A Goal Oriented and Decentrally Controlled Workflow Model
for Facilitating Exception Handling

by

Huijin Guo

B. Econ., Zhejiang University, 1992

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

IN

THE FACULTY OF GRADUATE STUDIES
(Faculty of Commerce and Business Administration)

We accept this thesis as conforming to the required standard

THE UNIVERSITY OF BRITISH COLUMBIA

September 2002

© Huijin Guo, 2002
In presenting this thesis in partial fulfilment 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.
Abstract

More and more organizations are starting to use workflow management systems (WfMS) to monitor, control and manage business processes. However, currently available commercial workflow systems are rather rigid and cannot meet the requirements of a dynamic and fast-changing business. Exception handling capabilities of the systems are very limited. Some research work has been done to address the issue by extending database technologies in workflow domain. In this thesis, we begin with a brief review of some main workflow concepts and do a survey of current research work on exception handling. We propose a leveled workflow model based on Micro-Organization Activity Processor (MOAP) and Object-Oriented Workflow Model (OOWM), which is an extension of Object-Oriented Enterprise Modeling (OOEM). The MOAP construct is extended with a goal concept and the OOEM service concept. We then propose a mechanism for exception handling which utilizes artificial intelligence technologies such as means-end analysis. We further demonstrate the functionalities and exception handling processes with a web-based simulator by applying some workflow exception cases.
# Table of Contents

Abstract ........................................................................................................................ ii 
Table of Contents ........................................................................................................... iii 
List of Tables .................................................................................................................. vi 
List of Figures ................................................................................................................. vii 
Acknowledgments ......................................................................................................... ix 
1  Introduction ................................................................................................................ 1  
   1.1 Motivation .............................................................................................................. 1  
   1.2 Thesis Objectives ................................................................................................. 2  
   1.3 Thesis Outline ..................................................................................................... 3  
2  Workflow Management System ................................................................................. 5  
   2.1 Workflow and Related Concepts ......................................................................... 5  
   2.2 Workflow Management Systems ......................................................................... 8  
   2.3 Challenges and Limitations .................................................................................. 10 
3  Workflow Exceptions and Related Research Work .................................................. 12  
   3.1 Exceptions in Workflow ....................................................................................... 12  
       3.1.1 Exception Scope: Schema and Instance ....................................................... 12  
       3.1.2 Exception Level: Expected and Unexpected ............................................... 13  
       3.1.3 Level of Abstraction: Domain, Process, Resource and Infrastructure. .14  
   3.2 Scope of Exception in this Thesis ....................................................................... 15  
   3.3 Survey of Related Research Work ....................................................................... 16  
       3.3.1 The Workflow Activity Model .................................................................... 16  
       3.3.2 OPERA Process Support System ................................................................. 19  
       3.3.3 ADOME-WFMS ......................................................................................... 21  
       3.3.4 CapBasED-AMS ....................................................................................... 24  
       3.3.5 Adaptive Workflow .................................................................................... 26  
   3.4 Summary of Survey .............................................................................................. 27  
   3.5 Different Views of Workflow ............................................................................. 30  
   3.6 Micro-Organization Activity Processor (MOAP) ................................................ 32
List of Tables

Table 3-1 Category of Exception Handling Approach .................................................. 29
Table 3-2 Object Activity Template (adapted from (Hui 1997)) .................................. 39
Table 5-1 Controller MOAP – Purchasing Process .................................................... 69
Table 5-2 MOAP – Purchasing Dept ........................................................................... 70
Table 5-3 Exception Handling Example in ADOME-WFMS ....................................... 87
Table 5-4 Comparison of ADOME-WFMS and MOAP-Wf ......................................... 90
Table 6-1 Simulator Functional Modules ..................................................................... 95
List of Figures

Figure 2-1 Workflow Terminology (adapted from WfMC (1999)) .................. 6
Figure 2-2 WfMC's Workflow Reference Model (adapted from (WfMC 1995)) .... 9
Figure 3-1: Event-State for Activities (adapted from (Eder and Liebhart 1995)) .... 18
Figure 3-2: Control Flow of Exception Handling in OPERA
(adapted from (Hagen and Alonso 1998)) .................................................... 20
Figure 3-3: Architecture of CapBasED-AMS
(adapted from (Karlapalem et al 1995)) ..................................................... 24
Figure 3-4 Synthesized Adaptive Workflow Approach
(adapted from (Han et al 1998)) ................................................................. 26
Figure 3-5 Micro-Organization Activity Processor (adopted from (Woo and Lochofsky 1991)) ............................................................... 32
Figure 3-6 ADEPT Agent Architecture (adapted from (Jennings et al 1998)) .... 35
Figure 3-7 Object-Oriented Workflow Model (adapted from (Hui 1997)) ......... 38
Figure 4-1: Modified Micro-Organization Activity Processor
(adapted from (Woo and Lochofsky 1991) with modifications) ..................... 42
Figure 4-2 Goal Tree Diagram ................................................................. 46
Figure 4-3 Goal for Purchase Process ..................................................... 47
Figure 4-4: OOWM Object vs. Modified MOAP ...................................... 56
Figure 4-5: A MOAP-Based Workflow Model ........................................ 58
Figure 4-6: An Example of Business Views .......................................... 59
Figure 5-1: Execution Mechanism of MOAP-Wf .................................. 66
Figure 5-2 MOAP-Wf Structure for Purchasing Process ......................... 68
Figure 5-3: Overview of Exception Handling Mechanism ....................... 73
Acknowledgments

First of all, I am very grateful to my thesis supervisor Dr. Carson Woo. Without his research guidance, encouragement and support, this thesis would not have been possible. I also wish to express my gratitude to Dr. Yair Wand and Dr. Jacob Steif for being my thesis committee members and for their insightful input and advice.

