory based workflow models (formal models cannot fully capture the knowledge people use while acting within a business process); others insist on their introducing strict coupling between modeling and executing (models should be cognitive artifacts; Norman, 1991; not constraining the behaviour of the actors; Suchman, 1987; Dourish et al., 1996).

In this thesis it is believed that the rigidity of existing workflow management systems should be attributed neither to their using formal theory based models nor to their coupling of modeling and execution; on the contrary, it is the lack of formal theoretical foundation that is seen as the cause of rigidity (Jablonski, 2000). Formal theory based models can contribute to the solution of the above problems. There are many formal theory based models in literature, but they are based on theories from adjacent area, most frequently computer science (Jablonski,2000).

In fact, a good workflow model offers effective tools for creating a process modelling environment exhibiting the following properties (Agostini and De Michelis, 2000):

- it allows formal verification of some workflow properties;
- it allows us to obtain the analysis results of workflow systems through running the algorithm of the model;
- it supports multiple views of the process;
- it automatically enacts model changes on the running instances of a workflow, protecting them from undesired outcomes.

In this thesis, we focus on process definition and workflow enactment service. W we will propose a Bunge-Wand-Weber Ontology (BWW Ontology) based workflow model for representing, executing and controlling workflow and exception handling. We choose BWW Ontology as theoretical foundation because it is a well-established theoretical domain within philosophy dealing with models of reality.

2.3 Related Works

Since the advent of workflow management technology in the beginning of the nineties, a lot of contributions to this area were made. Several hundreds of business process models have been

proposed by the research literature and by commercial WFMSs in the last decade. However, the overall goal to develop a generally applicable theory and a conceptual basis for workflow management still has not been met. According to Jablonski (2000), "very many contributions narrow down to a very particular issue of workflow management; they find a tricky and often very valuable solution for a niche problem". Nevertheless, a step forward towards a generally applicable, conceptual model for workflow management is still not made. Some of the contributions are focused on specific aspects of WFMSs, which are limited in their lack of an overall view so that they failed to provide a powerful solution. Others provided solutions using specific languages, which are limited in the failure of ensure the compatibility of the specific language with the workflow modeling language. Three peculiarities characterize contributions in the workflow management area (Jablonski 2000):

1) Contributions are too pragmatic.

Many researchers state that workflow management is a discipline driven totally by the application. Since in many applications, theory is not directly applicable, they conclude that a formal theory for workflow management is also not relevant. This results in "How-I-did-it" approaches and solutions, which contribute nothing but singular solutions to specific problems with only minor general value.

2) Contributions claim to be general but are not.

There is another group of researchers that are convinced that a formal theory is necessary for workflow management. Thus, many contributions present approaches towards this goal. Among

other things, they introduce meta-models that ought to define a global concept for workflow management. Nevertheless, most of these approaches cannot fulfill what they promise.

3) Contributions are too much influenced by adjacent research areas.

Workflow management is a discipline on its own. However, many researchers stem from other, mostly adjacent research areas. Database management, in particular transaction management, and Petri Nets are two very dominant examples. "Thus, often theories developed within one of these adjacent research areas are transferred into workflow management without critical assessment and necessary adjustment. The question whether these borrowed theories are applicable at all in the context of workflow management is asked too rarely" (Jablonski, 2000).

Related works are examined, with respect to the three categories mentioned above. Casati, Ceri, Paraboschi, and Pozzi (1999) presented Chimera-Exc, a language for the specification of exceptions for workflows based on detached active rules. Dickson K.W. Chiu (2001) proposed a three-level model to provide semantic recovery support for exception handling in an object-oriented workflow environment. Lee, Han, and Shim (2001) defined a workflow definition language to detect all the possible access conflicts within concurrent workflow processes. These approaches are developed to provide specific problems with single solutions in some aspects of workflows without considering an integration with the overall workflow analysis and design.

Agostini, De Michelis and Petruni (1994) proposed MILANO workflow management model; Agostini and De Michelis proposed MILANO workflow management system, both are based on Elementary Net System, a subclass of Petri Nets, which borrowed theory from Computer Science, an adjacent academic area. Wirtz, Weske and Giese (2001) tried to integrate Object-oriented

modeling languages with workflow methodology by proposing Objects Coordination Nets (OCoN), which employing UML diagrams with some extensions and Petri Nets graphics. This approach was developed alongside theories in adjacent areas.

Casati and Discenza (2001) extended traditional workflow models by using concept of "event node" to allow workflows to publish and subscribe to events, and to enable the definition of points in the process execution where events should be sent or received, so that to realize the interaction and cooperation between workflows. Zeng, Ngu, Benatallah, and O'Dell (2001) proposed an approach which integrated agent with workflows and an agent-based workflow architecture. They introduced the concept "agent" to describe the actors that execute tasks, which are assigned according to agents' capabilities, and monitor workflow execution as well. Dong, Hull, Kumar, Su, and Zhou (1999) presented a framework based on families of "communicating flowcharts" (CFs) for optimizing the physical distribution of workflow schemas, and the mapping of sub-workflow schemas into flowcharts, and developed a set of formulas to quantify the metrics used for choosing a near optimal set of CF clusters for executing a workflow.

To the best of our knowledge, there has been no ontological study done on generic workflow management constructs so far. Additionally, we did not discover any previous work that attempts to find the ontological workflow model. We will propose an ontological model for workflow management later to show that a model based on BWW Ontology is simple, clear, flexible, interoperable and comprehensive.

2.4 Generic Constructs of Workflow Management

Based on previous observations of the workflow management, it is clear that workflow management models define the dynamic parts of a system. It describes the facts about the

following: the participants in the workflow, the tasks triggered by an event or events, the events that trigger a task or tasks, the assignment of tasks to participants, the order of tasks to be performed, the order of events that occur, the resources utilized in a task, and the input and output of a task.

This section attempts to summarize the generic constructs, which are necessary to represent a good process model for workflow management. We generally use WfMC terminology (Workflow Management Coalition Terminology & Glossary, WfMC 1999), but use "activity" to describe "task" because we need to separate the "task" from the "activity" construct in BWW Ontology. We differentiate "agent" from "non-agent" in the construct "workflow participant" according to whether they actively perform tasks. In addition, we add the construct "exception" because it is important for exception handling mechanism in workflow management. The basic generic constructs are discussed below:

Process ------ A process is a set of one or more tasks, which collectively realise a business objective or policy goal, normally within the context of an organisational structure defining functional roles and relationships. It may involve one or many participants, which interact with each other by executing tasks. When all participants stop their state-change, the process is over.

Task ------ A task is a piece of work which forms a logic step in a process. The execution of a task is atomic, which means that once the task starts it will finally complete and the task performer will not change its state unless it is triggered to change again.

Agent ------ An agent is a person, an organizational unit, or a computerized aiding tool that participates in the process. It is the participant that actively executes tasks. An agent can execute a

task and keep changing states until it is stable during the task. It can also activate other participants to start changing.

Non-agent or Resource ------ Another kind of participant in a process is non-agent or resource, which is not able to activate any participants including itself during the process. It is a resource utilized by agents in tasks.

Event ------ An event is an occurrence of a particular condition (which may be internal or external to the workflow management system) which causes the workflow management software to take one or more actions.

Business rules ------ Business rules provide the law or rules which must be followed during workflow; failure to follow a business rule may cause an exception or other defined procedure.

Process state ------ A representation of the internal conditions defining the status of a process at a particular point in time.

Exception ------ An exception is an undesired occurrence that causes error and needs to be fixed (Analyzed in next section).

2.5 Exception

Different definitions of exception can be found in literature. For example, Chiu defines exception as an event which deviates from normal behaviour and may prevent forward progress of a workflow (2001). In Casati, (1998), exceptions are described as "may be caused by system (hardware or software) failures, or may be related to the semantics of the business process". Exceptions can occur when business rules change while process definition is not changed yet,

when organizational structure changes, when agents make mistakes, or when there are system errors.

Business rules are set in advance and all tasks must be performed according to business rules. Business rules decide order of tasks, results of performing tasks, and what actions are to be taken following various results. From the perspective of business rules, we define and differentiate the concepts of "exception" and "non-exception". When agents perform tasks, the results can be categorized into four types as follows:

1) Results are anticipated and the following actions are decided by business rules. For this type of results, the following actions are taken according to business rules and the process continues.

2) Results are not anticipated and are outcomes of specification errors of business rules. For this type of results, the following actions can be found after more analysis is performed, and missing specification is found or wrong specification is corrected. The results and following actions need to be recorded as "valid" for future use.

3) Results are not anticipated but relevant business rules can be found. For this type of results, relevant business rules can be found and the following actions will be found according to business rules. They need to be recorded as "valid" for future use.

4) Results are not anticipated and no business rules can be found. For this type of results, no following actions can be found. Agents who perform the tasks need to report to supervisor in usual business situation. The process is halted.

In the first three situations, either results can be anticipated or their following actions can be anticipated according to business rules. We define these three results as "anticipated results", or we can call them just "results". The last situation is really beyond anticipation and no actions can be taken to fix it. We define it as the real "exception".

For example, in the process of "mortgage application review", the agent "bank clerk" performs a task "review applicant's income". The business rules for this process: approve the application when the applicant's annual income is no less than 80000 CAD, and the applicant has no more than 3 negative credit records. Various results could appear and would be analyzed with respect to the categorization above:

The applicant's annual income is less than 80000 CAD. This result falls into the first type. The application is rejected.

The applicant's annual income is less than 80000 CAD, but his employer offers one extra benefit that is not counted into income according to the mortgager's business rules, and the total of annual income and that benefit amount is more than 80000 CAD. This result falls into the second type. Business rules need to be modified to count that benefit into income. The application should be approved.

The applicant's annual income is less than 80000 CAD, but the applicant is getting married very soon, and the total of his annual income and his fiancée's annual income is more than 80000 CAD. One business rule need to be added: the income of the applicant's spouse-to-be need to be counted in. The application should be approved.

The database that contains applicant's credit information is not working. This result falls into the forth type. No business rules could be found to fix it. The process is halted. Human intervention is needed.

For the first type of results, we suggest a certain form, a "Process Knowledge Repository", to store them. We call it "Process Knowledge". It also stores the following actions for each result. It can be a list, a database, or a function. It will be used as a way of operationalizing non-exception and exception.

The second and the third result categories are actually caused by mistakes in analysis or design. What needs to be done to fix them is to perform more process modeling analysis to correct the mistakes. In this thesis, we refer to the second and third types of results as "design errors". They need to be fixed by performing more analysis and the changes will be recorded as valid results in the "Process Knowledge" for future use. In our thesis we will provide a mechanism to handle the errors, which will be shown in chapter four. We have these two categories in non-exception because such situations would happen in reality. Business rules are modified and completed over time.

An exception-centric workflow management system framework is proposed by Dickson Chiu (2000) in his PhD dissertation. It defines exceptions as "events that deviate from normal execution behavior" (Chiu, 2000. P24). An event is "a general form of trigger in DBMS implementation" (Chiu, 2000. P11). Chiu classified exceptions into two categories: *expected exception*, and *unexpected exception*. Expected exceptions are "those anticipated or already planned with explicit exception handlers" (Chiu, 2000. P26). Unexpected exceptions, on the other hand, "require human intervention since they are unanticipated" (Chiu, 2000. P26). Based on the above, we claim that

Chiu's definition is derived from implementation aspects of computation. The taxonomy of exceptions is based on the different exception handling mechanisms, which is rooted in implementation issues.

Some research works (Eder and Liebhart, 1995; Basu and Kumar, 2002; Casati, 1998) have the similar definition and classification of exception to Chiu's work. Our definition of exception is quite different from theirs. We will propose an ontological model of process and workflow. Specifically the exceptions will be linked to the concept of laws. Laws will be manifested as business rules. We define exception on the basis of business rules. Business rules control the execution of a process. A workflow is enacted by the workflow management system which follows business rules. In our definition, exceptions are the occurrences that are not defined by business rules, so that they are beyond process knowledge known to workflow management system. Therefore, there is no automatic mechanism for handling them. Our definition attempts to reflect the natural view of the organizational business process in reality. We compare the two different definitions in the following table.

	Our definition	**Chiu's definition**
Theoretical foundation	Ontology	Implementation concepts
Those events that are unanticipated and cannot be handled by automatic handlers	Exception	Unexpected exception
Those events that are anticipated but are not desired	Design error	Expected exception

Table 2-1: Comparison Of Exception Definition

From the table of comparison, we can find some similarities between the two definitions. In Chiu's work, unexpected exceptions are those deviations that are unanticipated and no automatic

exception handlers are available for them so human intervention is needed. This parallels the meaning of exception in our work. In Chiu's work, expected exceptions are those deviations that can be anticipated but are not desired normal behaviors. They can be fixed by explicit exception handlers which may have been supplied by the WFMS, the workflow administrator, or the user. This parallels the meaning of design errors in our work. The two definitions are based on different theoretical foundation and aim at different aspects of workflow management. Chiu's definition serves the purpose of implementation aspect, while our definition serves the purpose of modeling the business. In addition, Chiu's definition does not explain the concept of "normal behavior", therefore does not provide a way of operationalizing the exception definition. In contrast, we propose a "Process Knowledge" as a way of operationalizing exceptions and non-exceptions.

In chapter four, we will show an analysis and a way of operationalizing of exception handling using our definition of exception from an ontological perspective.

2.6 An example of workflow management

To understand the workflow management, we need to offer a realistic example. This is a typical business process in a large enterprise within workflow management context.

The enterprise has a certain procedure for procurement for the sake of payment security and consistency. We consider the procedure of purchasing computer in our example. There are six steps in the example: 1. Buyer selects the computer. 2. Buyer requests the procurement. 3. Procurement manager approves the procurement. 4. Buyer contacts the supplier. 5. Account payable staff makes payment with cheque. 6. Computers arrive. For the ease of reference in this thesis, the example will be referenced as "Computer Procurement" in later chapters.

The triggering event is "enterprise needs computers". When the triggering event occurs, the buyer will collect information, compare various computers and select an appropriate type. Then the buyer will request the procurement.

The procurement manager will approve or disapprove the procurement. If the request not approved, the buyer has to select another type of computer. If the request approved, the buyer will contact the supplier.

If the computers in exact quantity are in stock, receiving the notice of it, the account payable staff will make payment with cheque. If out of stock, the enterprise may have to wait until the supplier has them later.

Finally, the computers in exact quantity arrive. The buyer receives these computers. The procurement process is finished.

In this example, buyer, manager and A/P are agents; the computer information, mail, cheque, and are used resources. Possible exceptions could be: manager is absent so that the procurement request has to be suspended; some copies of documents are missing so that A/P cannot make payment immediately; the supplier has run out of business; etc.

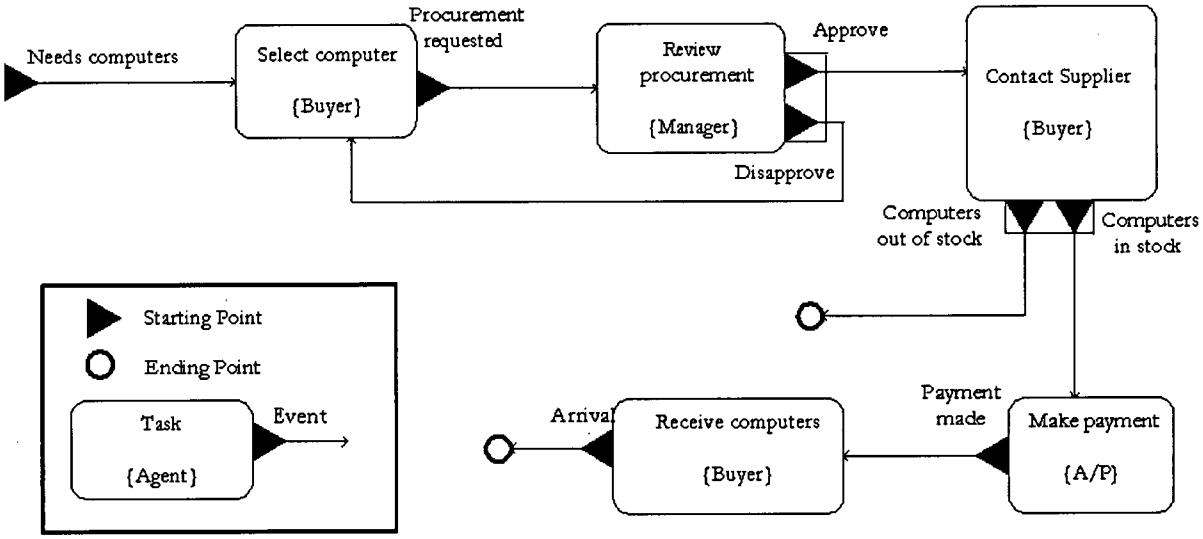
Starting event	**Triggered task**	**Ending event**	**Agent**	**Resource**
Needs computers	Select computer	Procurement requested	buyer	Computer information
Procurement requested	Review procurement	Request approved or disapproved	manager	Computer information
Request approved	Contacts supplier	Computers in stock	buyer	-
Request approved	Contacts supplier	Computers out of stock	buyer	-

Request disapproved	Re-select computers	-	buyer	Computer information
Computers in stock	Make payment	Payment made	A/P	Mail, cheque
Out of stock	-	-	-	-
Payment made	Receive computers	Computers arrive	buyer	-
Procurement closed	-	-	-	-

Table 2-2: Definition of "Computer Procurement" process

The "Computer Procurement" process is illustrated with Event-driven Process Modelling (EDPM) in Figure 2-3.

Figure 2-3: "Computer Procurement" process in EPDM



Chapter Three

THEORETICAL FOUNDATIONS

3.1 Objective of this chapter

The objective of this chapter is to examine the theoretical foundations of Workflow Management Model by the following steps:

- Introduce the well-defined BWW Ontology.
- Introduce the basic BWW Ontological constructs, and process related BWW Ontological constructs.
- Analyze the process model from the ontological point of view.

3.2 Choosing BWW Ontology

Ontology is a well-established theoretical domain within philosophy dealing with models of reality. Wand and Weber have adopted an ontology presented by Mario Augusto Bunge (1977, 1979) and applied it to the modeling of information systems (Wand and Weber, 1989, 1990, 1993). The thesis refers to this ontology as the BWW Ontology. Bunge's ontology is chosen as the foundation for our work due to the following strengths:

It is a well formulated theory.

It has been applied in a number of studies related to information systems modeling and object-oriented concepts, thus providing a benchmark for evaluating other models (Green and Rosemann, 2000; Wand and Weber, 1993).

It has been used to suggest an ontological meaning to object concepts (Wand, 1989).

It has been used to generate predictions about conceptual models that have been corroborated empirically (Gemino, 1999; Weber and Zhang, 1996).