I also thank Gary Schmidt for his excellent editing and Leon Qiu for fruitful discussion on programming. Last but not least, I would like to thank all my fellow Master's students for their support.
1 Introduction

1.1 Motivation

In recent years, workflow management systems have been one of the most-spotlighted subjects for both the software industry and the research community. For the industry, there are more than 200 workflow products commercially available (WfMC, 1997). Vendors vary from small and mid-sized companies to big names such as IBM (IBM, 2002), Oracle (Oracle, 2002), SAP (SAP, 2002), and HP (HP, 2002). For the research community, there are also a number of projects and research work on workflow specification and verification, transactional workflows and extensions of ideas from active database management to workflow management (Mohan, 1997).

The focus on business process reengineering as a cost saving and service improvement measure has been instrumental in this recent interest in workflow management systems (Mohan, 1997). As an information technology tool to automate, control and coordinate business processes, workflow management systems help companies to streamline processes, reduce process time, and improve business planning.

Despite their popularity in the industry, current workflow systems might be hindered from playing a more effective role in business process re-engineering due to the systems' rigid designs, which were optimized for fixed and repetitive processes. The rigidity remains a problem in the context where people and organizations have to evolve constantly in a changing environment (Kappel et al., 2000). Because of the
dynamic and ever-changing nature of business, a process description can rarely capture all possibilities of what will occur. Exceptions always happen. However, these dynamic changes and exceptions are not adequately addressed – e.g. how to uncover exceptions and how the exceptions should be resolved, by current commercial products or even research prototypes (Chiu, 2000; Hagen and Alonso, 1998; Han et al., 1998; Klein and Dellarocas, 2000).

1.2 Thesis Objectives

Motivated by the above observations, we try to find out how workflow systems can be made flexible, adaptive, and dynamic, and how this in turn can handle exceptions and changes.

There have been a number of research works done on workflow exception handling, (e.g., (Casati et al., 1999; Chiu, 2000; Eder and Liebhart, 1995; Hagen and Alonso, 1998; Klein and Dellarocas, 2000)). However, most of them only focus on extending database technology and examine the problem from a transaction processing point of view. Particularly they lack a mechanism to handle unexpected exceptions and the support of process learning. As Abbott and Sarin noted that, “It should be possible to build the process definition ‘library’ or hierarchy based on experience in executing processes. On completion of a process (or before that, if necessary) it should be possible to review how it was performed and to abstract information from it in order to ‘save’ a process definition, or components thereof, in the library” (1994).
To address these limitations, we propose a flexible workflow model by extending the Micro-Organization Activity Processor (MOAP) architecture (Woo and Lochovsky, 1991) and incorporating Object-Oriented Workflow Model (OOWM) (Hui, 1997). The advantage of OOWM is that it captures an organizational process from a company-wide perspective and focuses on how independent objects work together in order to achieve company objectives (Hui, 1997). The MOAP architecture is an intelligent agent that supports acquisition and communication of knowledge through learning and reasoning.

The objectives of this thesis are:

- To construct a flexible and dynamic workflow model that supports exception handling;

- To explore and present the exception handling capability and mechanisms of the proposed workflow system and find out what kinds of exceptions can be handled.

- To develop a workflow system simulator to demonstrate and study the feasibility and limitations of the proposed approach.

1.3 Thesis Outline

The thesis is outlined as follows:

In Chapter 2, we give an overview of workflow management systems and definitions of some common terms and concepts that are used throughout the thesis.
Chapter 3 has three parts: first, we examine the nature of workflow exceptions and discuss some different classifications. Then we discuss features and limitations of some surveyed research work. The last part introduces OOWM and MOAP concepts, which serve as the theoretical foundation of this thesis.

In Chapter 4, the proposed MOAP-Based Workflow Model (MOAP-Wf) is presented.

In Chapter 5, we first discuss the normal execution mechanism of MOAP-Wf. Then by giving a business process example, we elaborate how exceptions are handled in MOAP-Wf.

We present the design and demonstrate the implementation of a workflow execution simulator based the MOAP-Wf in Chapter 6.

Chapter 7 concludes the thesis by reviewing the contributions, acknowledging the limitations, and suggesting future research.
2 Workflow Management System

In this chapter, we briefly review main concepts of workflow systems so as to build a common ground of terminology for analysis, because researchers and commercial software vendors might have different definitions and interpretations on the same term.

2.1 Workflow and Related Concepts

Generally speaking, 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). There are also some other similar definitions, for example Jablonski and Bussler say 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, while 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.

Because the terms workflow and process are so closely related, researchers sometimes use them interchangeably. In any case, a workflow is basically a business process. From a modeling point of view, a process model describes the structure of business process in the real world, while a workflow model talks about the aspects of a business that are run on a computer (Leymann and Roller, 1999).
WfMC’s terminology can be described in Figure 2-1. Other main 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 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 to automate necessarily 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 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 Systems*

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 the 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 executions.

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

- **Workflow Enactment Service** – is a workflow system's core component, which supports the execution of workflow processes by providing a run-time execution environment. 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)** - are 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 call third-party applications, e.g. ERP systems, to perform a task.

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

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

In this thesis, we focus on process definition and workflow enactment service. We propose a goal- and agent-oriented workflow modeling, execution and exception handling mechanism.