3.3 Original BWW Ontological Constructs

The modeling constructs in BWW Ontology can be organized into the following four categories (Wand and Weber, 1990; Zhao, 1995):

1. Static model of an individual:	1) Thing, composite thing; 2) Property, intrinsic property and mutual property, hereditary

	property and emergent property; 3) State, conceivable state space, state law; 4) Class, kind, natural kind;
2. Dynamic model of an individual:	1) Event, conceivable event space; 2) Transformation, transformation law; 3) History;
3. Static model of a system:	1) Coupling; 2) System; 3) System composition, system environment, system structure, system decomposition, subsystem, level structure;
4. Dynamic model of a system:	1) Stable state and unstable state; 2) Internal event and external event; 3) Well-defined event and poorly defined event.

Table 3-1: Original constructs of BWW Ontology

Their ontological meaning will be explained with WFM related BWW Ontological constructs together in section 3.4.

3.4 Process related BWWP Constructs

Some of the original BWW Ontological concepts are closely related to the Process Modeling concepts, and those Process-related BWW Ontological concepts are referred to as Process-related BWWP constructs. The Process-related BWWP constructs include the original BWW Ontological constructs of thing (simple thing and composite thing), property (intrinsic property and mutual

property, hereditary property and emergent property), state (stable state and unstable state), event (internal event and external event), transformation, and law (state law and transformation law).

The basic BWW Ontological constructs can be summarized as follows (Evermann and Wand, 2001a; Wand and Weber, 1995; Wand and Weber, 1990b):

The world is made of substantial *things*. Things possess *properties* that are either *intrinsic* or *mutual*. A property could be *intrinsic* if it is possessed by the thing itself (e.g. colour of a car), or it could be *mutual* if it is possessed by two or more things (e.g. distance between two cars). Thing can be categorized into two types: actor and non-actor (Wang, 2002). They are difference in the way of ability to change.

Things can combine to form a *composite* thing. Composite things possess *emergent* properties that are not possessed by any component (e.g. the speed of a car is the emergent property of a car). The properties that are only possessed by components in a composite thing are *hereditary* properties (e.g. the horsepower of car engine).

Attributes are representations of the properties of a thing as perceived by an observer. They can be modeled in terms of *state functions* that are functions on time. A set of attributes used to describe a set of thing with common properties is called a *functional schema* – common sets of attribute function. The set of values of the state functions at a given time comprises the *state* of the thing.

A state may be *stable* – which means it will not change until the thing is affected by another thing, or *unstable* – where the thing will undergo a spontaneous state transition.

A *law* is a relationship between properties. In particular, a law can be specified in terms of precedence of properties: Property A precedes property B if and only if whenever a thing possesses B, it possesses A. The state change from one to another over time is a *transformation.* The transformations a thing can undergo are determined by its *transformation laws.* The *state laws* restrict the values of the properties of a thing to a subset that is deemed lawful because of natural laws or human laws.

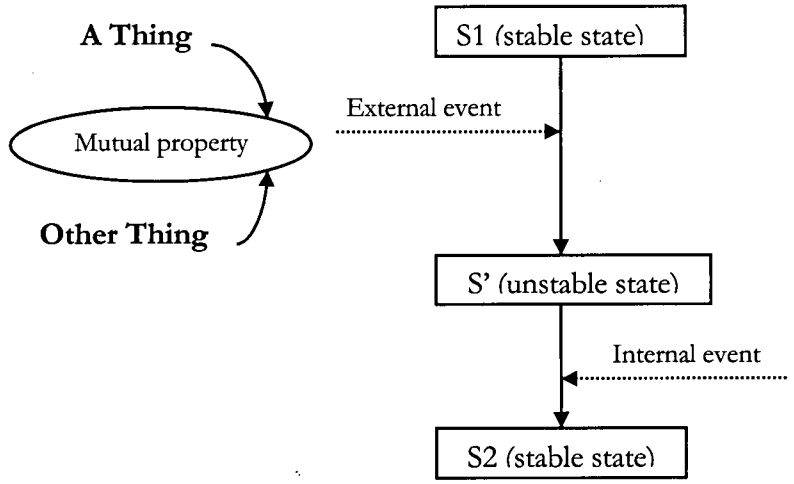
Interaction is defined through the state history of a thing: If state changes in one thing depend on the presence of another, the second is said to act on the first. Interactions may be described in terms of *changes in mutual properties* (Evermann and Wand, 2001a).

A change of state is termed an *event.* An event can be described as an ordered pair of states <s1, s2> where s1 and s2 are the state before and after the change, respectively. An *external event* is an event that arises in a thing, subsystem or system by virtue of the action of some thing in the environment on the thing, subsystem, or system. The before-state of an external event is always stable. The after-state of an external event may be stable or unstable. An *internal event* is an event that arises in a thing, subsystem, or system by virtue of lawful transformations in the thing, subsystem, or system. The before-state of an internal event is always unstable. The after-state may be stable or unstable (Wand and Weber, 1995).

For example, a thing is in a stable state s1. The thing has a mutual property with another thing. An interaction with another thing by changing their mutual property can cause the thing changing its state, say to s'. We shall call <s1, s'> an external event. Now, if the state s' is unstable, then according to the assumptions the thing will keep changing its state until it reaches a new stable state s2. Note, s2 is actually defined by the state law of the thing. The event <s', s2> is an internal

event for it does not interact with other things during the state change. The internal event does not depend on the interaction with other things but only on the lawful transformations that are intrinsic characteristics of the thing itself (Wand and Weber, 1990b). Figure 3-1 illustrates this process.

Figure 3-1: BWWP-Event & State



Besides the original BWW Ontological concepts introduced above, some concepts are added in Process-related BWWP constructs. They are described as follows (Wang, 2002):

A thing can be differentiated into two types: an *actor* or a *non-actor*. An *actor* is a thing that changes the state of itself or at least one other thing (actor/non-actor). A *non-actor* is a thing that does not change the state of any thing including itself. The state of a non-actor can only be changed by an actor. For example, when a clerk walk, her state is changed by herself and she is an actor. When a clerk uses a telephone set, the clerk is an actor for he changes the states of both himself and the telephone set. The telephone set is a non-actor as it is not able to change the state of itself, nor is it able to change any other things. Its state could only be changed by the clerk.

A process involves a set of actors and non-actors who interact with each other. The process is *initiated* when at least one actor is changed from a stable state to an unstable state. One or more events can trigger a process. The process is *completed* when all actors and non-actors finish state changes and are in stable states.

3.5 Ontological Model of a Process

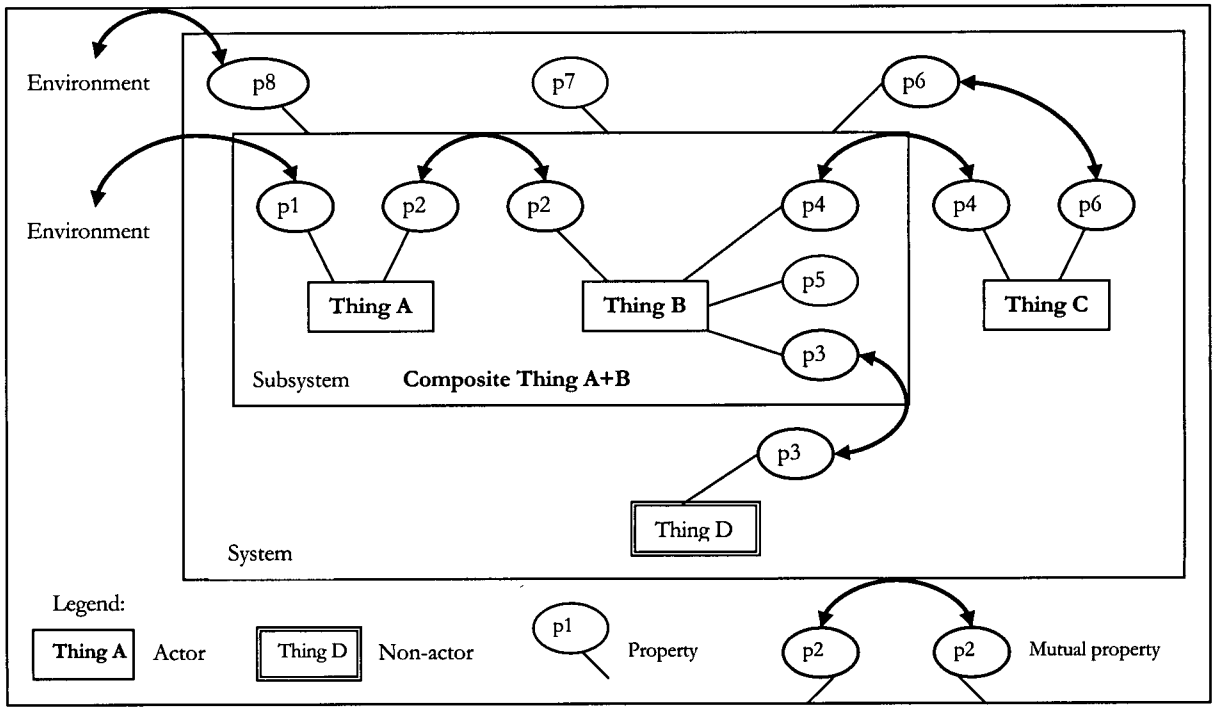
An ontological model of process (Wang, 2002) is the description of a business process using ontological concepts.

The model starts with analyzing how several things interact in a system in terms of BWWP concepts. Some questions need to be clarified, such as: What is the system like before the process starts? How is the process initiated? How do things interact with each other? When do things stop changing? What is the state of the system when the process is completed?

3.5.1 A System of Six Things

To capture the essentials of an ontological process model, an ontological analysis is needed for a generic scenario of a system that is composed of several things (including actors and non-actors, simple or composites). The ontological analysis here is performed by considering the state-change happening to each thing in the system. The scenario being used here is described in Figure 3-2. For simplicity, we do not distinguish the concepts between *property* and *attribute*.

Figure 3-2: A system of Six things (adapted from Wang, 2002)



There are four individual things: A, B, C, and D. There is one composite thing A+B. A, B, and C are actors, and D is a non-actor, which only interacts with B. A has properties: p1, and p2. The mutual property of A and the environment is p1. B has properties: p2, p3, p4, and p5. The mutual property of A and B is p2. The mutual property of B and C is p4. Non-actor D has mutual property p3 with thing B. It does not have any other properties. Composite thing A+B is also an actor, it has emergent properties p6, p7, and p8. p6 is the mutual property of C and A+B. The mutual property of composite thing A+B and the environment is p8.

3.5.2 Ontological Analysis

The interactions between the four things are described from BWWP ontological perspective in Figure 3-3. For clarity, all the stable states are labelled as *s1, s2, s3,* ... and all unstable states are labelled as *u1, u2, u3,* ... Based on the definitions of BWWP concepts, the analysis in this section attempts to find the ontological principles in a process, and demonstrate them by the example. The ontological principles found in this section will be summarized and extended in the ontological WFM model in chapter four. The detailed analysis is presented as follows (Wang, 2002):

- All things stay in stable states (s1) before the process starts.

Demonstration: Before the process starts, A, B, C, and D are all in stable states.

- The process is activated by an interaction between a thing in the system and the system environment. This interaction is initiated by a thing in the environment to change the mutual property of a thing in the system.

Demonstration: A thing in the environment changes the value of p1, the mutual property of the thing in the environment and actor A, and triggers A to change from stable state s1 to unstable state u1. The process is activated now.

- Based on BWW Ontological principle of interaction, all things interact by changing the value of mutual properties.

Demonstration: A thing in the environment interacts with actor A by changing the value of mutual property p1. Actor A interacts with actor B by changing the value of mutual property p2. Actor B interacts with non-actor D by changing the value of mutual property p3. Actor B interacts with actor C by changing the value of mutual property p4. Actor C interacts with composite actor

A+B by changing the value of mutual property p6. Composite actor A+B interacts with a thing in the environment by changing the value of mutual property p8.

➢ When an actor modifies its intrinsic properties, it does not affect any other things but only changes the state of the actor itself.

Demonstration: Actor B modifies the value of p5 (intrinsic property) and this only causes B to continue changing. Composite actor A+B modifies the value of p7 (intrinsic property) and this only causes A+B to continue changing.

➢ After a thing starts change, it modifies all its properties and eventually reaches a new stable state.

Demonstration: Actor A stops changing after all its properties (p1 and p2) are modified, and stays in a new stable state (s2). Actor B stops changing after all its properties (p2, p3, p4 and p5) are modified, and stays in a new stable state (s2). Actor C stops changing after all its properties (p4 and p6) are modified and it stays in a new stable state (s2). Composite actor A+B stops changing after all its properties (p6, p7 and p8) are modified, and stays in a new stable state (s2). Non-actor D stops changing after its only property p3 is modified, and stays in a new stable state (s2).

➢ A non-actor is not able to change any thing including itself. Therefore, all its properties are mutual with actors, and its intrinsic properties are of no interest in the model. Its state changes are always caused by actors through changing the values of the mutual properties, and they are always from a stable state to another stable state.

Demonstration: Non-actor D does not have any intrinsic properties. D changes state because its mutual property p3 with another actor B is changed value. And it changes from a stable state (s1) to another stable state (S2).

➢ The interaction between a composite thing and a simple thing or another composite thing follows the same principle: the interaction is performed by changing the value of mutual properties.

Demonstration: Actor C interacts with composite actor A+B by changing the value of mutual property p6.

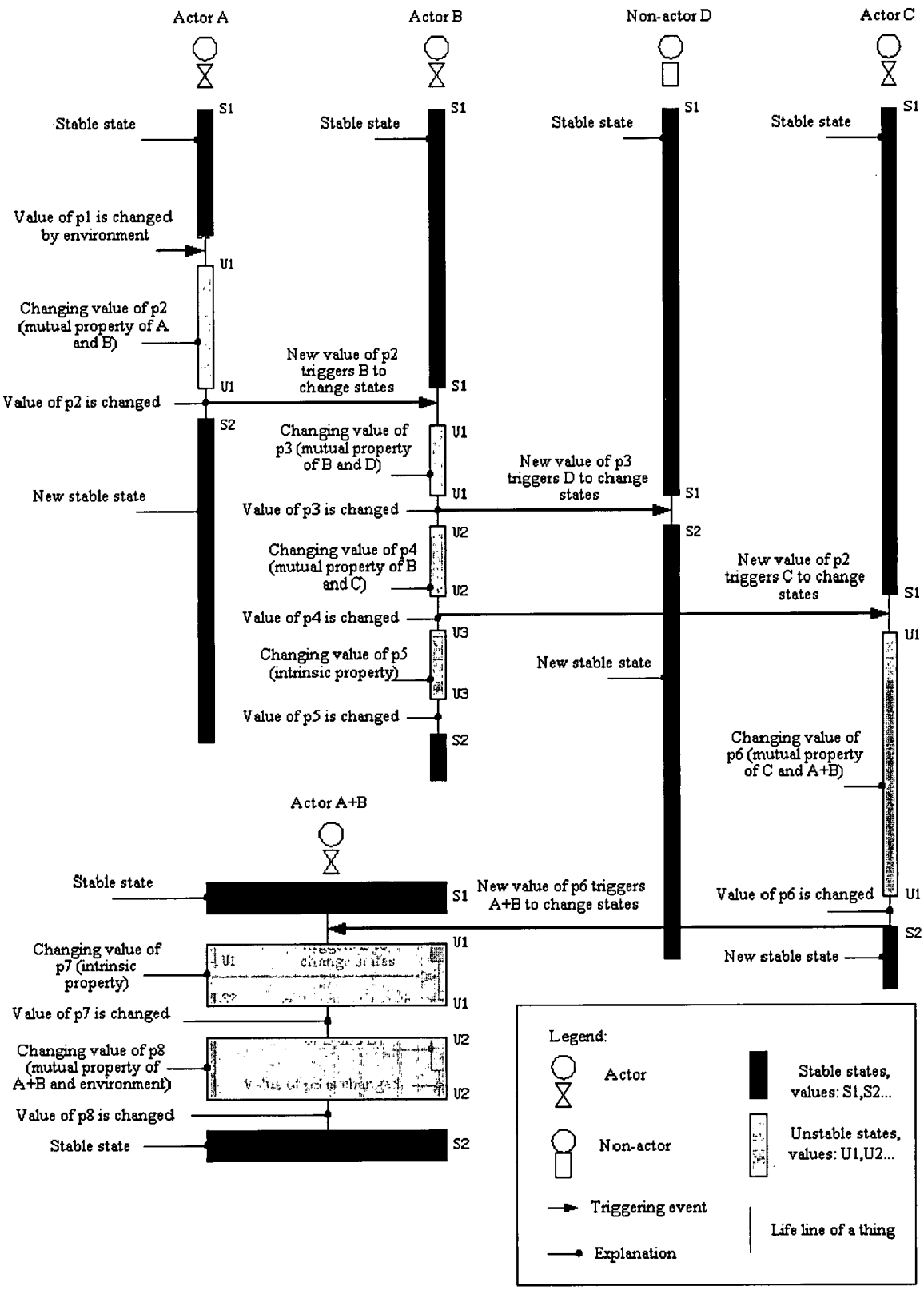
➢ A composite thing and its components are observed from different standpoints.

Demonstration: When we observe composite actor A+B as one thing, we have no idea what happens in its component A and B. Similarly, when we observe actor A and B separately, we have no idea what happens to the emergent properties of composite thing A+B.

➢ The process is completed when the last thing (actor or non-actor) stops changing states and all things are in stable state.

Demonstration: A stops changing first, and D, B, and C stops later on. The last thing A+B stops finally and then the process is completed.

Figure 3-3: Ontological analysis to the system of Six things (adapted from Wang, 2002)


Actor A
Actor B
Non-actor D
Actor C
Stable state
S1
Value of p1 is changed by environment
U1
Changing value of p2 (mutual property of A and B)
Value of p2 is changed
S2
New value of p2 triggers B to change states
Changing value of p3 (mutual property of B and D)
New stable state
Value of p3 is changed
New value of p3 triggers D to change states
Changing value of p4 (mutual property of B and C)
U2
Value of p4 is changed
New value of p2 triggers C to change states
Changing value of p5 (intrinsic property)
U3
Value of p5 is changed
New stable state
Changing value of p6 (mutual property of C and A+B)
Actor A+B
Value of p6 is changed
New value of p6 triggers A+B to change states
Changing value of p7 (intrinsic property)
Value of p7 is changed
Changing value of p8 (mutual property of A+B and environment)
Value of p8 is changed
Legend:
Actor
Non-actor
Triggering event
Explanation
Stable states, values: S1,S2...
Unstable states, values: U1,U2...
Life line of a thing

3.5.3 State Curves

A state curve diagram (Appendix A, adapted from Wang, 2002) is used to analyze the concepts of external event, internal event and transformation in a process. As in the previous section, a demonstration in the example will follow the principles found in the analysis. The detailed analysis and demonstration are presented as follows (Wang, 2002):

- In order to change, things must interact with the environment. Therefore, each thing must be activated by an external event.

Demonstration: A, B, C, D and A+B all start to change when triggered by an external event. These external events are: A (s1, u1); B (s1, u1); C (s1, u1); D (s1, s2); A+B (s1, u1).

- The before-state of an external event is always stable. For an actor, the after-state of an external event is always unstable, so that the actor can keep changing itself or another thing. A non-actor has its external event end with a stable state because it is unable to change anything including itself.

Demonstration: The before-state of each external event is all stable (s1). The end-state of external event for actors A, B, C, and A+B are unstable (u1). The end-state of external event of non-actor D is stable (s2).

- Internal events occur by virtue of transformations in a thing. For an actor, all the subsequent events following an external event are internal events. Since an actor needs at least one transformation to reach a new stable state, there is at least one internal event during an actor's change. Similarly, since a non-actor does not have a transformation, it does not have any internal events.

Demonstration: A has internal events (u1, s2). B has (u1, u2), (u2, u3), (u3, s2). C has (u1, s2). A+B has (u1, u2), (u2, s2). Non-actor D does not have any internal events.

➢ An actor undergoes a transformation when its state is changed from one to another over time. For a non-actor, since its state changes from stable to stable with no time, it does not undergo any transformation.

Demonstration: Actors A, B, C, and A+B undergo transformations. Non-actor D does not undergo any transformation.

➢ After an actor starts to change, whenever it modifies the value of one property, it undergoes one transformation. When an actor modifies all its properties, it undergoes one sequence of transformations.

Demonstration: After A starts to change (p1 is modified by the environment), A modifies another property p2 and undergoes a transformation. After B starts to change (p2 is modified by A), B modifies p3, p4 and p5. Each modification leads B to a transformation. When all of p3, p4, and p5 are modified, B undergoes a sequence of transformations. After C starts to change (p4 is modified by B), C modifies p6 and undergoes a transformation. After A+B starts to change (p6 is modified by C), A+B modifies p7 and p8. Both modifications lead A+B to transformations. When all of p7 and p8 are modified, A+B undergoes a sequence of transformations.

➢ A transformation always starts from an unstable state. It could end with either a stable state or an unstable state.

Demonstration: The only transformation of A starts from an unstable state: u1. It ends with a stable state s2. Each of the transformations of B starts from an unstable state: u1, u2 and u3. They end with unstable states: u2, u3; and stable state: s2. The only transformation of C starts from an unstable state: u1. It ends with a stable state s2. Each of the transformations of A+B starts from an unstable state: u1 and u2. They end with unstable sate u2 and stable state s2.

Chapter Four

AN ONTOLOGICAL MODEL FOR WORKFLOW MANAGEMENT

4.1 Objective of this chapter

The objectives of this chapter are:

- Find out BWWP-WFM construct mapping rules
- Introduce the concept of controller
- Present a model for workflow management in an ontological context.

The workflow management system controls and monitors the process and tasks, therefore we regard WFMS as a controller, who assigns tasks, monitors progress, controls orders, and fixes errors. The ontological workflow management model is built based on the ontological process model, which is presented in Chapter Three, while it has some additional constructs and rules of its own. Given that our WFM model is focused on the dynamic part of process, additional constructs for ontological WFM model are the constructs of process state and exception.

The model will be presented by incorporating the concept of the controller and WFM constructs.

4.2 BWWP-WFM constructs mapping rules

After the detailed ontological analysis of a business process in chapter three, the counterparts of workflow management constructs can be described. We will find a set of mapping rules that map BWW Ontological constructs to workflow management constructs to associate the theoretical

foundation with workflow management. For some of the constructs, such as agent, resource, event, and business rule, we follow Wang's BWWP construct mapping rules (Wang, 2002). For some of the constructs, such as task, process, process state and exception, we will find the mapping rules. The WFM constructs will be analyzed in the following order: Agent, resource, event, task, process, process state, business rule, and exception. To avoid confusion between the two types of concepts, the prefixes of BWWP- and WFM- will be used.

4.2.1 BWWP-Actor

The construct in BWW Ontology that defines the entity to be modeled is "thing", which is defined as a substantial individual that is perceived to exist physically. A BWWP-actor is "a thing that changes the state of itself or at least one other thing (actor/non-actor)".

A WFM-agent is defined as "a person, an organizational unit or a computerized aiding tool that participates in the process". In a process, an agent is the participant that actively executes tasks. There are two types of participants in a process. The active participant who has the ability to take actions in the process and change states of either itself or other participants are considered as agents, and the passive participants whose states are totally affected by agents' behaviours are WFM-resources. For example, the agents could be human beings and organizations such as an enterprise, department or team, and resources could be things used by agents such as documents, computers, or mails.

A WFM-agent could be either a person or an organizational unit, which is also composed of persons and other substantial individuals. For example, an organizational unit such as a company, a department, or a team is the object on behalf of which that human beings participants in the process.

The concept of BWWP-actor extends the concept of "thing" and expresses an active participant that can cause itself or other participants to change states, which has the same implication of WFM-agent concept. Accordingly, we claim that a BWWP-actor is represented by a WFM-agent.

Mapping rule 1: BWWP-actor maps to WFM-agent.

For example, a bank clerk assists a customer to deposit money into his saving account. The bank clerk is an agent. He is an active participant in the process and his behaviour changes the states of customer and the saving account. The clerk is a substantial entity, whose actions cause other participants to change states.

4.2.2 BWWP-Non-actor

In BWWP concepts, a non-actor is defined as a thing that is not able to change the state of any thing including itself.

A WFM-resource is a substantial thing that is utilized or referenced by an agent to assist the execution of the activity. It is not able to activate any participants including itself during the process. The state of a resource can only be changed by agents.

Therefore, we claim that a BWWP-non-actor is represented by a WFM-resource.

Mapping rule 2: BWWP-non-actor maps to WFM-resource.

For example, a bank clerk refers to the database of corporate credit grades when he has to decide whether or not loan to a corporate. The clerk is agent, and the database of corporate credit is resource. It is referenced by the clerk and does not able to affect any participants including itself.

During the process, the database is not changed. For another example, a software company burn their software to blank disks and then sells them. During the process, the disk is resource. It is passive participant and it is changed state after the process. It is not able to change any participants' state while its state could only be changed by other participants.

4.2.3 BWWP-State

In BWWP, the state of a thing is the set of values of properties of the thing at a particular moment. There is no construct of "state" in workflow management, yet there is a construct that expresses the implication of BWWP-state.

A WFM-event is a notable occurrence at a particular point in time, which causes an agent or a resource to change state. It is the cause or the trigger for a task or a process.

From ontological perspective, BWWP-event is defined as a state-change of a thing caused by an action of some other thing, or by the transformations of the thing itself.

It is clear that there is a difference between WFM-event and BWWP-event concepts. WFM-event is actually the starting point of a thing's state-change, while BWWP-event is the state-change itself. They cannot be simply mapped to each other. A WFM-event that occurs to an agent is actually a state vector of values of the agent's properties at the time when an expected fact occurs, which expresses the implication of BWWP-state concept. Therefore, we claim that a BWWP-state of an actor is represented by a WFM-event.

Mapping rule 3: BWWP-state of an actor maps to WFM-event.

The WFM-event can be either *internal* or *external.* An internal WFM-event only involves the thing itself, that is, only a vector of values of the thing's internal properties. An external WFM-event involves not only the thing itself but other participants as well. This kind of event is represented by a vector of values of the thing's attributes representing both intrinsic properties and mutual properties with other participants.

Figure 4-1: WFM-Internal and External Event

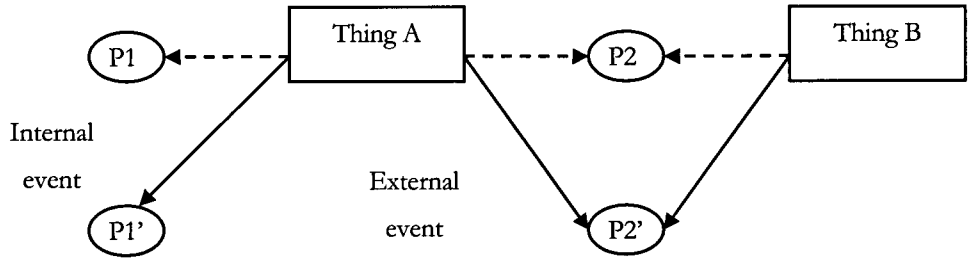
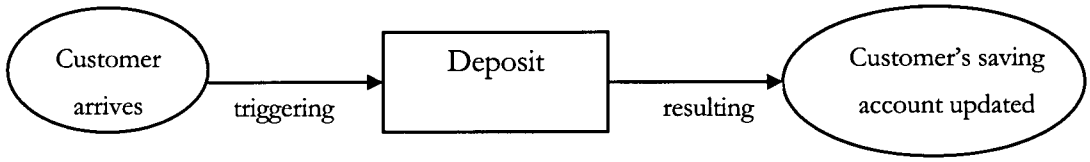


Figure 4-1 illustrates the two types of events. For thing A, p1 is an intrinsic property and p2 is a mutual property with thing B. When an internal event occurs and the value of p1 is changed, the event only involves thing A itself. When an external event occurs and the value of p2 is changed, the event involves both thing A and thing B, and it is the set of values of both p1 and p2.

Also, WFM-event can be differentiated with respect to tasks or processes related with the event. The WFM-event that triggers a task or a process could be named "triggering event". And, the WFM-event that is the result of a task or a process could be named "resulting event". For example, the event that a customer arrives at a bank is a *triggering event* that invokes process of deposit, and causes the state changes of bank clerk, customer himself and his saving account. After the process of deposit, the event that the customer's saving account is increased amount is the *resulting event* of the process. Figure 4-2 illustrates these two types of WFM-events with the example.

Figure 4-2: WFM-Triggering and Resulting Event



A resulting event can also be a triggering event. For example, a resulting event "car assembled" that is the result of the process "assemble car" is a triggering event of the process "deliver car". We need this distinction because we will differentiate them in discussion in later parts for the purpose of an algorithm for our Ontology-based workflow management model.

In BWW Ontology, an actor changes from a stable state to an unstable state at the beginning of an activity; an actor changes from an unstable state to a stable state at the end of an activity. In the sense of precedence of events during an activity, there are two types of WFM-events:

Type one: For a WFM-agent, when a task or a process is completed, it reaches a stable state and will never change until a new event arouses it. The WFM-event at the end of an agent's task or process represents BWWP-stable state of the actor (WFM-agent). Such a WFM-event at the end of task or process is a "final event".

Mapping rule 4: BWWP-stable state of the agent (BWWP-actor) maps to the WFM-event at the end of an agent's task or process.

For example, in the process of assembling cars, the event of "assembling completed" is the BWWP-stable state of the agent manufacturer.

Type two: If the WFM-event triggers an agent to change, it means that the agent is in unstable state and its change starts. This type of WFM-event is between tasks or processes, so it is an unstable state, and the agent will continue to change state until it reaches a stable state at the end of task or process. Therefore, we claim that an agent's all WFM-events, except for its final event at the end of task or process, are BWWP-unstable states.

Mapping rule 5: BWWP-unstable states of the agent (BWWP-actor) map to an agent's all WFM-events except for the final event.

For example, in the process of assembling cars, the event of "installing wheels" is a BWWP-unstable state for agent "manufacturer". This kind of event triggers new tasks. In this example, "installing wheels" triggers tasks as "installing engine". This unstable state will keep changing till the agent reaches a stable state.

4.2.4 BWWP-Transformation

A BWWP-transformation is "a state change from one to another over time". It could be a state change of either actor or non-actor. For non-actor, the transformation is initiated only by an actor. An actor's transformation could involve only itself, or affect another actor or non-actor.

4.2.4.1 WFM-Task

A WFM-task is the elementary activity that forms a logic step in a process. The execution of a task is atomic, which means that once the task starts it will complete and the agent will not change its state unless it is triggered to change again. A WFM-task is a component of WFM-process.

A WFM-task is more than a BWWP-transformation. It is a logical step, rather than a singular state change. At the beginning of a task, agent changes from a stable state to an unstable state. Agent

changes from an unstable state to a stable state when it finishes performing the task. Therefore we claim that the concept of WFM-task expresses the implication of BWWP-a sequence of transformations.

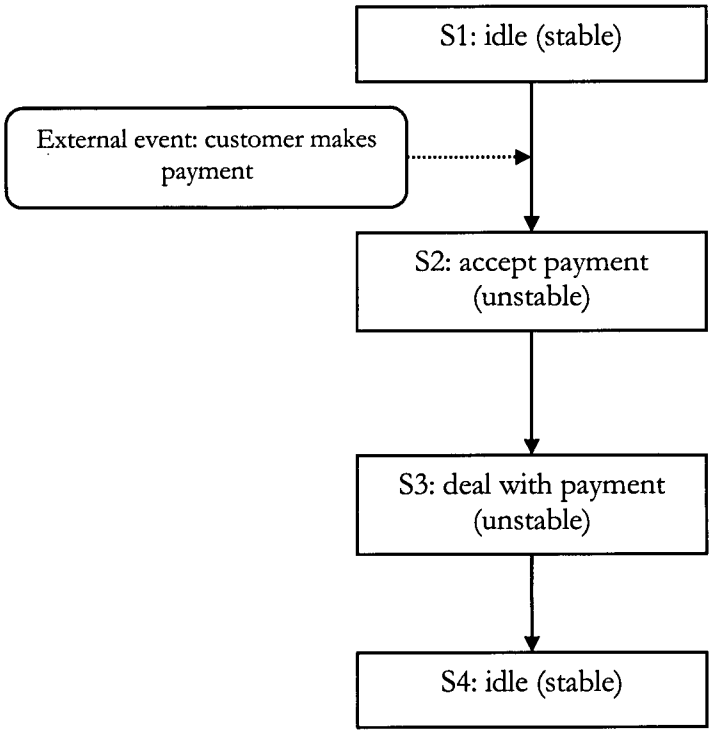
Mapping rule 6: BWWP-a sequence of transformations maps to WFM-task, where the first follows a state-transition from stable to unstable, and the last ends with a stable state.

A WFM-task is necessary for our model because the agent's intermediate states can then be captured during the process. These intermediate states can trigger other agents to change states, or affect resources.

The states before and after the task are both stable. A WFM-task starts with a state transition of the agent who performs the task, from stable to unstable, and ends with a state transition from unstable to stable.

The sequence of transformations is caused by an external event, i.e., its mutual property with others changes value. For example, the agent cashier performs the task of "dealing with customer's payment". Before the task, the cashier's state is stable (s1), the event of "customer making payment" triggers the cashier's state change, when the cashier accepts customer's payment in the form of cash or credit card, he undergoes the state change, from stable state to unstable state (s2). This initiates a task for the cashier. The task is triggered by an external event "customer makes payment", here the cashier's mutual property "accepting payment" with the customer changes value. The cashier continues to change until he reaches a stable state then the task is completed. Figure 4-3 illustrates this task.

Figure 4-3: WFM-task of "cashier accepts customer's payment"



Agents' tasks are usually triggered by the controller's request in our workflow model.

4.2.4.2 WFM-Process

A WFM-process is defined as "a set of one or more tasks which collectively realise a business objective or policy goal, normally within the context of an organisational structure defining functional roles and relationships".

A WFM-process consists of a sequence of tasks, which are performed by one, or more than one, agent. Therefore, a WFM-process reflects all the state changes of multiple agents and resources from the first change caused by an external event to the last change (unstable state to stable state), it reflects the interactions between agents and resources. We call all these changes over time "history of transformations". All participants are involved and interact with each other by

executing tasks. The process is completed when all agents reach stable states and stop changing states.

In this sense, a WFM-process can be regarded as the history of the state changes of participants in BWWP context, which represents the interaction between agents over time.

Mapping rule 7: BWWP-history of transformations maps to WFM-process.

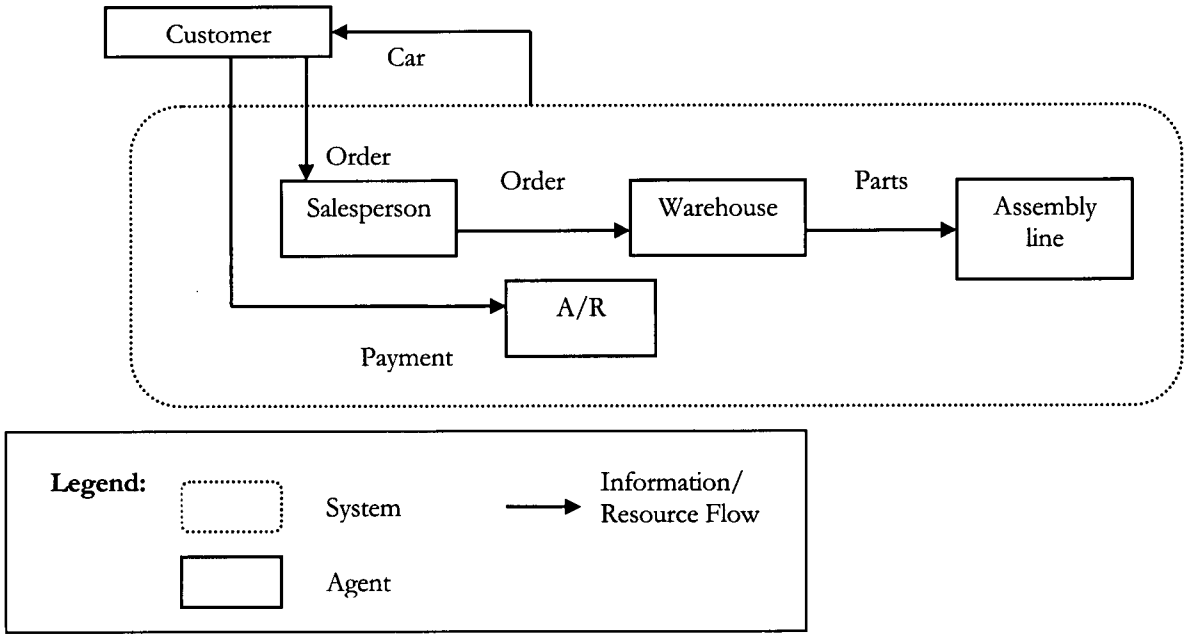
For example, a customer purchases a car from a factory, which assembles the car according to customer's requirements. In this process, the agents are: salesperson, warehouse, assembling line, and account receivable (A/R) staff. During the process, affected resources are: parts, and transaction database. Before the process, all the staff of the factory are idle, i.e., in stable state. The customer is in the environment and triggers the process by interacting with salesperson. The external event of "customer orders a car" is the triggering event that triggers the process. First, the salesperson is changed state. When he accepts the order, he undergoes the state change, from stable state to unstable state. Then the salesperson sends the order to factory and the car is assembled. Two tasks are to be performed in a sequence: "warehouse provides auto parts"; "assembly line assembles all parts into a car". Then another external event that "customer makes payment" occurs. This external event triggers A/R to change state from stable to unstable. Therefore, the third task is: A/R deals with customer's payment. After all these tasks, the process is completed, and all agents and resources reach stable state. The process itself is the history of the state changes of salesperson, warehouse, assembling line, and A/R staff. The history of state changes of all agents and resources is displayed in Table 4-1. In Table 4-1, to make the state changes clear, all unstable states are listed and all stable states are represented by "--".