### 2.3 Challenges and Limitations

Despite their popularity in the industry, current workflow systems might be hindered from playing a more effective role in business process re-engineering due to the systems' rigid designs, which were optimized for fixed, repetitive processes. The rigidity is widely cited as a critical problem in a context where people and organizations have to evolve in a constantly changing and dynamic business environment (Abbott and Sarin, 1994; Chiu, 2000; Grigori *et al.*, 1996; Han *et al.*, 1998; Heinl *et al.*, 1999; Keddara, 1999).

Business processes involving people usually cannot be described completely a priori by a strict activity schedule, and exceptional situations may arise. Alternative paths for achieving the global goal must be provided or automatically generated as needed (Borghoff and Schlichter, 2000). In business reality there are some special
cases in which the defined process rules are by-passed. For example, a CEO may approve sales orders to a key customer without the task of credit checking.

However, current workflow management systems are ill-suited to handling such exceptions and do not provide support for uncovering what kinds of exceptions can occur in a given process model and how they can be resolved (Klein and Dellarocas, 2000). When such exceptions occur, the solution of current workflow management systems is to redefine and re-execute the process. One example of current workflow management systems is IBM MQSeries Workflow which uses the following steps to handle an exception (IBM, 2001): 1). Create new process model. 2). Export the new process model from Buildtime and import FDL (MQ Workflow Description Language) file into Runtime environment. New process instances will follow the new model while existing old process instances do not. Existing processes always use the template (model) that exists in the Runtime on the moment that the instances are created.

This type of solution has the following shortcomings: 1). It is inefficient in that it requires redefinition and re-execution of workflow process. 2). Redefining workflow model affects existing running instances. 3). Each time an exception occurs, a new model is required to incorporate exception handling procedures, which can greatly complicate process models and 'obscure' the 'preferred' process, making it difficult to define, understand and modify (Klein and Dellarocas, 2000).

In this thesis, we try to address these challenges and limitations with the proposed goal- and agent-oriented approach.
3 Workflow Exceptions and Related Research Work

In the previous chapter, we reviewed the main concepts of workflow management systems. Challenges and limitations of current WfMS are also discussed. We will now cover the details of workflow exceptions.

In this chapter, we first review some of the definitions and classifications of workflow exceptions, and then proceed with a survey of previous research work on exception handling.

3.1 Exceptions in Workflow

Currently there is not established framework for exception handling in WfMSs (Chiu, 2000). Different definitions can be found in the literature. For example, Chiu defines exception as an event which deviates from normal behavior 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 the business environment changes, when organizational structure changes, or when there are system errors.

In this section, we review the classification of workflow exceptions and the characteristics that might influence exception handling approaches.

3.1.1 Exception Scope: Schema and Instance

An exception can be classified as schema exception or instance exception depending on its scope. Schema exceptions occur when a workflow process definition
is changed. However, when an exception occurs only for one process execution, it is an instance exception.

Some research work, e.g. Ellis and Keddara 2000, differentiates schema change and instance change, and defines instance change as an exception. We classify the former as ‘schema exception’ in a wider sense in that a ‘schema change’ is still a deviation, thus an ‘exception’ from the original one.

3.1.2 Exception Level: Expected and Unexpected

An exception can also be classified as expected or unexpected (Chiu, 2000; Eder and Liebhart, 1995) according the workflow level at which it occurs.

Expected exceptions are those deviations and special cases that can be anticipated. Special mechanisms or ‘exception handlers’ can be embedded in normal workflow to handle such exceptions (Basu and Kumar, 2002). Unexpected exceptions, on the other hand, are those deviations or special cases that cannot be anticipated. Another similar definition given by Casati is that unexpected exceptions correspond to inconsistencies between the business process in the real world and its corresponding workflow description (1998). No special mechanisms or handlers are available to handle unexpected exceptions. Human intervention is usually required. Expected exceptions are to be solved at workflow system level, while unexpected exceptions need to be handled at the process definition (process modeling) level (Eder and Liebhart, 1995).

Some other similar classifications in this category are: known/unknown exceptions and detectable/undetectable exceptions (Luo et al., 2000). Unknown exceptions are beyond the workflow system’s current knowledge, while known
exceptions are anticipated within it. Detectable and undetectable exceptions are defined according to whether a workflow system is capable or incapable of detecting such exceptions.

3.1.3 Level of Abstraction: Domain, Process, Resource and Infrastructure

Another classification is to use a 'separating concerns' strategy to provide a taxonomy of workflow adaptation according to abstraction levels – domain, process, resource, and infrastructure (Han et al., 1998).

At the domain level, workflow systems should be adapted to changes in the business context such as laws and regulations. For example, the changes of US immigration policies have significant impact on the process of issuing visas. Domain level adaptation may in turn result in a series of adaptation requirements at different levels as discussed below. Process level adaptation includes workflow model evolution and workflow instance ad-hoc modifications. This level adaptation is equivalent to a schema exception and an instance exception which we defined in section 3.1.1. At resource level, workflow systems need to be adapted to changes and rearrangement of supporting workflow resources, including organization-related resources such as personnel changes and data-related resources such as data structures. The fourth level of adaptation is infrastructure, which deals with technical advances and changes of software system configuration.
3.2 Scope of Exception in this Thesis

Because there are many different definitions and classifications as discussed above, we limit the scope of exceptions in this thesis as follows:

First, we focus on business process level exceptions, i.e. from the viewpoint of business process rather than that of software. The other levels of adaptation, for example domain and infrastructure, can be treated as ‘external’ to the workflow system (Chiu, 2000). We do not consider those basic failures and application failures such as program failures and system crashes (Eder and Liebhart, 1995). These exceptions are at a lower system/software level.

Secondly, it is very difficult to define exceptions. Instead of giving precise definitions, we state the types of exceptions that we consider in this thesis. We also try to find out other kinds of exceptions through the research. Similar to (Chiu, 2000) and (Casati, 1998), we classify exceptions into expected and unexpected ones.

We further classify expected exceptions into two types. One is that the result of the workflow is not as expected. This type of exception usually can be resolved if there exist other equivalent and acceptable outcomes. The other type of expected exception is that one or more steps of the workflow fail or do not response. For this type of exceptions, the system can try other steps that have the same results.

We also classify unexpected exceptions in this thesis into two types. The first type is those unexpected exceptions that can be resolved by the workflow system through reasoning and searching. The second type is those that the system does not know what to do and can only be solved by user intervention.
An example for resolving through reasoning is that, if the execution sequence of a workflow instance is unexpected (i.e., not specified in the normal execution pattern nor in the exception handler), but the eventual outcome/result is the same as the normal one, then if the workflow system can deduce this same effect, it should allow the execution and thus provides a mechanism to handle unexpected exceptions. An example for resolving through searching is as follows. If the execution of a workflow instance cannot proceed because of an exception, and if the know-how to resolve it resides somewhere and the workflow system can find it, then the system can resume the execution using this newly found solution.

3.3 Survey of Related Research Work

In this section, we review some of the exception handling mechanisms in related research work.

3.3.1 The Workflow Activity Model

The Workflow Activity Model (WAMO) (Eder and Liebhart, 1995) is a transaction-oriented model that “enables the workflow designer in modeling not only correct business processes but also potential (expectable) exceptions which may arise during process execution”.

WAMO classifies exception into four categories: basic failures, application failures, expected exceptions, and unexpected exceptions. WAMO targets at solving ‘expected exceptions’ by breaking down a complex business process into smaller
work units (activities) which themselves consist of pre-existing tasks and guarantee reliable flow control with automatic exception handling.

A conceptual WAMO architecture consists of activities, forms and agents. Activities represent any abstract description of work units in a business process; Forms are data containers to store relevant process data; Agents are responsible for execution of activities. In addition, the WAMO model defines some transaction specific parameters for each task. The parameters specify how a task behaves when compensation is needed, which in turn decides how an exception should be handled. There are four possibilities:

- None: the committed task does not need to be compensated;
- Undoable: the committed task can be undone by the corresponding compensation task without any side-effects;
- Compensable: the committed task can be semantically undone by the corresponding compensation task but there are side-effects;
- Critical: the task cannot be undone or compensated afterwards because there is no compensation task to undo the already committed effects.

Compensation conception is one of the important features of WAMO. When a vital activity fails, a compensation is initiated. The compensation and exception handling mechanism is shown in Figure 3-1 as event-state change of activities. The general solution is to compensate all previous successfully committed activities in the current control structure in reverse order.
Once all relevant activities in the current control structure are compensated, the parent activity of the current control structure changes its state to committed unsuccessfully. Then the system can make forward progress or start a new compensation process on the next higher level.

**Discussion**

The WAMO model is transaction oriented and targeted at handling ‘potential (expected) exceptions’. Unexpected exceptions are not covered in the model.

Because of its transactional-oriented nature, the WAMO has the following limitations:

- Not only the ‘normal’ process sequence, but also all of the alternative and compensation processes have to be pre-defined. This requirement of rigid and inflexible predefinition limits the exception handling capability of WAMO.
- It does not capture the reasons for “why?” and “how?” an exception happens.

In general, there is an assumption of static relationship between the original...
execution and its compensation (Du and Elmagarmid, 1997). Because there is no information about the cause of problem, a compensation task may just repeat what the original one has done.

- It only solves failures by providing alternative/compensation tasks. It does not support deviations from workflow schema like, for example, adding or bypassing some steps in the process. These deviations are not ‘failures’ and thus need not be ‘compensated’ for.

### 3.3.2 OPERA Process Support System

The OPERA process support system (Hagen and Alonso, 1998) is a failure handling mechanism made flexible by using a combination of programming language concepts and transaction processing techniques.

In OPERA, a workflow process has a modular structure in which different tasks are nested in a tree. The exception handling mechanism is divided into two parts: exception detection and exception handling. The mechanism is such that, when a task fails, its execution is stopped and the control of the process is handed over to exception handler. If the handler cannot solve the problem, it propagates the exception up to higher level in the task tree. The control flow of exception handling is shown in Figure 3-2.
The exception handler is controlled by the parent process (or the Caller) while the exception is returned by the sub-process (or the Signaler). The exception handler is also a type of process, which can contain arbitrary activities, blocks or sub-processes. As shown in Figure 3-2, the entry point to a handler process is a so-called proxy activity (E_FAILED), which can be seen as a placeholder for the exception that occurred. Predicates such as P1 and P2 are defined on the data that has been passed by the signaler together with the exception. Depending on the information provided by the signaler, either P1 or P2 is called. Thus different execution paths are allowed depending on the information provided. Terminator proxies (RESUME and ABORT) define the endpoints of a handler and determine how the control flow has to proceed after termination of the handler.

**Discussion**

Like WAMO discussed in the previous section, the OPERA process support system is also based on transactional concepts. As such, OPERA has the same limitations as WAMO, in that all exceptions need to be pre-defined, therefore it cannot handle unexpected exceptions, etc. If one undefined exception occurs, the whole process fails. For each task without exceptions being defined, the system
default handler will take control. However, this 'catch-all' default handler only aborts the signaler and propagates all exceptions to the caller, without doing 'real handling' to find a solution.