Salesperson	Warehouse	Assembly line	A/R	parts	Trans. DB.
Accepts order	--	--	--	--	--
Sends order	--	--	--	--	--
--	Provide parts	Got parts	--	--	--
--	--	Assemble parts	--	Be assembled	--
--	--	--	Receive payment		
--	--	--	Deal with payment	--	Be updated
Closes order	--	--	--	--	--

Table 4-1: History of state changes of example "car ordering"

The interactions between agents, and between agents and resources, are illustrated in Figure 4-4.

Figure 4-4: WFM-process of "car ordering"



Given that the additional constructs for our WFM model are WFM-Process state and WFM-Exception, we analyze these two constructs separately for clarification. We will discuss BWWP-law ahead of WFM-exception because we need the concept of "law" in explaining "WFM-exception".

4.2.5 WFM-Process state

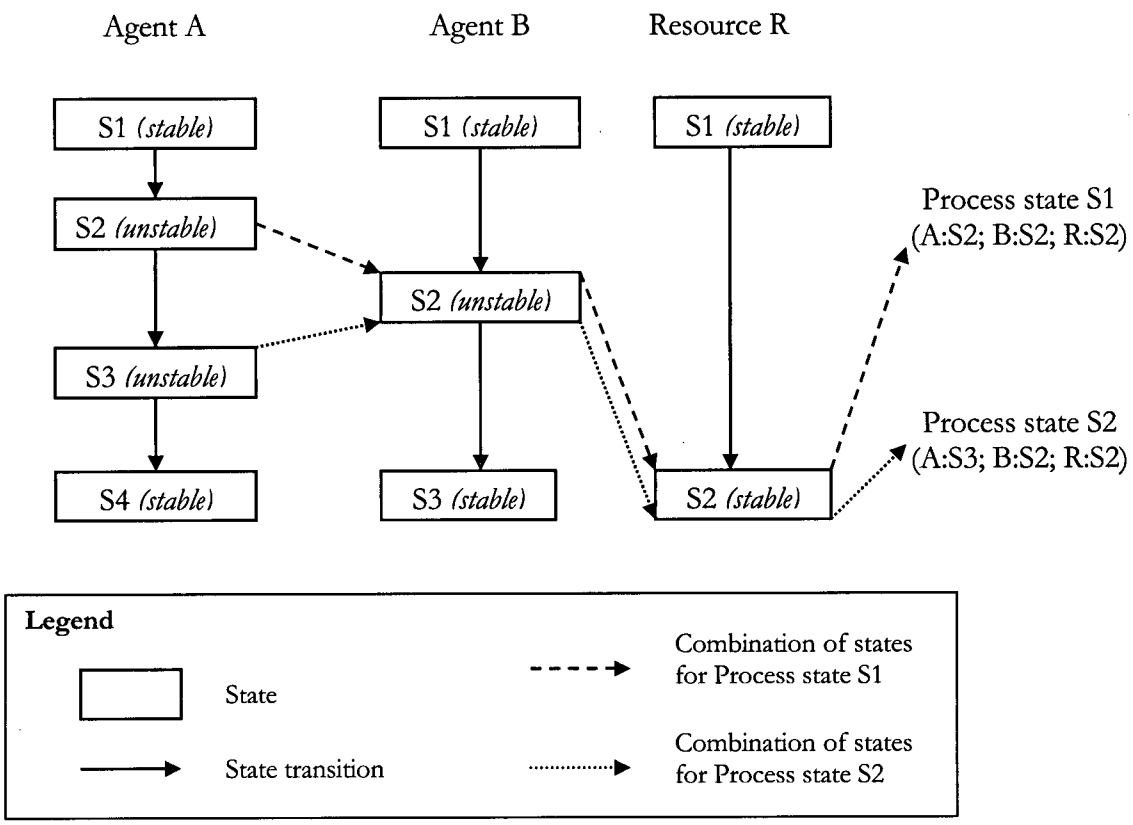
Since WFM model is focused on the dynamic part of the process, the process state needs to be captured and analyzed. A WFM-process state is defined as "the state of a process at a particular point of time" in WfMC terminology.

As we have claimed, a process is the history of transformations of all participants in a process. A process is described in terms of only actions and states of participants. It is all stable states and unstable states of all participants over time during the whole process. State of a process is the state at a particular point of time. Accordingly, we can define process state as the combination of states of all participants at a particular time instant in BWWP context.

Mapping rule 8: BWWP-states of all participants maps to WFM-process state.

Since process state is a combination of all participants' states, there can be various process states. Once one of the participants' state changes, the process state accordingly changes. For two process states, if, and only if, all participants' states in both processes are the same, could we say the two processes are in the same state. Figure 4-5 illustrates WFM-process state. Process state S1(A:S2, B:S2, R:S2) and S2 (A:S3, B:S2, R:S2) are two different states.

Figure 4-5: WFM-Process state



For example, in the process of "deposit", the state of this process is the combination of states of all participants: agent bank clerk A (dealing with deposit), agent bank clerk B (checking customer information), resource customer information database, and resource bank account database. There could be various states for this process. These are two possible states:

- State s1: Clerk A state: waiting for checking customer information; clerk B state: checking customer information; customer information database state: being referenced; bank account database state: not yet involved.

- State s2: clerk A state: receiving result from clerk B; clerk B state: giving result of checking information; customer information database state: not involved (stable); bank account database state: not involved.

The task state can be captured in exactly the same way.

4.2.6 BWWP-Law

BWWP-law is "a relationship between properties". It decides precedence between properties. Property A precedes property B if and only if whenever a thing possesses B, it possesses A. For example, agent bank clerk possesses property "update database" if and only if it possesses property "accept customer's deposit". WFM-business rule expresses this implication. It articulates the policies that guide a business process in its effort to achieve its goals (Keddara, 1999). Business rules are modeled as laws. Basically business rules determine the allowed states of business entities and the allowed actions of agents. They are derived from business objectives and policies. Business rules decide how the tasks are to be done and business rules must be followed during process. Business rules are treated separately because they change frequently over time. Modeling business rules separately makes it easier and more flexible to cope with such changes. We claim that WFM-business rule represents BWWP-law.

Mapping rule 9: BWWP-law maps to WFM-business rule.

In BWWP context, laws are differentiated into two types according to what they restrict: state laws restrict the values of properties of a thing to a subset that is deemed lawful; transformation laws determine the transformations a thing can undergo.

4.2.7 WFM-Exception

A WFM-exception is defined as "an unexpected occurrence which causes error, and needs to be corrected". A WFM-exception prevents task and process from continuing in a pre-arranged way. Actions are needed to fix it so task and process can continue according to business rule.

As we analyzed in previous section, WFM-exception cannot be anticipated and no business rules can be found to fix it. WFM-exception is actually the cause of recovery, it triggers agent to start changing state to recover to its lawful state. Obviously, it is the starting point of state change, which expresses the similar implication to WFM-event, which we mapped to BWWP-state of actor. BWWP-lawful state is a state that is defined by state laws, and unlawful state is a state that is not defined by state laws. Since a WFM-exception is a particular occurrence for which is not defined by business rules which we mapped to BWWP-law, we claim that a WFM-exception represents BWWP-unlawful state of the agent.

Mapping rule 10: BWWP-unlawful state maps to WFM-exception.

Accordingly, WFM-result represents BWWP-lawful state of the agent.

Mapping rule 11: BWWP-lawful state maps to WFM-result.

A WFM-exception cannot be anticipated and is beyond the business rules, so it can only be solved by human intervention rather than by WFMS. In the example of "review the application of credit card", an exception could be: credit information database temporarily down when the applicant's credit information is needed. This is not anticipated by business rules so that it is an unlawful state of the resource "credit information database". In this situation, WFMS can do nothing with it.

We have eleven mapping rules of BWWP constructs and WFM constructs. We discussed them based on previous research work on mapping between process modeling (PM) and BWWP. For comparison between these two works, we have a figure to summarize them at the end of this chapter.

4.3 Controller and Controlling Mechanism

First, we discuss the role and functions of the workflow management system in a workflow.

4.3.1 Controller

We regard the workflow management system as a "Controller" in our model. The concept of the controller was introduced in previous work (Hui, 1997; Guo, 2002) and it will be extended and formalized in this thesis. It is a special actor who interacts with all agents, monitors and controls the whole process and all the individual tasks. The functions of the controller are:

- Assign tasks to agents.
- Keep track of the progress of tasks and processes.
- Ensure that business rules be followed in an automated workflow environment.
- Handle exceptions.

The controller can be considered as a "Super Actor" who acts different from other actors. It follows business rules that are set in advance and it acts above all other agents and resources.

As mentioned in chapter two, the "Process Knowledge" stores all results that are defined by business rules. Since business rules are mapped to law in BWW Ontology, the "Process Knowledge" actually stores all lawful states of agents and resources and the following actions. For example, there are two lawful states for agent "clerk" for performing task "review application complete or not", one is "receiving complete application" and the following action is "review applicant qualified or not"; another lawful state is "receiving incomplete application" and the following action is "return to applicant and attach a note about the missing document".

We care about the interactions between the controller and agents rather than the state changes inside agents. Therefore, those state changes inside agents are not included in the database. Since an agent's state is a vector of values of properties of the agent at a particular moment, we suggest the database stores all lawful states in the form of the combinations of values of mutual properties of the agents. All states of the controller are decided by interactions between agents and Controller. Therefore, each of agents' mutual properties would be one of the controller's mutual properties. To be readable for computer systems, the information in the database appear as numeric values. For example, a state of agent A, $S_A(1)$, is the combination of values of its three properties, say (P1=5, P2=10, P3=30). We represent it as $S_A(1)$: (P1=5, P2=10, P3=30). Similarly, $S_A(2)$: (P1=5, P2=10, P3=50). When an external event occurs to agent A, it changes from $S_A(1)$ to $S_A(2)$. This transformation is A's lawful transformation according to the transformation law L1: $S_A(1) \rightarrow S_A(2)$. Controller interacts with agent A when A is in state $S_A(1)$ by their mutual property P3. Before the interaction the controller is in a state $S_C(1)$: (P3=30, P4=5). P4 is another property of the controller. After the interaction, A changes to $S_A(2)$ according to its transformation law, and the controller changes from state $S_C(1)$: (P3=30, P4=5) to state $S_C(2)$: (P3=50, P4=5).

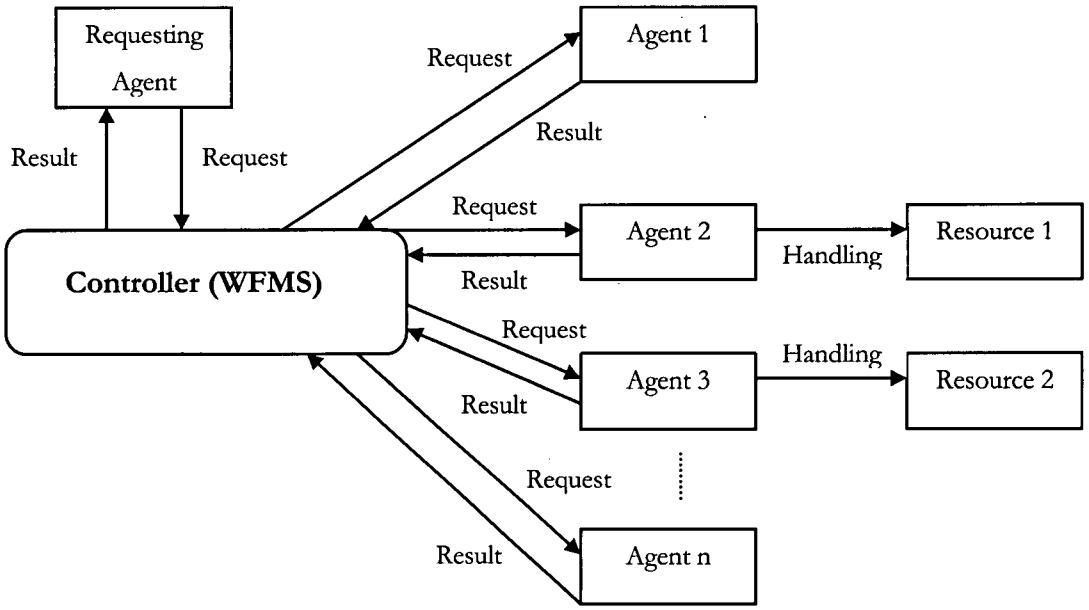
In our model, the process state is represented by states of all agents and the controller at a particular moment. For agent A_1, there are a certain number of mutual properties and we store all combinations of values of its mutual properties in the database, i.e., all lawful states that are known to others. We represent a lawful state S_A of agent A_1 in the form of the combination of values of all mutual properties: $(A_1^1, \ldots A_{n1}^1)$, (n is the number of agent A_1's mutual properties), at a particular moment and this state is a lawful state that is stored in the database. For agent A_k, its state at a particular moment is represented as: $(A_1^k, \ldots A_{mk}^k)$, (m is the number of agent A_k's mutual properties). Similarly, the controller's state is represented as: $(C_1, \ldots C_r)$, (r is the number of the controller's properties). If there are agents $A_1, \ldots A_k$, (k is the number of agents), and the controller in the process, the process state is: $(A_1^1, \ldots A_{n1}^1, \ldots A_1^k, \ldots A_{mk}^k, C_1, \ldots C_r)$.

The "Process Knowledge" is utilized by the controller during process. The controller receives a triggering request from agent, makes decisions about performing tasks, sends requests to agents, receives responses of agents' performing tasks, and solves problems if any occurs. In addition to the "Process Knowledge", the controller needs to know the current states of agents at a particular moment during the process. The controller has these two sets of information to monitor and control the process according to business rules. At the moment of inactivating process, agent A changes state when an external event occurs to it and it reports to the controller by changing the mutual property (P1) of the controller and itself. Another property of the controller (P2) changes during this interaction. P2 is mutual property of the controller and agent B. After that, the controller decides whether A's new state is lawful by retrieving information from the "Process Knowledge". If the controller decides the state is lawful, the changing of P2 affects B to change state. The controller sends request for performing task T. B sends response to the controller after it performs the task T.

The controller takes care of all requests and the responses to the requests. Given agents' states at a particular moment after receiving responses from agents, the controller identifies the process state.

The controller retrieves anticipated results from the "Process Knowledge", and compares results (after-state) received from agents with the anticipated results to decide the following actions. If the controller cannot find the results, it decides they are actually exceptions and it handles exceptions. From this perspective, the controller is the "communication centre". Figure 4-6 illustrates this interaction.

Figure 4-6: Interaction between Controller and agents



A typical workflow will work in the following steps:

1. Agent 1 is activated by an external event. The process is triggered.

2. The activated agent (Agent 1) interacts with the controller by changing their mutual property. If this transformation is unlawful, the process is halted. The controller decides whether the transformation is lawful by retrieving information from "Process Knowledge".

3. The controller is activated by the interaction with Agent 1. The controller starts changing another mutual property, thus affects Agent 2. The controller sends request to Agent 2 by changing their mutual property. The controller changes state according to business rules.

4. Agent 2 changes state and then interacts with the controller to report its state change.

5. The controller is activated again. If Agent 2 changes state to an unlawful state, the controller handles the exception; otherwise, the controller continues changing according to normal process route, and affects Agent 3 to continue the process.

6. Repeat step 4 and 5 until the controller reaches a stable state and stops sending requests. The process is completed.

In the rest of this thesis, we follow our initial definitions of "result" and "exception".

4.3.2 Controlling Mechanism

From an ontological perspective, the controller is a substantial thing and it is an actor that can change not only its own state but other things' states as well. The controller is a "super actor" who interacts with all agents, monitors and controls the whole process and all individual tasks. It assigns tasks to agents, keeps track of tasks and processes, ensures business rules are followed in automated workflow environment, and handles exceptions. The controller's controlling mechanism reflects business rules that decide the interactions between agents and the controller.

When the controller knows the current state of an agent, it knows which action will be taken and what lawful transition the agent will undergo.

A typical set of interactions between the controller and two agents (A and B) will explain the procedure from an ontological perspective.

1. A state of agent A is the combination of values of its three properties, $S_A(1)$: (P1=5, P2=10, P3=30). Agent A interacts with the controller by changing their mutual property P3. A changes from $S_A(1)$: (P1=5, P2=10, P3=30) to $S_A(2)$: (P1=5, P2=10, P3=50).

2. The controller changes from a stable state $S_c(1)$: (P3=30, P4=5, P5=12) to an unstable state $S_c(2)$: (P3=50, P4=8, P5=12). During the interaction, A sends information about its state $S_A(2)$ to the controller. The controller retrieves information from the "Process Knowledge" and decides that $S_A(2)$ is a lawful state since it can be found following $S_A(1)$ in the "Process Knowledge". P4 "communicating between the controller and B" is a mutual property of the controller and agent B.

3. B is in a lawful state $S_B(1)$: (P4=5, P6=20, P7=40). The controller refers to the "Process Knowledge" and determines that B will perform task T. The controller sends another request to B to ask it to perform T (change B's state). This interaction changes the value of P4. After the interaction B will change state from $S_B(1)$ to $S_B(2)$: (P4=9, P6=20, P7=40). The state change of B is according to B's transformation law L2: $S_B(1) \rightarrow S_B(2)$.

4. B performs T and changes another property P6 "performing T". B reaches to state $S_B(3)$: (P4=9, P6=25, P7=40). B sends response to Controller to report its state after performing T. This interaction changes P4 again. B reaches a stable state $S_B(5)$: (P4=11, P6=25, P7=40).

5. The controller reaches a stable state $S_c(2)$: (P3=50, P4=11, P5=12) when it receives B's response and decides B is in a lawful state. If Controller cannot find B's state after performing T in the database, it decides that is an exception and it handles exception.

For example, in the process "dealing with deposits", there are the controller and two agents: "Clerk A" to perform task "deposit into customer's account", and "Clerk B" to perform task "check customer information". The external event "customer arrives" occurs to affect Clerk A and initiate the process. Clerk A changes from a stable state to an unstable state "receiving customer's request of deposit". Clerk A interacts with the controller by changing the mutual property "communicating" of Clerk A and the controller. The controller refers to "Process Knowledge" and decides that Clerk A is in a lawful state. It then changes state to decide the following action will be "check customer information" and Clerk B will perform the task because Clerk B is going to change to a lawful state of "checking customer information". The controller identifies this property of Clerk B from the state representation in the "Process Knowledge". It is represented as: (Clerk B, check customer information, value). The controller sends a request to Clerk B by changing their mutual property "communicating for a task of checking the customer information". Clerk B performs the task and changes state. Clerk B sends response to the controller about the result of checking customer information by changing their mutual property "communicating for a result of task of checking the customer information". The controller decides Clerk B is in a lawful state and decides the following action "accept customer's deposit" will be performed by Clerk A.