Finally, recursive handler calls are not allowed (Hagen and Alonso, 1998), which means that the handler is the 'last stop' for solving an exception. There is no way to solve any exception the handler itself may encounter. Thus the problem-solving capability of the handler is limited.

3.3.3 ADOME-WFMS

The ADOME-WFMS (Chiu, 2000) is an exception-centric workflow management system framework proposed by Chiu in his PhD dissertation. It uses an event-driven approach designed to coordinate workflow execution and to handle exceptions.

An ADOME-WFMS model consists of the following sub-models:

- Organization and Resource Model, which specifies: 1). organizational composition such as subsidiaries, divisions, departments, employees and customers etc. 2). Problem Solving Agent (PSA) objects, attributes and their methods. 3). token/role multiple-inheritance, which includes capability tokens, roles and positions that a PSA object can possess.

- Match Making Model, which enables the workflow system to find a PSA that is capable of performing a specific task.
• Activity Model, which recursively decomposes an activity (workflow) into sub-activities and eventually tasks. Dependencies between activities are also defined in this model.

• Event-Driven Execution Model, which provides a unified approach for normal activity execution and exception handling.

The ADOME-WFMS uses a centralized control and coordination execution model on the Activity Executor, which initiates the problem solving agents (PSA) to carry out assigned tasks. PSA is selected by Match Maker. If a task raises an exception event, the Exception Manager will take control. An event can be any of the following: data operation, workflow, clock time, external notification, or abstract event.

In ADOME-WFMS, the exception handling steps carried out by the Exception Manager upon a defined exception event are as follows: 1.) Perform notification of relevant PSAs if necessary. 2.) Identify and execute appropriate exception handlers, which are modeled as sub-activities. There are several types of handlers such as Mandatory Event-Condition-Action (ECA) handlers, Procedural handlers, and ECA handlers. For example, a mandatory ECA rule specifies that all exceptions should notify the purchasing manager in a procurement activity; a procedural handler specifies extra activity branches. 3.) If no exception handler is found, the Human Intervention Manager will be invoked and it is up to user to decide how the exception should be solved. 4.) Implement rollback if necessary. 5.) If necessary, exceptions are propagated to parent activity for further handling.
The exception handler resolution of ADOME-WFMS is similar to that of OPERA does. Instead of defining each process, ADOME-WFMS models exception handlers in the form of ECA (Event-Condition-Action) rules that can be bound to selected objects dynamically at run-time.

**Discussion**

The ADOME-WFMS employs a meta-modeling approach to support exception handling. It also gives an interesting exception-centric view of the workflow model. In comparison with other workflow models, it has the following limitations:

- Although ADOME-WFMS provides a dynamic binding mechanism to reuse exception handlers, it could be difficult to make an exception handler ‘reusable’ (e.g. one-to-many binding to workflow instances). Since each exception could have different causes and characteristics, the handler for one exception may not work for others.

- ADOME-WFMS focuses on solving ‘expected exceptions’. If unexpected exceptions occur, human intervention is usually required although the system may provide some assistance, e.g. finding handlers. In this regard, ADOME-WFMS still requires that exceptions be pre-defined and only those ‘repeatable’ or ‘replaceable’ tasks can be handled. If a ‘critical’ task fails, the parent activity (thus the whole workflow process) fails.

- ADOME-WFMS uses an activity executor and an exception manager to control normal process execution and exception handling, respectively. This approach incurs the extra expense of transformation of task data between the
activity executor and the exception manager. It can be argued that there is no reason for putting a boundary between regular process execution and exception handling. This would remove the possibility of considering exceptions that happen all the time during regular execution.

Nevertheless, ADOME-WFMS is one of the most comprehensive frameworks for exception handling noted by the author. In a later part of this thesis, we provide a relatively more detailed comparison of the ADOME-WFMS and our proposed MOAP-Wf by using a business process example.

### 3.3.4 CapBasED-AMS

The Capability-Based and Event-Driven Activity Management System (CapBasED-AMS) (Karlapalem et al., 1995) proposes a framework with capability based activity specification, decomposition and identification of the problem solving agent to execute the tasks, and event-driven execution of the activities.

![Figure 3-3: Architecture of CapBasED-AMS (adapted from (Karlapalem et al 1995))]
The architecture of a CapBasED-AMS is shown in Figure 3-4. A problem solving agent (PSA) is either a piece of hardware, a software system or a human, with a set of abilities to execute tasks. An activity consists of one or more tasks which can be executed by one or more PSAs. With capability-based activity specification and decomposition, each task is matched to a PSA by selecting a PSA that has the abilities to meet the needs of the task.

Within an activity are multiple inter-dependent tasks that need to be coordinated, scheduled and executed. All of these inter-dependencies are expressed by means of a uniform framework of events. The execution of a task is orchestrated by the occurrence of the events. Upon predefined events, the activity coordinator initiates and controls the execution of tasks by using mapping between events and tasks.

Two types of exceptions are supported in CapBasED-AMS: 1). Task execution failure. Upon such failures, the activity coordinator will initiate a predefined recovery and contingency plan to complete the execution of an activity. 2). Undefined event. Those events that are not predefined in the ECA rules will be added during runtime after validating the triggered actions by users.

Discussion

PSA in CapBasED-AMS is modeled as an individual unit in a flat structure. This type of flat structure is not flexible in the sense that some processes may need to be executed by a group or department instead of individuals.

Generally speaking, exception handling in CapBasED-AMS is weakly supported. It only supports predefined recovery and contingency plans upon failures of task
execution. As the authors of this system note, “[it] has no inherent ability to reason about the nature of the problem and solve it. It solves a certain class of problems which the system has the knowledge to solve. Also, the problem decomposition knowledge and coordination strategy are a priori defined”.