The controller makes this decision by retrieving information representation from the "Process Knowledge". It is represented as: (Clerk A, accept deposit, value). The controller sends request to Clerk A by changing their mutual property "communicating for a task of accepting deposit". Clerk A performs task "accept deposit" and sends response to the controller by changing their mutual property "communicating for a result of task of accepting deposit". The controller decides it is a lawful state by retrieving information from "Process Knowledge". Both Clerk A and Clerk B are in stable states at the moment. Hence, the controller is also in stable state. The process is completed.

However, the controller does not necessarily know the business rules within agents. These business rules are concerned with how agents act themselves when they interact with others. Our model does not care about the internal attributes inside agents that are unknown to others, and those internal attributes do not appear in the "Process Knowledge". For example, agent A and agent B can perform the same task T and provide the same results, but they may act differently depending on their skills and capabilities. This difference is not taken into consideration in our model.

The four functions of the controller are analyzed in further details in the following parts.

4.3.2.1 Assigning Tasks

One of the main functions of WFMS is to assign tasks to the appropriate agents. This matching can be very complicated when taking into consideration many factors such as the quantity of agents, the qualities of each agent for each task, the cooperation between agents, and the sharing of resources. We will show that the assignment can be made simple and clear from the perspective of BWW Ontology.

Basically, tasks are the state changes of agents and resources, and processes reflect the history of state changes of all participants. As we suggested in previous section, the controller can retrieve all lawful states of agents and resources and the following actions of their lawful states from the "Process Knowledge". Therefore, the controller decides which task to perform based on the information from the "Process Knowledge". Given all lawful states of agents and resources, the controller can decide which agent can perform the necessary task. Basically, this decision is made based on business rules.

With our ontological analysis on workflow system, the information requirements for automation are derived clearly and easily. The state variable of an agent can be represented as: (agent, attribute, value). Since an agent has internal attributes and interface attributes, the agent's state is represented as: (agent, $A_1,\ldots A_m$, $B_1,\ldots B_n$) (A represents the interface attribute and m is the number of interface attributes; B represents the internal attribute and n is the number of internal attributes). When an agent undergoes a state change, the value of the interface attribute A_k is changed from A_k to A_k'. The state change of can be represented as <S1, S2>, S1 = $<A_1,\ldots A_k,\ldots A_m, B_1,\ldots B_n>$, and S2 = $<A_1,\ldots A_k',\ldots A_m, B_1,\ldots B_n$. The controller only knows the interface attributes of agents and it knows the current states of agents. Therefore, the controller needs to know the change of interface attributes only, which is represented as <agent, A_k'>. The controller only needs to know the new value of the changed attribute because it already knows the previous value. For the purpose of clarity and brevity, it is sufficient to describe requests and responses by those interface attributes whose values are changed. All requests sent from the controller, exception and non-exception can be represented in this way. For example, agent "cashier" undergoes state change when it performs the task "issue cheque". "Issuing cheque" is an interface attribute of agent "cashier" that can generate a filled cheque after the state change. The state change of agent "cashier" is represented as

<S1, S2>. S1 is the state before the task "issue cheque", and S2 is the state after the task. S1 can be represented as: (cashier, request to issue cheque, to whom and amount). And S2 can be represented as: (cashier, issued cheque, to supplier and amount $500).

This could also be a way to represent of the laws. The process knowledge available to the controller includes every lawful state, whether it is stable or not, and, if unstable, the next state of the controller. Changing to the next state can be described in terms of only the state variable that will be affected. In particular, these could be mutual state variable with other agents. In such cases, the law can be described as an external event to an agent,: <agent, A_k=v>.

Accordingly, we claim that the controller assigns a task to an agent when the agent has to perform the task by changing state from (agent, attribute, value1) to (agent, attribute, value2) according to business rules. Value 1 and value2 are both lawful values of attribute of the agent according to "Process Knowledge". For example, the controller sends a request "issue cheque" to an agent "cashier". The request can be represented as: (cashier, issuing cheque, to supplier and amount $500).

Assigning task: Assign a task to an agent if the agent has to perform the task by changing from a lawful state (agent, attribute, value1) to another lawful state (agent, attribute, value1').

In our model, if an agent needs to contact another agent during a task, it interacts with another agent and finishes the task without contacting the controller. The interactions between agents during tasks can be either in or out of the control domain of the controller. The control domain of the controller is decided by the system owner. In the control domain, interactions between two agents are represented by the interaction between one agent and the controller, and the interaction

between the controller and another agent. The controller controls all the requests and responses in a workflow. Sometimes an agent needs to contact another agent for the purpose of finishing a task. Such interaction can be either known to the controller or not. It is decided by the system owner.

4.3.2.2 Ordering

The controller sets the order of tasks in a process. The order of tasks depends on business rules and reflects them. Business rules actually decide the mutual properties of agents and the controller. The controller retrieves information from "Process Knowledge" to identify when an agent is in a particular state, i.e., its combination of values of properties, and to then decide what action needs to be taken and which agent will be affected to change state.

Order of tasks is decided by business rules.

When an agent performs a task by changing the value of its interface attribute, another agent will perform a new task by changing the value of its interface attribute following the previous task. If the interaction is in the control domain, the controller decides the order of tasks by retrieving information from "Process Knowledge".

When agent A1 changes state from <A1, a1=v1> to <A1, a1=v1'>, it interacts with the controller. The controller changes state and retrieve information from "Process Knowledge" to identify the following action. If any error occurs, the controller handles the error. If the controller changes to a stable state, it does not send any request. If it changes to an unstable state, it identifies the following action will be <A2, a2=v2>. The controller sends request to agent A2 and A2 changes state from <A2, a2=v2> to <A2: a2=v2'>.

The order can be represented as:

(agent 1, a1=v1) -> (agent 1, a1=v1');

(agent 2, a2=v2) -> (agent 2, a2=v2').

The interactions between the controller and agents describe the mutual properties and are decided by business rules. The controller makes decisions according to the information from "Process Knowledge" that is established based on business rules. The order of tasks and events in a process expresses their logical relationship. Here we attempt to capture the logical relationships in following three categories:

- WFM-event → WFM-task
- WFM-task → WFM-event
- WFM-task → WFM-task

An event can initiate one or more tasks by changing the mutual properties between environment and actor, or between actors, which occurs at the triggering of process or during the process. This kind of event is what we called a "triggering event" in previous section. Sometimes there is nothing but only one event in a process even though that event will initiate more actions. After an event occurs, another event occurs to stop the process and either or not to initiate a new process, so there is only one event in the first process. For example, a customer orders a car, then the order is placed and event "customer makes order" initiates the "fulfil car ordering" process; immediately after that the customer cancels the order, then a new event "order is cancelled" occurs to stop the previous process.

A task, which is a set of transformations of agent and the affected resources, always results in an event, which we called "resulting event". A task also initiates one or more other tasks in a process by changing mutual properties between agents and resources.

The controller sets order according to business rules. When the controller receives a triggering request, it retrieves information from "Process Knowledge" to decide what action will be needed and which agents will be affected. Then it sends requests to the agents. In the example of "car ordering", when the controller receives a triggering request from salesperson, it sends requests in a sequence as follows:

- Request to warehouse: "provide auto parts".
- Request to assembling line: "assemble parts into a car when parts are available from warehouse".
- Request to A/R: "deal with payment when car is available and payment arrives".

The three requests are in an order based on business rules. Business rules are represented by law in Ontological context, which decide precedence of properties. Business rules are set in advance for a business process. The sequence of the controller's requests are based on business rule. The controller knows the lawful states of agents and resources, and decides the three requests and their order based on information retrieved from "Process Knowledge". The process is enacted by the value change of mutual properties between environment and agents, and it continues by the value change of mutual properties between the controller and agents.

4.3.2.3 Tracking

The controller has the "Process Knowledge" as we suggested in previous section. It tracks all state changes and states of all agents and resources at every particular moment by retrieving anticipated states from the "Process Knowledge" and receiving responses from agents. Agents send responses to the controller when they finish perform tasks.

Track tasks and processes by retrieving information from "Process Knowledge" and receiving responses from agents.

The process state is actually the combination of states of all agents and resources. The controller records the process state based on all participants' states.

4.3.2.4 Error Handling

We suggest the controller handles errors, including both exceptions and design errors. The controller can report the error state that causes the problems to the system owner for human intervention.

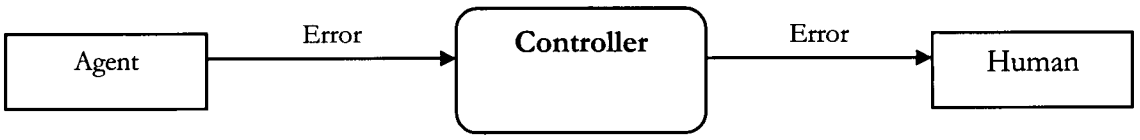
When the controller receives responses from agents, it refers to the "Process Knowledge" to identify the next action. Errors occur when the controller cannot determine the next action. The situations may arise when either of the following two happens:

- The controller cannot find the state
- The controller identifies more than one possible action

The first situation happens when the analysis is incomplete, while the second situation happens when the analysis result is inconsistent. Errors are modeled as the outcome of performing tasks that does not match lawful states allowed by business rules. An error can be represented as: (agent, attribute, value), and the value is unlawful. For example, value should be no less than 0 according to business rules. A negative number of the value will be identified as unlawful. Therefore, the state is an error.

When the controller cannot determine the next action, it can do nothing with it but report to system owner. The process is suspended or failed. This is a business rule which is only known to the controller. It has nothing to do with agents. It is not a business rule that decides lawful states of agents and resources for the "Process Knowledge". Figure 4-7 illustrates the mechanism.

Figure 4-7: Controller handles error



For example, the error "system down" in process "reviewing credit card application" is an exception. The controller can do nothing with it. The error will be reported to system owner, and technical support staff will be contacted to fix the system. The process is suspended because of the error and it will not continue until system is fixed.

Design errors are usually caused by changing definition of business process while not updating business rules, changing organization structure, or insufficient analysis. The controller reports the errors to the system owner for repairing. After that, business rules will be modified and "Process

Knowledge" will be corrected for future use. For example, in the process "mortgage application review", agent "bank clerk" performs task "review applicant's income". The agent is in a lawful state "calculating applicant's annual income". The agent "bank clerk" calculates the applicant's total income as salary plus self-employment income according to a business rule "income is the total of salary and self-employment". The agent "bank clerk" finds an applicant has salary and commissions as well. Since the commission is not specified in the relevant business rule, the bank clerk is in an unlawful state. A design error occurs and the process is halted. The bank clerk sends response to the controller. The controller reports the design error to the system owner. The business rule will be modified as "income is the total of salary, self-employment and commissions". The "Process Knowledge" will be updated. The process will be initiated again and agent will perform task according to the modified business rule.

4.3.3 Comparison of ADOME-WfMS and our work

The Advanced Object Modeling Environment (ADOME) is proposed by Dickson Chiu (2000) in his PhD dissertation. We compare Chiu's work with ours to identify the difference.

Based on OODBMS, Chiu's work proposed an object-oriented model (ADOME) with an exception centric view. Based on its exception definition from implementational aspect, it discusses exception handling mechanism in great details in an implementational level. For example, it proposes a web-based user interface to facilitate users to access the WFMS; it describes how the ADOME-WFMS automatically resolves expected exceptions. In contrast, our ontological approach does not discuss implementation issues. Based on BWW Ontology, our model attempts to provide a high-level representation of business processes in reality from an organizational view. Chiu's work does not cover the representation issues. The following table of comparison identifies

some similarities and differences. Our ontological approach actually provides support for Chiu's work.

	OWFM	**ADOME**
Theoretical foundation	BWW Ontology	OODBMS
Task is executed by	Agent	Problem Solving Agent
Task is assigned by	Controller	Match Maker
The capability of executing task is defined by	Agent's lawful states	PSA role/token
Exception is handled by	Controller	Exception Manager

Table 4-2: Comparison of OWFM and ADOME

ADOME is based on OODBMS (Chiu, 2000. P7), which is an object-oriented technology for implementation. Our approach is based on BWW Ontology, a theory that deals with representation of real world. For the function of executing task, our approach suggests agent to perform task. Agent is an active thing that can change state of itself and other things. In contrast, ADOME proposed a "Problem Solving Agent" (PSA), a hardware/software system or a human being with an ability to execute a finite set of tasks in an application domain (Chiu, 2000. P1). Therefore, the "agent" in our approach and the PSA in ADOME are rooted in different aspects while providing same functions. For the function of assigning task, our approach suggests the Controller assign tasks according to the lawful state representation information from "Process Knowledge" based on business rules. ADOME proposed a "match maker" to assign PSA objects to workflow instances based on role/token matching. A PSA role/token embodies certain capabilities of a PSA to execute certain functions/procedures/tasks. For exception handling, our approach suggests the controller reports exceptions/design errors to the system owner for human intervention since exceptions/design errors are beyond the "Process Knowledge". ADOME

proposed a taxonomy of exception handlers for the automatic handling of expected exceptions and user assisted handling of unexpected exceptions. It also discussed the reusability of handlers. Therefore, Chiu's work provides implementation guidance, while our approach does not provide implementation guidance but provides a business modeling view.

4.4 Ontology-based Workflow Model (OWFM)

The Ontology-based workflow model is focused on the interactions between the controller and agents. The tasks and events are the main concerns in this model.

4.4.1 Participants

Participants in a workflow are differentiated into two categories: agents and resources. Mapping rules 1 and 2 defined these in chapter three.

The participants are differentiated by being observed their response to the event occurred to them.

The questions that arise are:

- Does this participant change its state from stable to stable, or from stable to unstable?

1) From stable to stable

If its state change is from a stable state to another stable state, the participant's state change is not able to affect other things in the workflow domain. Therefore, according to mapping rule 2, we consider it as a resource referenced or used by agent during tasks or process.

2) From stable to unstable

If the state change is from a stable state to an unstable state, the participant will keep changing states and will change the state of other participants in the workflow domain. According to mapping rule 1, we say this participant is an agent.

In a business process, there is usually more than one agents and resources. For clarity we index them as Ai (i=1 to n, n is the number of agents) and RESi (i=1 to m, m is the number of resources).

4.4.2 Triggering Request

The process is always triggered by an external event from the environment. An agent interacts with the environment and it is caused to change state, and then its state change arouses state changes of other things, the whole sequence of state changes and interactions is the process. Here we classify this kind of triggering request as "*external triggering event*" (ETR). For example, the event that "customer makes payment" is an external trigger that is from the environment and initiates the process "order processing", the cashier's state is changed because of this external triggering request. We index the external triggering event as ETRi (i=1 to k, k is the number of external triggering requests).

When the agent is affected by the external triggering event and changes state, it sends a request to the controller by changing their mutual property. We define it as "*internal request to controller*" (ITR to C). The controller changes state. If it changes to a stable state, it does nothing. If it changes to an unstable state, it refers to the "Process Knowledge" to identify the next action. If no error occurs, the controller sends out a request to another agent for the next action. We define the request sent from the controller as "*internal request from controller*" (ITR from C). Agent sends "*response*" to the controller when it finishes performing tasks or when the controller asks about agent's state. The response is actually the state of the agent and can be represented as (agent, attribute, value). When

receiving the response from agents, the controller changes state and refers to the "Process Knowledge" to identify the next action. The controller determines the next action and then send out other request. For example, agent "officer" sends to the controller the response "applicant's credit record is checked and the applicant is qualified for loan application", which is an internal request to controller that initiates the next task "processing loan".

We index them as (ITR to C)$_i$ (i=1 to p, p is the number of internal requests to controller); (ITR from C)$_i$ (i=1 to k, k is the number of internal requests from controller); and RESP$_i$ (i=1 to j, j is the number of responses).

Note that sometimes an external event occurs during the process while not triggering a new process, instead it triggers actions during a process. In this kind of situation, it should be handled as an internal request. For example, in the process of "car order fulfillment", the customer makes payment. The external event "customer makes payment" occurs to the agent "cashier" during the process while it does not trigger a new process. The agent "cashier" sends the internal request to the controller. The controller changes state either to a stable state or to an unstable state. When it changes to a stable state, it does not identify the next action. The information might be needed for future action. When the car is ready, the controller sends internal request to another agent "order fulfilling clerk" to ask it to deliver the car if it identifies the full payment. When the controller changes to an unstable state, it refers to the "Process Knowledge" to identify the next action and sends another internal request. The state change of the controller is decided by business rules.

4.4.3 Task

Task is a logic step in a process and it is a sequence of transformations of agent. Tasks are usually assigned by the controller to appropriate agents. A task is represented as (agent, attribute, value) -> (agent, attribute, value').

The external triggering event always triggers one or more agents to change states, i.e., to perform the first task. This first task is the only task that is not assigned by the controller. We classify it as "*start task*" (ST). For example, the external triggering event "customer makes payment" initiates the start task "A/R processing payment".

After that, the controller assigns tasks to agents. These tasks are classified as "*interval task*" (IT). For example, the task "assembling auto parts into a car" is an interval task that is initiated by interval triggering request "auto parts available" and is assigned by the controller to agent "assembly line". Agents send responses to the controller to report their performing tasks. The responses initiate the controller to send out internal requests for following tasks. The last task in a process is different from these interval tasks in the way that it is not followed by any more tasks. The controller evaluates the task and decides that it is the last one in the process and there is no more necessary task. When all agents are in stable states, the controller is also in stable state and does not send out requests. Therefore, we classify the last task as "*end task*" (ET). For example, the task "sending credit card" is the end task in process "credit card application reviewing" because it does not initiate any other tasks and the controller stops sending out requests.

Sometimes there are more than one agent that are affected by external triggering event and there would be more than one start tasks. The same situation fits end tasks. Usually there are more than

one interval tasks. Therefore, we index these tasks as STi (i=1 to r, r is the number of start tasks) and ITi (i=1 to s, s is the number of interval tasks) and ETi (i=1 to t, t is the number of end tasks).

Note that sometimes a certain task is performed more than once and it should not be indexed repeatedly.

For each agent, there is a list of start tasks, and a list of interval tasks and a list of end tasks as well. Therefore, we have the following pairs:

Ai ↔ STi; **Ai ↔ ITi;** **Ai ↔ ETi.**

And for each task performed by the agent, there is a list of affected resources. Therefore, we have:

Ai ↔ STi ↔ RESi; **Ai ↔ ITi ↔ RESi;** **Ai ↔ ETi ↔ RESi.**

For each internal request sent to the controller, the controller refers to the "Process Knowledge" and decides what task(s) will be done next and which agent(s) will perform the task(s). Therefore, each internal request to the controller initiates one or more interval tasks. The responses (both non-exceptions and exceptions) sent by agents to the controller about their performing tasks could trigger the controller send out one or more internal requests. The start task could generate one or more internal requests to the controller. So we have the following pairs:

(ITR to C)i ↔ ITi.

RESPi ↔ (ITR from C)i.

STi ↔ (ITR to C)i.

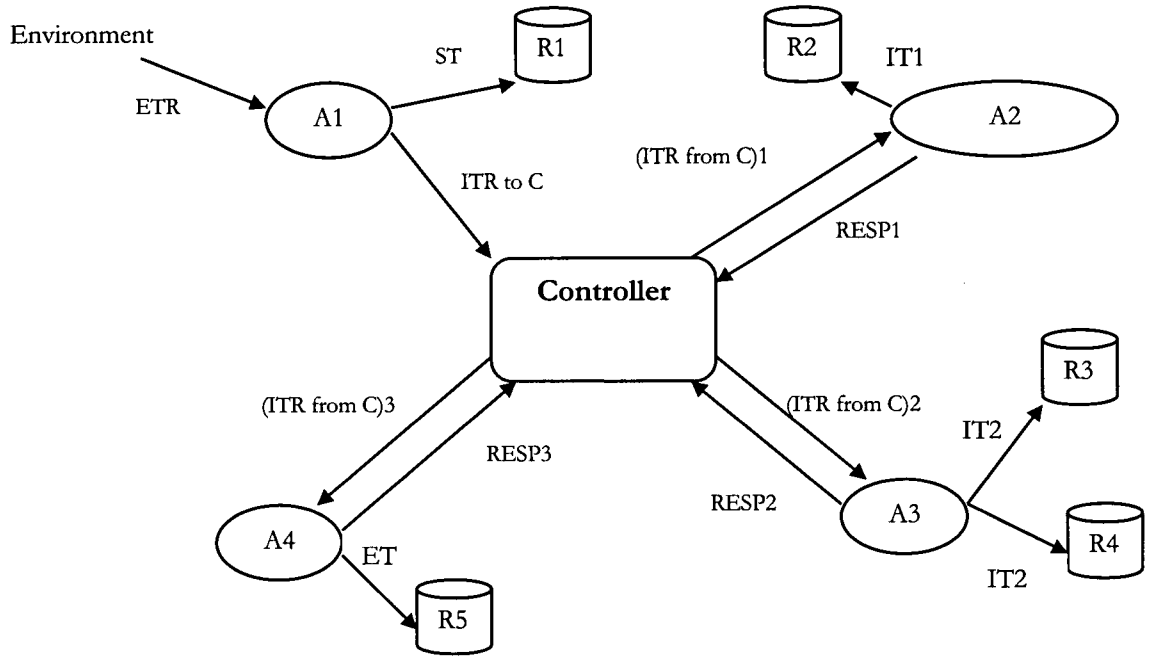
Finally, each task is performed by only one agent.

Ti ↔ Ai.

All internal requests from the controller are the events in the controller which change the mutual properties between the controller and agents. All internal requests to the controller and responses are the events in the agents which change the mutual properties between the controller and agents. All tasks are actually the state changes of the agents. We have the above pairings for the purpose of an algorithm on how to use the ontological workflow model in business practice. The algorithm will be presented in Chapter Five.

Figure 4-8 shows the Ontology-based workflow model.

Figure 4-8: Ontology-based Workflow Model



In Figure 4-11, a workflow system of the controller, four agents and five resources is presented to illustrate the model. The process is initiated by the interaction of environment and agent A1. Agent A1 performs start task and affects resource R1. Agent A2 performs interval task IT1 and affects resource R2. Agent A3 performs interval task IT2 and affects resources R3 and R4. Agent A4 performs end task and affects resource R5. All results sent to the controller are compared with those from "Process Knowledge" and initiate following actions. The whole process in full details is shown in Appendix B.

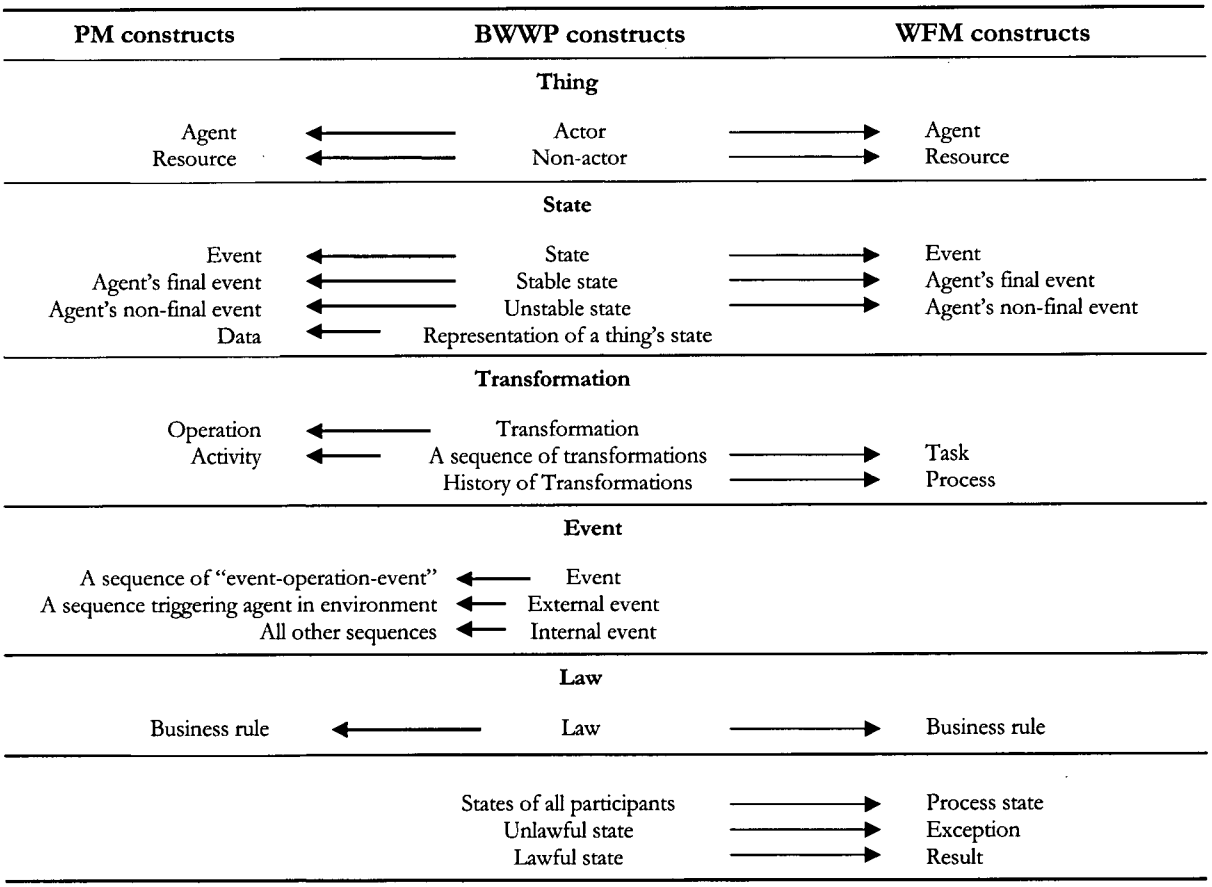
4.5 Summary

This chapter presented an Ontology-based Workflow Model. First we outlined a set of BWWP-WFM construct mapping rules. Then we showed the Ontology-based Workflow Model, which incorporates the controlling mechanism of the controller with workflow generic constructs. They are summarized in the following parts.

4.5.1 Summary of BWWP-WFM construct mapping rules

The BWWP-WFM construct mapping rules are summarized in Figure 4-9. This mapping summary will be used in Chapter Five for algorithm. And we also put the mapping between PM and BWWP for comparing the two research works.

Figure 4-9: BWWP-WFM construct mapping rules



4.5.2 Summary of Ontology-based Workflow Model

The Ontology-based Workflow Model incorporates the controlling mechanism of the controller and workflow generic constructs.

The controlling mechanism consists of four functions of WFMS, what we call "the controller". It is the essential part of the Ontology-based workflow management model. It provides the foundation of an algorithm on how to employ the model. The algorithm will be presented in Chapter Five.

The controlling mechanism is summarized as follows:

- The controller decides what new task is to be performed according to "Process Knowledge" decided by business rules and assigns a task to an agent if the agent is going to perform the task by changing from a lawful state (agent, attribute, value1) to another lawful state (agent, attribute, value1').

- The controller sets order of tasks according to the business rules that decide the mutual properties of agents, resources and the controller, by referring to "Process Knowledge". When an agent performs a task by changing the value of its interface attribute, another agent will perform a new task by changing the value of its interface attribute following the previous task.

- The controller tracks tasks and process state by retrieving information from "Process Knowledge" and receiving responses from agents about their performing tasks.

- The controller reports exceptions and design errors to system owner since exceptions and design errors are beyond the business rules.

In our Ontology-based Workflow Model, all agents (A), resources (RES), internal requests to the controller (ITR to C), internal requests from the controller (ITR from C), external triggering events (ETR), responses from agents (RESP), start tasks (ST), interval tasks (IT), and end tasks (ET) are

indexed and paired for the purpose of proposing a methodology on how to use our approach in business practices. They do not necessarily represent a one to one relationship. In stead, they are often one to many or many to many relationship in reality. They are summarized as:

- Ai ↔ STi ↔ RESi; Ai ↔ ITi ↔ RESi; Ai ↔ ETi ↔ RESi.
- (ITR to C)i ↔ ITi.
- RESPi ↔ (ITR from C)i.
- STi ↔ (ITR to C)i.
- Ti ↔ Ai.

4.5.3 Advantages of Ontology-based WFM model

The advantages of Ontology-based workflow management model will be shown in this section. The advantages form a good foundation and provide the requirements for a WFMS. The model is simple, general and based on formal theory. It provides the advantages as follows:

First, by extending Ontology-based process model, Ontology-based workflow management model is built based on BWW Ontology, which enables it to provide an abstract representation of the real world, thus reflects a "business view" rather than implementation view.

Secondly, with the introduction of the controller whose controlling mechanism reflects business rules, Ontology-based workflow management model has the ability to separate what needs to be controlled and what does not. The controller is actually the workflow engine while we model it from a business view rather than implementation view. Another advantage of the controller is that

it represents the owner of a business process and helps keep track of the overall status of each process (Guo, 2002).

Thirdly, it provides a formal definition of "exception" and clearly separates it from non-exception. With the introduction of "Process Knowledge", it defines non-exception and exception from ontological perspective.

Lastly, it provides a well-formalized notation of the concepts of state, law and process state in the dynamic aspects of business process analysis from ontological perspective, which enables it to provide a simple and clear way to consider business process generally and comprehensively.

Using the Ontology-based workflow management model provides for the requirements from a WFMS that we have listed earlier (Section 2.2).

Flexibility – supporting changes. Changes in the process will be reflected as changes to the law for some states in the process knowledge.

Flexibility – supporting exception handling. The formal definition of "exception" and the controller's mechanism clarify the concept "except" based on formal theory, and support exception identification and handling.

Comprehensiveness. The workflow model is based on a few fundamental constructs (things, properties, state, events, laws) and definitions (stable and unstable states). This provides for a general model.

Interoperability. Its simple and highly abstract representation of business process provides a standard platform for various workflow engines to communicate and coordinate well, thus support enacting business process on the various ongoing instances.

Chapter Five

ONTOLOGY-BASED WORKFLOW MODEL ALGORITHM

In this chapter, a theory-based methodology is developed to provide general procedures on how to employ the Ontology-based Workflow Model to guide workflow system analysis and design. The foundation is the set of BWWP-WFM construct mapping rules and Ontology-based Workflow Model presented in Chapter Four. Following the algorithm, a business process example, which has been presented in Chapter Two, is analyzed to demonstrate the procedures of the algorithm.

5.1 Ontology-based Workflow Model Algorithm

This section introduces the proposed conceptual modeling algorithm, which is derived from BWW Ontology theory. The purpose of this algorithm is to describe the triggering event(s) for a process, find all participants (agents and resources), recursively identify tasks for each participant, and find out their resulting events. Since we regard WFMS as the the controller, i.e., a "super actor", for the workflow environment, here we do not need to identify it once again but rather we just focus the interaction between it and other agents. For a particular process, first we need to find out how the process is initiated, therefore the inputs of the algorithm are the environment situation and the analysis result of BWW process modeling (Wang, 2002). The environment situation provides information on the external events that occur to the observed system. After the algorithm is executed, users could obtain the following outputs (some of them could be inputs for analysis steps that follow):

1) A list of agents who participate in the process: agent_list

2) A list of resources which are used or modified in the process: resource_list

3) A list of the external triggering events: ETR_list

4) A list of the internal requests to the controller: (ITR to C)_list

5) A list of the internal requests from the controller: (ITR from C)_list

6) A list of the responses: RESP_list

7) A list of start tasks: ST_list

8) A list of interval tasks: IT_list

9) A list of end tasks: ET_list

10) The results of performing tasks: result_list

11) The errors which occur: error_list

12) For each agent, there are:

    A list of tasks the agent executes, and their order: task_list.

Among the responses sent to the controller, we have both normal results and errors. Since error handling is an important function of the controller, we need to record all the errors and separate them from normal results. The word "result" refers to the desired outcome of tasks, while the errors are not included. Therefore, we still have an error_list.

5.2 Algorithm Description

The proposed algorithm for identifying these lists consists of the following three parts.

5.2.1 Main Algorithm

Given the environment situation in terms of the external events that occur to the observed things, and the analysis result of BWW process modeling, the algorithm shows how to describe the external triggering event, triggered start task, and the agent who performs start task. Input: Environment situation (interactions of environment and affected things) and analysis result of process modeling.

Environment situation provides the information of the external triggering event(s) that is needed to initiate the workflow. For example, when a customer arrives at the bank counter, the process "deposit" is initiated.

The question here is:

- What are the external triggering event(s)?

The external triggering event is the interactions between environment and affected things. Find the interactions and add to ETR_list.

By identifying the triggering event, we can find out which things are affected by the triggering event. Here, the word "affected" means that the thing's stable state is changed.

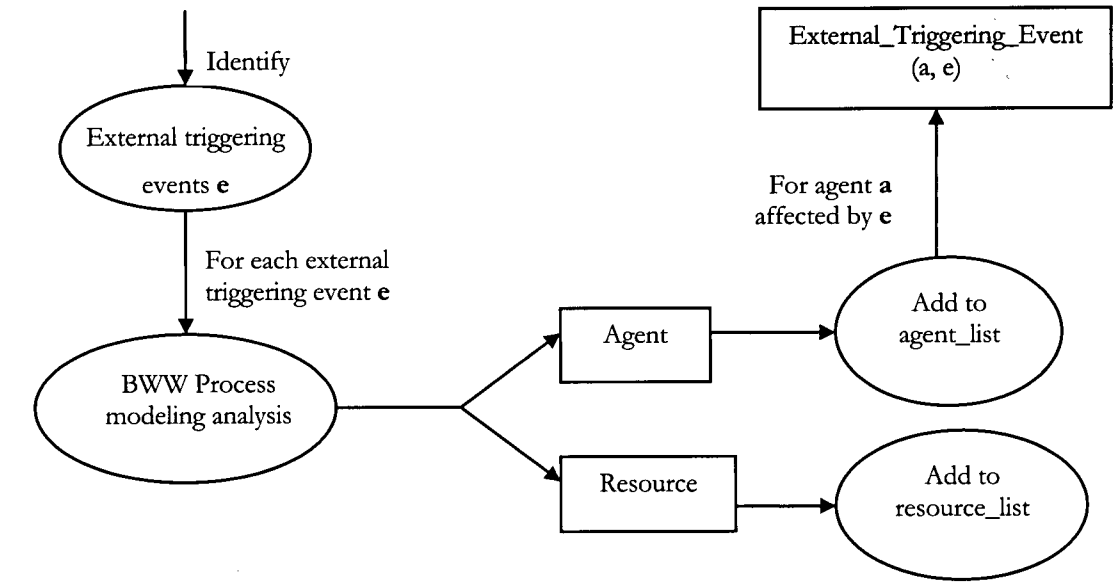
Now we have the following question:

- When the triggering event(s) occurs, what will happen in the process and which participants are affected?

Analyzing the environment situation and workflow domain using BWW process modeling provides an appropriate result for our workflow algorithm. Based on BWW process modeling analysis, we have the affected things differentiated as agents or resources, the sequence of state changes of agents, and the sequence of events resulted from each state change of agents. With the found participants, agents should be added to agent_list and resources should be added to resource_list. For the rest of them, we run our algorithm to transform the process into a workflow system. First, we analyze the external triggering event *e* and the agent(s) *a* affected by it. The subroutine *External_Triggering_Event (a, e)* will be invoked.

The main algorithm is illustrated in Figure 5-1.

Figure 5-1: The Main Algorithm



5.2.2 Subroutine: External_Triggering_Event (input: Agent a, Event e)

This subroutine provides instructions on how to identify the start task triggered by external triggering event *e* and the internal request sent by an agent *a* to the controller when it is affected by event *e*. There are two inputs: Agent *a* and Event *e*. The agent *a* will undergo a state change at the external triggering event *e*, and then perform start task, and send the result of start task to the controller. The result of start task sent to the controller is actually the first internal request to the controller. The subroutine *Internal_Triggering_Request (Agent a, Request r)* will be invoked.

Now we have following questions:

- What start task will be performed by agent *a*?
- What internal request will be sent to the controller?

Here are the two steps to run this subroutine.

1) Identify the start task and add to ST_list. Add the start task to agent *a*'s task_list if it has not been added.

According to mapping rule 8, a task is a sequence of transformations of agent and agent reaches a stable state after it. Therefore the sequence of state changes that agent *a* undergoes is the start task in the workflow.

2) Identify all internal request *r* sent to the controller by agent *a* and added to (ITR to C)_list

Agent *a* performs start task and sends the result to the controller. The result of start task is actually the states of agent *a* and any affected resource and it is an internal request *r* sent to the controller. If the controller changes to a stable state, it does nothing, which means nothing happens in the system when a request sent to the controller. It is a special occasion in reality and it is decided by business rules. If the controller changes to an unstable state, it does the following step.