### 3.3.5 Adaptive Workflow

Han, Sheth, and Bussler propose a two-layer adaptive workflow system as shown in Figure 3-4. In the upper layer, a workflow model, there are abstract business tasks. The lower layer, a resource level, includes a workflow model with versions, applications and persons. Tasks are executed by binding different workflow resources. This allows a dynamic binding between a business task and various potential implementations (*Task Functionality* vs. *Workflow Model with Versions* in Figure 3-4). The ownership of task is resolved at runtime by concrete users who become runtime task owners (*Task/Process Owner* vs. *Persons* in Figure 3-4).

![Figure 3-4 Synthesized Adaptive Workflow Approach (adapted from (Han et al 1998))](image)

As the authors of this system note, the proposed approach has the following features to manage workflow changes:
• Business processes are hierarchically organized but not totally fixed. A business task is abstracted by separating its declaration from its implementation. The binding between a business task and various potential implementations are dynamic instead of fixed.

• Exception handling is enabled by embedding the generic resource binding and adaptation handling mechanism into each task. A user can try different variants (of a task) to retry, postpone, or abort the task.

Discussion

Although this approach provides a way to bind tasks to workflow model versions dynamically, how the tasks are actually bound is not discussed. Because the tasks are abstract ones, they need to be ‘concretized’ in order to be adapted to workflow model.

Secondly, because the task variants and workflow model versions still need to be defined a priori, the number of possible choices is fixed and the capability of the system to handle exceptions is also limited. There is no mechanism for deciding under what circumstances which task and which workflow model should be executed. The only way to accomplish this is by having users try those variants one by one.

3.4 Summary of Survey

In this section, we reviewed existing research work on workflow exception handling. We do not include commercial products because very few of them provide support for managing exception situations (Casati et al., 1999).
In summary, most of the research work and system prototypes try to solve workflow exceptions at a system level by concentrating on extending database transaction process techniques. One example is use of the recovery capability of (database) environment on the top of which the WfMS is executed (Casati, 1998). One common limitation is that most of them target expected exceptions, for example by predefining such exceptions and providing 'compensation' processes, while ignoring or at least paying less attention to 'unexpected', 'unknown' or 'undetectable' exceptions.

Furthermore, most of the surveyed works are traditional activity-based models, of which the process/activity/task must be predefined in rigid sequence during build time. From an exception handling point of view, a typical workflow model consists of 'core processes' and 'exception processes'. The latter is used to solve any problems the workflow system encounters. Examples of the surveyed research work that apply this approach include: WAMO, OPERA, CapBasED-AMS and ADOME-WFMS. Both the 'core' and 'exception' processes have to be defined a priori. Changes during runtime are usually not allowed. This rigid, inflexible and sequential feature limits the capability of the exception handling of the systems. A cooperative process cannot be executed as a flowing sequence of activities, but contain many unpredictable exchanges that cannot be modeled in advance (Grigori et al., 1996). Another important consideration is that the workflow system should be able to 'learn' from its experience of handling exceptions and then use such knowledge to avoid exceptions.
Because one of the common assumptions most workflow research makes is the availability of pre-specified workflow definition (Du and Elmagarmid, 1997), few of the surveyed research works focus on dynamic workflow modeling. We believe that building a flexible, dynamic workflow model in the first place is much easier than solving problems afterwards. Dynamic workflow systems have no pre-specified process specifications (Du and Elmagarmid, 1997). The workflow starts with some initial activities. When an activity is completed, new activities will be selected according to the execution status and results of the current activity.

The dynamic workflow is different from ‘dynamic modification’ of existing process definitions (Du and Elmagarmid, 1997). With dynamic modification approach, the system tries to ‘modify’ pre-defined processes when encountering exceptions. Then the running process is either restarted or migrated to new definitions. Examples of this dynamic modification approach include: Han et al (1998) and van der Aalst et al (1999). In this thesis, our approach does not handle dynamic modification.

<table>
<thead>
<tr>
<th>Category of Approach</th>
<th>Examples of research work</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Core Process + Exception Process</td>
<td>WAMO, OPERA, CapBasED-AMS, ADOME-WFMS</td>
<td>Activity-based model (pre-defined sequence)</td>
</tr>
<tr>
<td>Dynamic Modification</td>
<td>(Han et al., 1998), (van der Aalst et al., 1999), (Heinl et al., 1999)</td>
<td>Migration of running instances - Activity-based model (pre-defined sequence)</td>
</tr>
</tbody>
</table>
Lastly, to our best knowledge, few of the research works take into consideration the organizational and social aspects of workflow including, for example, how a user's goals affect workflow process execution.

A summary of categorization of the surveyed research works is shown in Table 3-1.

3.5 Different Views of Workflow

Generally speaking, there are two views of a business: an enterprise view and a process view (Wand and Woo, 2001). The enterprise view focuses on 'actors' or 'roles', while the process view focuses on 'activities' – how a process is comprised of a series of actions. Most of the research works that we surveyed are from only a process view, i.e. linking a number of sequenced activities to form a workflow process. As we found in our survey, this approach is not capable of efficient and effective exception handling.

We take an enterprise view of workflow on actors and communications because of the following factors and requirements:

- Organization and its environment can be viewed as composed of interacting agents and all actions in the organization are done either by agents or by interactions between agents who exchange information and materials (Wand and Woo, 1999). Different groups (and agents) with the organization are relatively autonomous (Jennings et al, 1996).
• Given that workflow systems operate in an organizational context, it is extremely important that they must be 'organizational aware' for the successful application of workflow technology (Basu and Kumar, 2002).