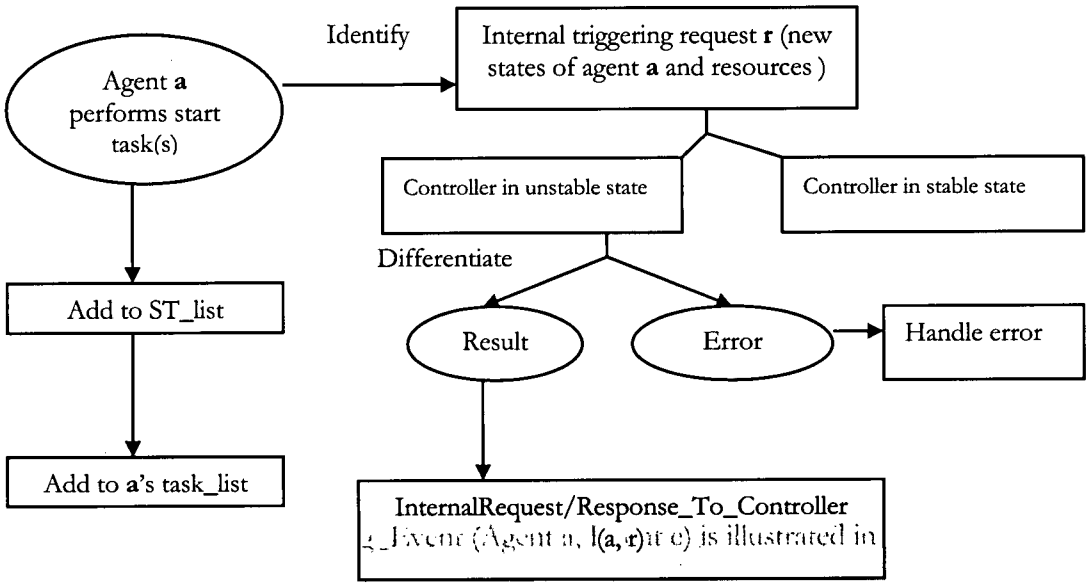
3) Among all the internal request, differentiate the result and error. Add results to result_list and add errors to error_list if they have not been added.

The error is the system state after agent *a* changes state. The result is the lawful state of agent *a*. If error occurs, it is the unlawful state of agent *a*. The controller will report the error to system owner and the process will be suspended.

4) For each internal request to the controller, invoke the subroutine *InternalRequest/Response_To_Controller (Agent a, Request r).*

The subroutine External_Triggering_Event (Agent a, Event e) is illustrated in Figure 5-2.

Figure 5-2: Subroutine External_Triggering_Event (Agent a, Event e)



5.2.3 Subroutine InternalRequest/Response_To_Controller (Input: Agent a, Request r)

This subroutine provides instructions on how to identify the interval task and the agent *a'* who performs it, and what response *resp* will be generated and sent to the controller. The two inputs are: Agent *a* and Internal request *r*.

The questions here are:

- What interval task is triggered by the internal request *r*?
- Which agent will perform the interval task?
- What response will be generated?

We have the following six steps:

1) Identify the interval task(s) and add to IT_list if it has not been added.

Agent *a* sends an internal request *r* to the controller. This interaction arouses the controller to change state. The controller retrieves information from "Process Knowledge" to know what will happen when agent *a* is in such a state. By doing this, the controller decides what interval task(s) will be performed.

2) Identify the agent(s) *a'* who changes state to perform the interval task(s).

the controller retrieves information from "Process Knowledge" to know which agent is going to perform the interval task. The controller assigns task to agent *a'* if the agent will perform the task by changing from a lawful state (*a'*, attribute, value1) to another lawful state (*a'*, attribute, value1'). The controller sends the internal request *r* to agent *a'* to ask it to perform the interval task.

3) Identify the response *resp* sent by agent *a'* after it performs the interval task. Add to RESP_list.

Agent *a'* undergoes state changes when it performs interval task(s). It sends response *resp* to the controller. The response *resp* is represented as (*a'*, attribute, value) and (resource, attribute, value). If the controller changes to a stable state, it does nothing. If it changes to an unstable state, it does the following step.

4) Among all the responses, the controller differentiates the results and errors by referring to "Process Knowledge". Add results to result_list, and add error to error_list if they have not been added.

5) The controller reports the errors to the system owner. For the response sent to the controller, invoke the subroutine *InternalRequest/ Response_To_Controller (a, resp).*

The controller reports the error to the system owner for human intervention. The result sent back to the controller triggers a state change of the controller like new internal request to the controller. The controller will identify the following action from "Process Knowledge" and assign the new task to appropriate agents according to business rules. Therefore the subroutine *InternalRequest/ Response_To_Controller (a, r)* is invoked for further analysis.

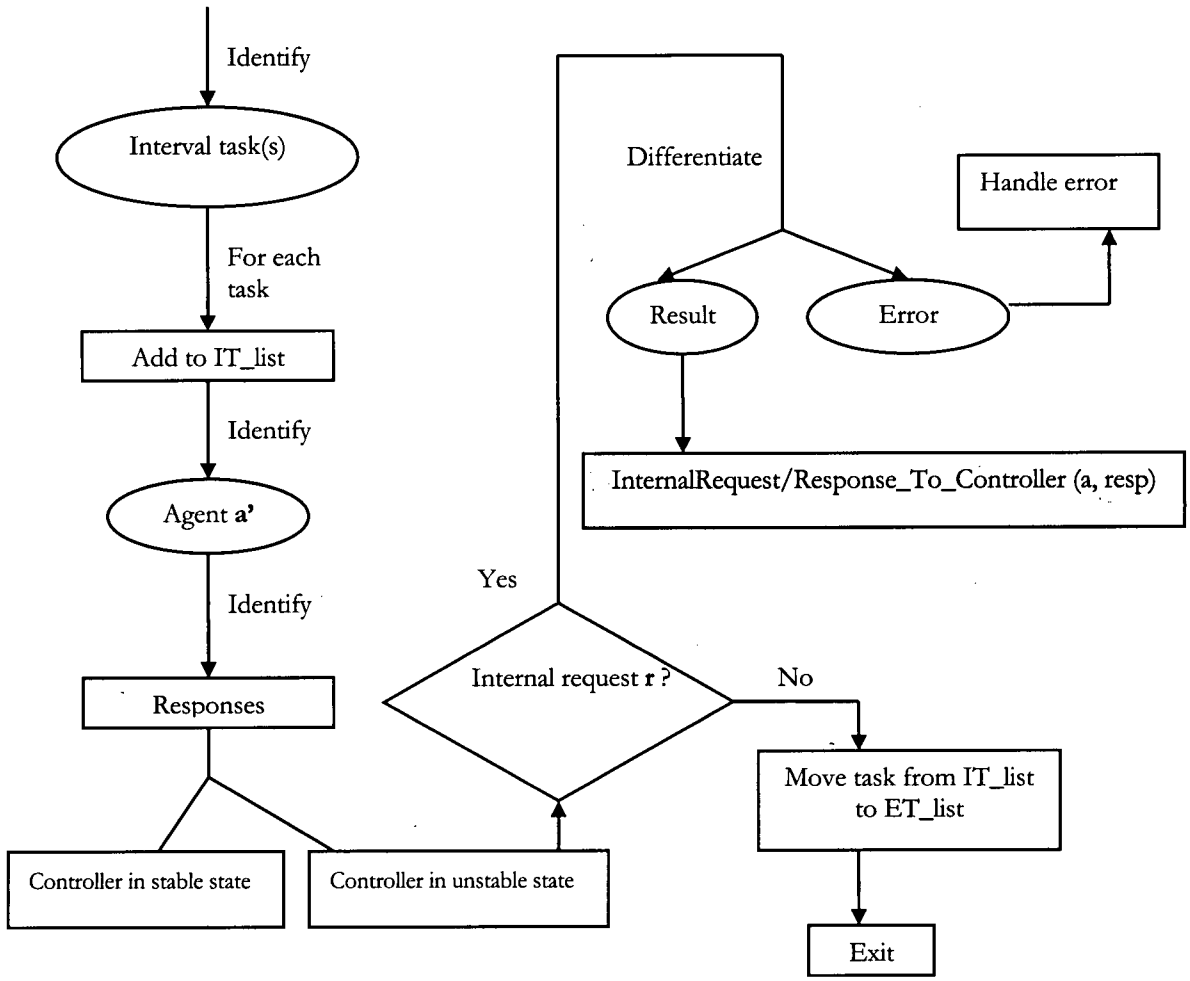
This subroutine will recursively do the following: identify responses, and the subroutine *InternalRequest/ Response_To_Controller (a, r)* will be invoked recursively.

6) Differentiate end task(s)

If the controller reaches a new stable state, which means it stops sending new request after it receives the response from agent *a'*, the task should be identified as end task rather than interval task. Remove it from IT_list and add to ET_list if it has not been added. The subroutine *InternalRequest/ Response_To_Controller* will not be invoked any more. The workflow is completed.

Figure 5-3 shows the algorithm.

Figure 5-3: Subroutine InternalRequest/Response_To_Controller (Agent a, Request r)



5.2.4 Summary of OWFM Algorithm

We proposed an Ontology-based Workflow Management Model Algorithm in previous section. The algorithm is summarized in the following table:

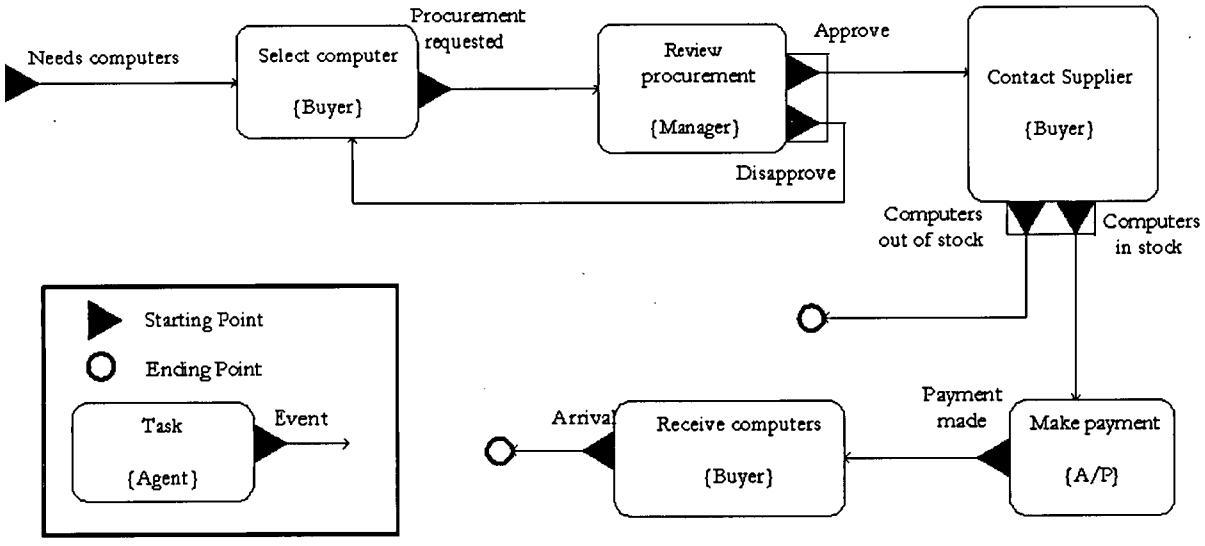
Main/Subroutine	Input	Output	Invoked subroutine
Main	Interactions of environment and observed system; Process modeling analysis result	External triggering events	External_Triggering_Event (a, e)
External_Triggering_ Event (a, e)	Agent a, Event e	Start tasks, internal triggering requests, results/errors	InternalRequest/ Response_To _Controller (a, r)
InternalRequest/ Response_To _Controller (a, r)	Agent a, Request r	Interval tasks, internal requests, results/errors, end tasks	InternalRequest/ Response_To _Controller (a, r)

Table 5-1: Ontology-based Workflow Management Model Algorithm Summary

5.3 Computer Procurement Example

For illustration purpose, this section will use the "Computer procurement" example introduced in Chapter Two to present how to use the algorithm. Figure 5-4 is the example in EDPM.

Figure 5-4: "Computer Procurement" Process in EDPM



5.4 Running Algorithm on Example

The Ontology-based Workflow Model algorithm is run on the "Computer Procurement" example to demonstrate the procedure of the algorithm.

- Step 1: main algorithm

There is one external event generated from the environment: enterprise needs computers. The first preparation is done by analyzing the environment situation (interactions of environment and observed system) and workflow domain using BWW process modeling. Based on BWW process modeling, actors, resources, external events, internal events, and all state changes of actors and resources can be found. The analysis provides the basic results: agents buyer, manager, and A/P; resources computer information, mail and cheque; and all the state changes of agents and resources.

There is one agent "buyer" affected by the external triggering event. The external triggering event "enterprise needs computers" is added to ETR_list.

- Step 2: Subroutine External_Triggering_Event (buyer, needs computer)

Affected by the external triggering event "enterprise needs computers", agent buyer undergoes state changes. Buyer will select computer by referring to computer information. Start task "select computer" is added to ST_list and added to buyer's task_list. Buyer can perform this task with its state change from stable state "idle" to unstable state "selecting computer". In this task, resource "computer information" is used. This resource is not changed after the task.

After performing the start task "select computer", buyer sends internal request "a certain model of computer selected and procurement requested" to the controller. The internal request is added to (ITR to C)_list, and it is differentiated as a result and added to result_list.

An error may occur at the start task "select computer". Agent buyer needs information on various computer models to perform the task. The buyer obtains the information by doing research online. When there is something wrong with the Internet cable, buyer has no Internet access to obtain the information he needs. Buyer changes to an unlawful state "cannot connect to Internet". This situation is not anticipated before the process and is not stored in "Process Knowledge". Such an error makes the process suspended. The error is reported to the controller and added to error_list. Buyer cannot perform the task until the cable is fixed. The process continues at that moment.

- Step 3: InternalRequest/Response_To_Controller (buyer, computer selected and procurement requested)

The controller retrieves information from "Process Knowledge" so that it knows the following action is to review the procurement request. It evaluates the internal request "computer selected" sent by agent "buyer" and decides the triggered interval task should be "review procurement request". The interval task is added to IT_list.

The controller assigns IT1 to agent "manager" because manager is going to perform this task with its lawful transformation. It obtains the information from "Process Knowledge". It sends out the internal request "manager reviews procurement request". The request is added to (ITR from C)_list.

Manager performs IT1 "review procurement request" and sends response to the controller. There are two possible results: request approved, and request not approved. Whichever event occurs, it is differentiated as result and added to result_list.

➢ Step 4: InternalRequest/Response_To_Controller (manager, request approved/request not approved)

For these two different responses, the controller identifies different interval tasks that are triggered by it. Similarly, the controller makes the decision based on the information retrieved from "Process Knowledge".

For the response "request approved", the controller decides that the next interval task is "contact supplier". It is added to IT_list.

For the response "request not approved", the controller decides that the next interval task is "select computer" again. Then go back to step 3. This procedure goes through recursively until the response of "request approved" occurs.

For IT2 "contact supplier", the controller retrieves information from "Process Knowledge" and assigns it to agent "buyer" because buyer has to perform this task with its lawful transformation. The controller sends out the internal request "buyer contacts supplier".

Buyer performs IT2 "contact supplier" and sends response of its transformation to the controller.

If supplier has computers in stock but quantity not enough, buyer changes to a stable state that does not initiate tasks. Only when supplier has enough computers, buyer changes to an unstable state that requires following actions. Here we regard the two situations, computers out of stock and quantity of computers is not enough, as the same kind of situation. There are two possible responses: computer (in exact quantity) in stock, and computer out of stock or quantity not enough. Whichever event occurs, it is differentiated as result and added to result_list.

➢ Step 5: InternalRequest/Response_To_Controller (buyer, computer in stock/computer out of stock)

For these two different internal requests, the controller identifies the different interval tasks that are triggered by it by retrieving information on the following actions to these two events from "Process Knowledge".

For the response "computer (in exact quantity) out of stock", the controller decides that the process is suspended and will not continue until supplier has enough computers in stock.

For the response "computer in stock or quantity not enough", the controller decides that the next interval task is "make payment". Similarly, the decision is made based on information from "Process Knowledge". It is added to IT_list.

The controller assigns IT3 "make payment" to agent "A/P" because A/P has to perform the task with its lawful transformation. The controller sends out the internal request "A/P makes payment". The request is added to (ITR from C)list..

A/P performs IT3 "make payment" and uses resources "mail" and "cheque" during the task. Then A/P sends feedback "payment made" to the controller. During the task, resources "mail" and "cheque" are changed.

➢ Step 6: External_Triggering_Event (buyer, computers arrive)

Another external triggering event occurs: computers arrive. It should be added to ETR_list.

Buyer changes to a new unstable state "receiving computers". The external event "computers arrive" does not trigger a whole process so that the task "receive computers" is not regarded as a start task but an interval task. The procedure goes to InternalRequest/Response_To_Controller (buyer, receiving computers) to go through the steps that follow an interval task.

Buyer sends response to the controller. The controller decides that there is no more tasks in this process because all agents are in stable states therefore the controller is also in stable state. Therefore, the task "receive computers" should be an end task and it is moved from IT_list to ET_list.

A new process "computer assignment and configuration" will be enacted by the event "buyer receives computers". Buyer gives computers to agents who can perform the tasks in the new process, and then buyer reaches a stable state.

All participants reach stable states and the process is completed.

From running the algorithm on the example, we have the following outputs:

Agent_list: A1 buyer, A2 manager, A3 A/P.

Resource_list: RES1 computer information, RES2 mail, RES3 cheque

ETR_list: ETR1 "enterprise needs computers", ETR2 "computers arrive"

(ITR to C)_list: (ITR to C)1 "computer selected and procurement requested"

(ITR from C)_list: (ITR from C)1 "manager reviews procurement request", (ITR from C)2 "buyer contacts supplier", (ITR from C)3 "A/P makes payment".

RESP_list: RESP1 "request approved" or "request not approved", RESP2 "computers (in exact quantity) in stock" or "computers out of stock or quantity not enough", RESP3 "payment made"

ST_list: ST1 "select computer"

IT_list: IT1 "review procurement request", IT2 "contact supplier", IT3 "make payment"

ET_list: ET1 "receive computers"

Result_list: result1 "computer selected and procurement requested", result2 "request approved" or "request not approved", result3 "computer in stock" or "computer out of stock", result4 "payment made"

Error_list: error1 "buyer cannot connect to Internet"

Buyer's task list: T1 "select computer", T2 "contact supplier", T3 "receive computers"

Manager's task list: T1 "review procurement request"

A/P's task list: T1 "make payment"

Chapter Six

CONCLUSION

6.1 Conclusion

The main objective of this thesis is to find a formal theoretical foundation for the analysis and design for workflow management system.

We first reviewed some basic workflow concepts from peer-reviewed literature. Workflow management system was also discussed. Generic constructs of workflow management were presented with the relationships between each other. We then investigated the requirements of workflow management model which are derived from requirements of a workflow management system. Some related works were examined with respect to the requirements. We found that most of the surveyed research work was based on pragmatic technologies and borrowed theories from adjacent areas "without appropriate adjustment". We concluded that most workflow models are not flexible and adaptive enough because they lack of a formal theoretical foundation.

We chose BWW Ontology as our theoretical foundation for our workflow model. First we introduced the basic concepts of BWW Ontology. Then an ontological process model (Wang, 2002) was introduced by applying ontological analysis to a sample process. The sample process consists of six things (including actors and non-actors), which interact with each other. By looking at each thing's state changes, the relationships between BWWP concepts were clarified.