• Within organizations, there is a decentralized ownership of the tasks, information and resources involved in business (Jennings et al, 1996). Each entity may be motivated to control its own resources and activities, and may not accede control to its partners (Basu and Kumar, 2002).

• Business processes are highly dynamic and unpredictable and any detailed time plans are often by unavoidable delays or unanticipated events (Jennings et al, 1996).

Arguably, a single 'process view' of workflow is not sufficient. We found that Micro-Organization Activity Processor (MOAP) (Woo and Lochovsky, 1991) and Object-Oriented Workflow Model (OOWM) (Hui, 1997) can be extended to address the above requirements. MOAP is decentralized with problem-solving ability while OOWM is a workflow in an autonomous business environment. The proposed approach is agent-based and it not only supports autonomy of agents (enterprise view) but also allows processes emerge out of the interactions of agents (process view). It should be noted that there can be also other approaches that satisfies the requirements. However, choosing the best approach is beyond the scope of this thesis.

In the next section, we briefly introduce OOWM and MOAP and examine how their features can satisfy the requirements of enterprise view and facilitate workflow exception handling.
3.6 Micro-Organization Activity Processor (MOAP)

3.6.1 The Structure of MOAP

MOAP is a mobile agent architecture for modeling intelligent information systems. One of the most important features of the MOAP structure is its support of communication of knowledge between the agents. From the MOAP point of view, an organization is composed of many small units (as 'micro-organization') that can be departments or employees (users). For each of the units, an 'activity-processor' is assigned to perform certain functions. One MOAP is attached to only one user and each user has at least one MOAP.

A MOAP structure consists of six different types of entities, including: Data Repository, Procedural Knowledge, Problem-Solving Knowledge, Activity Coordinator, Activity Agent, and Workspace as shown in Figure 3-5.

![Figure 3-5 Micro-Organization Activity Processor (adopted from (Woo and Lochovsky 1991))](image-url)
Each of the components plays different roles in knowledge representation and communication. The Data Repository stores persistent information. Procedural Knowledge is well-organized, well-tested organizational knowledge, which is described in terms of tasks (and subtasks). Problem-Solving Knowledge consists of the knowledge related to facts, the missing parts of procedural knowledge, or that which is playing the role of procedural knowledge. Workspace is the communication medium for all of the active entities. The Activity Agent represents the MOAP and specializes in communication and coordination with other MOAPs. The Activity Coordinator controls the execution of tasks, the activity agent and use of problem-solving knowledge, and responds to inquiries by other MOAPs.

### 3.6.2 Applicability of MOAP in workflow systems

Because of its intelligent and autonomous agent characteristics, MOAP is an ideal building unit for a decentralized computing environment. One of the examples is OASIS (Martens and Woo, 1997), which is an implementation of a MOAP architecture. In the implementation, each MOAP has the ability to communicate and cooperate with other MOAPs to complete a task by utilizing their procedural or problem-solving knowledge.

Intelligent agents have been recognized as a new solution to manage workflow, or automated business processes. For example, agents are used to improve the management of business processes under control of a workflow management system (termed as ‘agent-enhanced workflow’) (Judge et al., 1998). Another research effort is ADEPT (Advanced Decision Environment for Process Tasks) (Jennings et al.,
1996; Norman et al., 1996), in which a business organization structure is modeled as a multi-agent system (termed as ‘agent-based business process management system’).

In summary, agents including MOAP can manage business processes better than the traditional activity-based workflow systems (Jennings et al., 1996). MOAP can satisfy the enterprise view requirements as we discussed in previous section in that:

- MOAP is decentralized and distributed and it supports decentralized ownership of the tasks, information and resources involved in the business;
- MOAP is modeled as role and organization unit. Such model meets the requirement of ‘organization-aware’ and helps alleviate the issue that most of today’s workflow management systems focus on the process dimension (i.e. routing, control flow) and oversimplify the organizational dimension (Basu and Kumar, 2002).
- MOAP has problem-solving ability that helps handle exceptions in a workflow.

3.6.3 Comparison of MOAP and ADEPT agents

In addition to general properties of agents such as autonomy, social ability, responsiveness and being proactive, a MOAP has differential features of explicitly defined procedural knowledge and problem-solving knowledge, and support of knowledge acquisition and communication. Before we examine, in later chapters, how these features could be incorporated and utilized in our workflow model to
support exception handling, we briefly review one of the agent types – ADEPT and compare it with MOAP.

![ADEPT Agent Architecture (adapted from Jennings et al 1998)](image)

The ADEPT agent architecture is shown in Figure 3-6. An ADEPT system may consist of a mixture of hierarchies of agencies and peers reflecting organizational structures. An agency is recursively defined with a single responsible agent, a set of tasks that the responsible agent can perform, and a set of sub-agencies. The responsible agent represents the agency and is in charge of communication with peer agencies. The responsible agent is comprised of several modules. Self and Acquaintance Modules store available resources, current schedule of activity, and capabilities of other agents. The Situation Assessment Module (SAM) maintains the agent’s schedule of activity. The Service Execution Module (SEM) is responsible for the initialization and management of the execution of services. The Interaction Management Module negotiates for services that are required from other agents.
In comparison with MOAP, the ADEPT agent architecture has the following limitations:

First, the ADEPT agent architecture focuses on inter-agent communication and negotiation. It puts more emphasis on communication mechanisms, message descriptions, and 'joint-decision making' processes. It lacks the representation of knowledge required for business process management. How tasks are organized to support process execution is not addressed. In contrast, the MOAP architecture focuses on agent knowledge representation. It explicitly differentiates procedural knowledge and problem solving knowledge. Thus each MOAP has the flexibility to use different knowledge to solve different problems. With an emphasis on process knowledge, the MOAP architecture captures 'what', 'when' and 'how' tasks are performed rather than, as in the case of as ADEPT, tracking 'how' communicating occurs between agents.