We proposed to build a theory-based workflow model by using process related ontological concepts (BWWP concepts). Based on the review of workflow generic constructs and BWWP

concepts, a set of BWWP-WFM construct mapping rules were found. The concept of the controller was introduced and the role and main functions of the controller were discussed. After these two sets of analysis, an ontological workflow model was presented. The ontological workflow model combined BWW Ontological analysis and concept of the controller. The model was focused on the dynamic parts of business process, which involved interactions between agents and the controller, and between agents and resources. Certain workflow constructs were discussed in detail, such as tasks, events, and exceptions.

We then proposed an ontological workflow model algorithm to demonstrate how to use the model in business practices. The algorithm had three parts: main algorithm (environment, BWW process modeling), subroutine External_Triggering_Event (agent, event), and subroutine InternalRequest/Response_To_Controller (agent, request/response). It provided a recursive mechanism to investigate each triggered task and each result of tasks and generates a workflow model when it completes.

6.2 Contribution

One of the significant contributions of this thesis is the BWWP-WFM construct mapping rules. The set of mapping rules extended previous research work, BWWP-PML construct mapping rules (Wang, 2002), by taking into consideration some important workflow constructs such as process state and exception, which appropriately links BWW Ontology with workflow management.

Another significant contribution is that it introduced and extended the concept of the controller. By formalizing the role of the controller from BWW Ontological perspective and the relationship between the controller and agents, an ontological workflow model provided theoretical guideline for the analysis and design of workflow management system.

The third significant contribution is that it proposed a formal definition of exception in BWW Ontological context. The concepts of "result" and "exception" were clearly differentiated in ontological terms and operationalized in the way of storing information in "Process Knowledge".

6.3 Limitations and Future Research

There are some limitations to the proposed approach that need to be addressed and that require future research.

One limitation is that it only suggested establishing a "Process Knowledge" to store all lawful states of participants and the following actions to these lawful states but it did not suggest a way on how to set it up in details. Further technical and detailed research work on how this could be operationalized needs to be done in future research.

Another limitation is that it was tested using a limited number of hypothetical examples. In future research, the model could be applied to actual cases so that the model and the algorithm were tested in practice.

Bibliography

Agostini, A., and De Michelis, G. 2000. Improving Flexibility of Workflow Management System. *Business Process Management 2000*: 218-234.

Agostini, A., De Michelis, G., and Petruni, K. 1994. Keeping Workflow Models as Simple as Possible. In: *CSCW, Petri Nets and related formalisms*, Proceedings of the 15th International Conference on Application and Theory of Petri Nets; Zaragoza; 1994

Agostini, A., and De Michelis, G. 2000. A Light Workflow Management System Using Simple Process Models. *Computer Supported Cooperative Work* 9(3/4): 335-363 (2000).

Alonso, G.; Hagen, C.; Agrawal, D.; El Abbadi, A.; Mohan, C. 2000. Enhancing the Fault Tolerance of Workflow Management Systems. *Concurrency, IEEE [see also IEEE Parallel & Distributed Technology]*, Volume: 8. Issue: 3, July-Sept. 2000. pp. 74 -81

Basu, A., and Kumar, A. 2002. Research Commentary: Workflow Management Issues in e-Business, *Information Systems Research* (13:1), 2002, pp. 1-14.

Borghoff, U.M., and Schlichter, J. *Computer-Supported Cooperative Work: Introduction to Distributed Applications*, Springer, 2000.

Borgida, A., and Murata, T. 2000. Handling of Irregularities in Human Centered System: a Unified Framework for Data and Processes. *Software Engineering, IEEE Transactions on*, Volume: 26 Issue: 10 , Oct. 2000 Page(s): 959 -977

Bowers, John, Button, Graham, and Wes Sharrock (1995): Workflow from Within and Without: Technology and Cooperative Work on the Print Industry Shopfloor. In H. Marmolin, Y. Sundblad and K. Schmidt (eds.): ECSCW'95. *Proceedings of the Fourth European Conference on Computer Supported Cooperative Work, Stockholm, Sweden, September 10-14, 1995.* Dordrecht, The Netherlands: Kluwer Academic Publishers, pp. 51-66.

Casati, F., Fugini, M.G., and Mirbel, I. 1999. An Environment for Designing Exceptions in Workflows. *Information Systems.* Vol.24, No.3, pp.255-273. 1999.

Casati, F., and Discenza, A. 2001. Modeling and Managing Interactions among Business Processes. *Journal of Systems Integration*, 10, 145–168 (2001) Kluwer Academic Publishers, Boston. Manufactured in The Netherlands.

Casati, F., Ceri, S., Paraboschi, S., and Pozzi, G. 1999. Specification and Implementation of Exception in Workflow Management System. *ACM Transactions on Database Systems*, Vol. 24, No. 3, September 1999, Pages 405–451.

Casati, F., Fugini, M.G., Mirbel, I., and Pernici, B. 2002. WIRES: A Methodology For Developing Workflow Applications. *Requirements Engineering* (2002) 7:73-106.

Casati, F. Models, Semantics, and Formal Methods for the Design of Workflows and Their Exceptions, PhD Dissertation, *Pipartimento di Electtronica e Informazione*, Politecnico di Milano, Milano, Italy, 1998.

Chiu, K.W. 2001. A Three-Layer Model for Workflow Semantic Recovery in an Object-Oriented Environment. Volume 2224, Issue, pp0541- Lecture Notes in Computer Science.

Chiu, K.W. 2000. Exception Handling in an Object-Oriented Workflow Management System, PhD Dissertation, Department of Computer Science, The Hong Kong University of Science and Technology, Hong Kong, 2000.

Divitini, M., Hanachi, C., and Sibertin- Blanc, C. Inter-Organizational Workflows for Enterprise Coordination. In: *Coordination of Internet Agents: Models*, Technologies and Applications, A. Omicini, F. Zambonelli and M. Klusch (eds.), Springer, 2001.

Dong, G., Hull, R., Kumar, B., Su, J., and Zhou, G. 1999. A Framework for Optimizing Distributed Workflow Executions. Proc. Int. Workshop on Database Programming Languages (DBPL), 1999

Dourish, P., Holmes, J., MacLean, A., Marqvardsen, P., and Zbyslaw, A. 1996. Freeflow: Mediating Between Representation and Action in Workflow Systems. In M. S. Ackerman (ed.): CSCW'96. *Proceedings of the Conference on Computer Supported Cooperative Work*, Cambridge, MA, November 16-20, 1996. New York, NY: ACM Press, pp. 190-198.

Eder, J., and Liebhart, W. 1995. The Workflow Activity Model WAMO. Conference on Cooperative Information Systems, Vienna, Austria, 1995. pp. 87-98.

Eder, J., and Liebhart, W. 1996. Workflow Recovery. *CoopIS* 1996: 124-134.

Ellis, C.A., and Keddara, K. 2000. A Workflow Change Is A Workflow. *Business Process Management 2000*: 201-217.

Green, P., and Rosemann, M. 2000. Integrated Process Modeling: An Ontological Evaluation. *Inf. Syst. 25(2) (2000)*: 73-87.

Guo, H. *A Goal Oriented and Decentrally Controlled Workflow Model For Facilitating Exception Handling.* MSc Thesis, Faculty of Commerce and Business Administration, University of British Columbia. 2002

Hagen, C., and Alonso, G. 2000. Exception Handling in Workflow Management Systems. *IEEE transactions on software engineering.* Vol.26, No.10, October 2000. pp.943-958.

Hui, S. *An Object-Oriented Workflow Management System.* Msc Thesis, Faculty of Commerce and Business Administration, University of British Columbia. 1997

Jablonski, S. 2000. Workflow Management between Formal Theory and Pragmatic Approaches. *Business Process Management 2000*: 345-358.

Jablonski S, Bussler C. *Workflow Management: Modeling Concepts, Architecture and Implementation.* Int. Thomson Computer Press: Bonn, Germany, 1996.

Klingeman, J. 2000. Controlled Flexibility in Workflow Management. CAiSE 2000: 126-141.

Koulopoulos, T. M. (1995): *The Workflow Imperative.* New York, NY: Van Nostrand Reinhold.

Kradolfer, M., Geppert, A., and Dittrich, K.R. 1999. Workflow specification in TRAMs. ER1999. 263-277.

Lee, M., Han, D., and Shim, J. 2001. Set-based Access Conflict Analysis of Concurrent Workflow Definition. Information Processing Letters 80 (2001) 189–194

Leymann, F., and Roller, D. Production Workflow: Concepts and Techniques, Prentice Hall PTR, New Jersey, 1999.

Luo, Z., Sheth, A., Kochut, K., and Miller, J. 2000. Exception Handling in Workflow Systems. *Applied Intelligence: the International Journal of AI*, Neural Networks, and Complex Problem-Solving Technologies (13:2), 2000, pp 125-147.

Norman, D. A. 1993. *Cognitive Artifacts.* In J. M. Carroll. (ed.): Designing Interaction. Psychology at the Human computer Interface, Cambridge, UK: Cambridge University Press, pp. 17-38.

Sadiq, W., and Orlowska, M.E. 1997. On Correctness Issues in Conceptual Modeling of Workflows. Proceedings of the 5th European Conference on Information Systems (ECIS '97), Cork, Ireland , June 19-21, 1997.

Schmidt, K. 1997. Of maps and scripts: the status of formal constructs in cooperative work. In S. C. Hayne and W. Prinz (eds.): GROUP'97. Proceedings of the International ACM SIGGROUP Conference on Supporting Group Work, Phoenix, AR, November 16-19, 1997. New York, NY: ACM Press, (1997) 138-147

Schmidt, K., Bannon, L.: Taking CSCW Seriously: Supporting Articulation Work. *Computer Supported Cooperative Work* (CSCW). An International Journal, Vol.1, nos. 1-2, (1992) 7-40

Suchman, L. 1987. Plans and Situated Actions. The problem of human-machine communication. Cambridge, UK: Cambridge University Press.

van der Aalst, W.M.P. 1999. Flexible Workflow Management Systems: An Approach Based On Generic Process Model. DEXA 1999: 186-195

van der Aalst, W.M.P., Barros, A.P., ter Hofstede, A. H.M., and Kiepuszewski, B. 2000. Advanced Workflow Patterns. CoopIS 2000: 18-29

van der Aalst, W.M.P., and Basten, T. 2002. Inheritance of Workflows: An Approach to Tackling Problems Related to Change. TCS(270)1-2: 125-203

van der Aalst, W.M.P., and van Dongen, B.F. 2002. Discovering Workflow Performance Models from Timed Logs. EDCIS 2002: 45-63

van der Aalst, W.M.P., Basten, T., Verbeek, H.M.W., Verkoulen, P.A.C., and Voorhoeve, M. Adaptive Workflow - on the Interplay Between Flexibility and Support. The IJCAI'99 Workshop on Intelligent Workflow and Process Management, Stockholm, Sweden, 1999, pp. 36-45.

Vossen, G., and Weske, M. 1999. The WASA2 Object-Oriented Workflow Management System. SIGMOD Conference: 587-589.

Wand, Y . 1989. A Proposal for a Formal Model of Objects. In W. Kim and F. Lchovsky,edit ors, *Object-Oriented Concepts, Languages, Applications and Databases*, pages 537 - 559. ACM Press. Addison-Wesley.

Wand, Y. and Weber, R . 1989. An Ontological Evaluation of Systems Analysis and Design Methods. In E. Falkenberg and P. Lingreen,editors, *Information System Concepts: An In-Depth Analysis.* Elsevier Science Publishers B.V., North-Holland.

Wand, Y. and Weber, R. 1990a. Mario Bunge's Ontology as a Formal Foundation for Information Systems Concepts", in: *Studies in Bunge's Treatise on Basic Philosophy*, G. Dorn and P. Weingartner (eds.), the Poznan Studies in the Philosophy of the Sciences and the Humanities, Rodopi, Amsterdam, 1990, pp. 123-150.

Wand, Y. and Weber, R. 1990b. An Ontological Model of an Information System. *IEEE Transactions on Software Engineering*, Vol. 16, No. 11, November 1990, pp. 1282-1292.

Wand, Y. and Weber, R. 1993. On the Ontological Expressiveness of Information Systems Analysis and Design Grammars. *Journal of Information Systems*, 1993, No. 3.

Wand, Y. and Weber, R. 1995. Towards a Theory of Deep Structure of Information Systems. *Journal of Information Systems.*

Wand, Y. and Wang, R. 1996. Anchoring Data Quality Dimensions in Ontological Foundations. *Communications of the ACM*, 39 (11) November 1996.

Wand, Y., Storey, V. and Weber, R. 1999. An Ontological Analysis of the relationship Construct in Conceptual Modelling. *ACM Transactions on Database Systems*, Vol. 24, No. 4, December 1999, pp. 494-528.

Wang, Q. 2002. *A Proposal For A Process Modeling Methodology.* MSc Thesis, Faculty of Commerce and Business Administration, University of British Columbia. 2002

Wargitsch, C., Ellis, C., Keddara, K., and Rozenberg, G.: Dynamic Change within Workflow Systems. In: Proceedings of the Conference on Organizational Computing Systems. ACM Press, New York (1995) 10-21

WfMC "Workflow Reference Model," The Workflow Management Coalition Specification TC00-1003, Workflow Management Coalition, 1995.

WfMC "Workflow Handbook 2002". website: http://www.wfmc.org

WfMC "Workflow Handbook 2003". website: http://www.wfmc.org

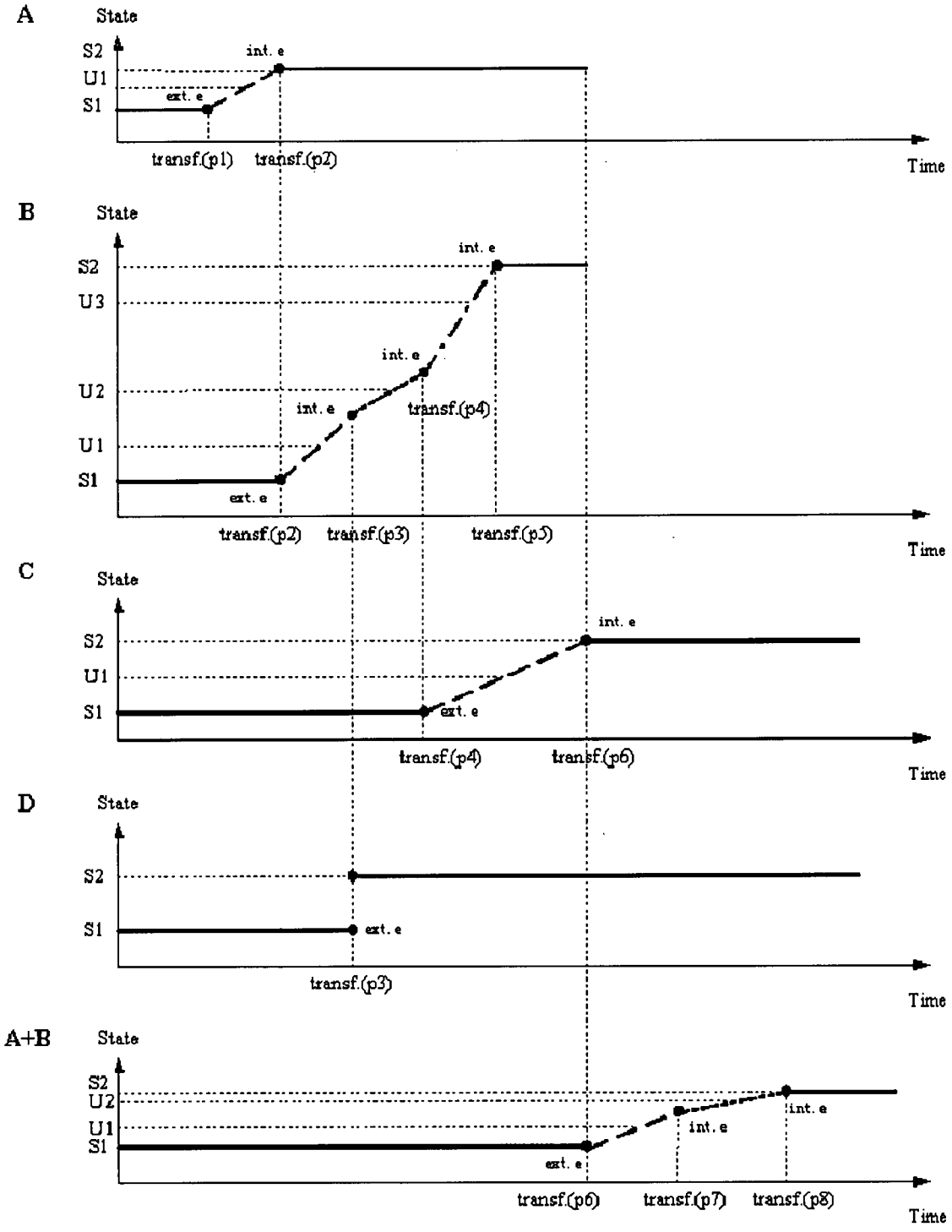
WfMC "WfMC Terminology and Glossary," WFMC-TC-1011, Workflow Management Coalition, 1999.

White, T. E., and Fischer, L. (eds.) (1994): The Workflow Paradigm. Alameda, CA: Future Strategies.

Wirtz, G., Weske, M., and Giese, H. 2001. The OCoN Approach to Workflow Modeling in Object-Oriented System. *Information Systems Frontiers* 3:3, 357–376, 2001 Kluwer Academic Publishers. Manufactured in The Netherlands.

Zeng, L., Ngu, A., Benatallah, B., and O'Dell, M. 2001. An Agent-Based Approach for Supporting Cross-Enterprise Workflows. Australiasian Database Conference January 29 - February 01, 2001.

APPENDIX A: STATE CURVES (ADAPTED FROM WANG, 2002)



APPENDIX B: ONTOLOGY-BASED WORKFLOW MODEL IN FULL DETAILS

1) An external triggering event occurs to triggers agent A1 to change state and perform start task. A1 affects resource R1 when performing start task.

2) A1 sends result of start task, i.e., the first internal request to the controller (ITR to C). the controller evaluates the result and sends the first internal request (ITR from C) to agent A2 to ask it to perform the first interval task (IT1).

3) A2 performs IT2 and affects resource R2. The controller receives response from A2 (RESP1) and identifies the following action. It sends the second internal request (ITR from C) to A3 to ask it to perform the second interval task (IT2).

4) A3 performs IT2 and affects resources R3 and R4. A3 sends response of IT2 (RESP2) to the controller. The controller identifies the following action and sends the third internal request (ITR from C) to agent A4.

5) A4 performs the last task, ET, and affects resource R5. A4 sends response (RESP3), which is the result of end task and does not trigger other interval tasks, to the controller.

6) The controller reaches a stable state and stops sending out requests. The workflow is completed.