Secondly, ADEPT takes a 'sub-agency' approach to reflect organizational structure. This approach puts certain restrictions on the agent's autonomy in that sub-agents can not communicate with other agents outside the scope of the current agent. All communications must be done through the responsible agent. For example, suppose there is a situation where a purchaser in the purchasing department needs to communicate with an accountant in the finance department. The communication must be done as 'purchaser → purchasing manager → finance manager → accountant' because both purchaser and accountant are modeled as 'sub-agents' within the purchasing department and the finance department respectively. This is not, usually
the case in business. There is no real business reason to explain why the purchaser and accountant cannot interact directly.

3.7 Objected-Oriented Workflow Model (OOWM)

Originally developed in the software engineering area, the object-oriented approach is considered to be a more natural technology for the requirements of autonomy, distribution and cooperation, because objects are a natural model of interactive components in a distributed environment (Kappel et al., 2000).

Among those studies on the object-oriented approach, Hui’s OOWM extends OOEM (Object-Oriented Enterprise Modeling) (Wand and Woo, 1999) to capture the informational, functional and organizational aspects of a process and the architecture can be very flexible and adaptable to fit into a constantly changing business environment (Hui, 1997).

In this section, we give a brief introduction of OOWM structure.
As shown in Figure 3-7, OOWM consists of six constructs: Object, Attribute, Service, Request, Activity and Business Rule. An object is defined as a model of a substantial thing in the problem domain that interacts with other objects. An object can represent an organizational unit, a division, a department, or a role. An attribute is the state of and must belong to an object. Service models a state transformation of an object. It comprises a series of actions performed by an object with the purpose of satisfying a request, which is an interaction between two objects. An activity is the basic unit of operation for an object. Activities form services. A Business Rule can be interpreted as pre-conditions and termination conditions for an activity.

An Object Activity Template (OAT) is used to capture workflow information that includes activities and business rules that govern the activities. The OAT is shown in the following table:
<table>
<thead>
<tr>
<th>Interface Attributes</th>
<th>Internal Attributes</th>
<th>Services</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Service 1</td>
</tr>
<tr>
<td>Returning Interface Attributes</td>
<td>Internal Attribute To Support Service 1</td>
<td>Access Code</td>
</tr>
<tr>
<td>Pre-condition 1 Activity 1</td>
<td>Condition 1</td>
<td>Request generated from activity 1</td>
</tr>
<tr>
<td>Pre-condition 2 Activity 2</td>
<td>Condition 2</td>
<td>Request generated from activity 2</td>
</tr>
</tbody>
</table>

Each row of the table represents an activity that is attached to pre- and termination conditions. If a pre-condition of one activity holds, an object will perform that activity. Similarly, if a termination condition is true while an activity is being performed, the activity will stop.

One important object of OOWM is the Controller, which controls and monitors what objects should react to requests and when. The functions of the Controller are to ensure that business rules be followed in an automated workflow environment by not only keeping track of, but also the evaluating of rules based on the information carried by a request and the state of a process.

We adopt some concepts of OOWM based on the following considerations:

First, by extending OOEM methodology, OOWM reflects a ‘natural view’ of organizational processes in an object-oriented context (Hui, 1997).
Secondly, with the introduction of the Controller object, OOWM has the ability to separate what needs to be controlled and what does not. Such ‘separation of concern’ strategy ensures the autonomy and flexibility of each object. Another advantage of the Controller object is that it represents the owner of a business process and helps keep track of the overall status of each process.

Thirdly, OOWM meets the demand for autonomy by decentralized units and allows locally operational autonomy while enforcing policies at the corporate level (Hui, 1997).

Lastly, OOWM separates business rules from activity definitions. This approach can improve the flexibility and capability of exception handling in that a change of business rules does not require the change of tasks and activities.

3.8 Summary

In this chapter, we examined the characteristics and classification of workflow exceptions. We then reviewed some of the research on exception handling. Advantages and limitations of the surveyed approaches were also discussed. Finally we introduced two research approaches, MOAP and OOWM, which constitute the theoretical foundations of this thesis.
4 MOAP-Based Workflow Model

In this chapter we introduce our proposal of MOAP-Based Workflow Model (MOAP-Wf). To demonstrate the MOAP-Wf concept, we use a simple example in a business context. This example describes a computer purchase process:

*The purchase process is initiated by a user who submits a purchase requisition (PR) to his/her Business Departmental manager. After the department manager approves it, the PR is passed to the purchasing department, which then issues a purchase order (PO) to the selected vendor.*

*After the computer is received, the purchasing department sends it to the user for quantity and quality verification. Upon verification, the purchasing department then passes invoices to the finance department for payment. The whole purchase process is completed when payment is made.*

4.1 Modified MOAP Architecture

4.1.1 Overview

MOAP is the main construct in our workflow system architecture. To fit the fully distributed MOAP into a workflow context and integrate it with the OOEM Object concept, we made some modifications to its architecture. The modified MOAP architecture is shown in Figure 4-1.
Figure 4-1: Modified Micro-Organization Activity Processor (adapted from (Woo and Lochovsky 1991) with modifications)

Like its original architecture, this modified MOAP has the following types of entities: data repository, procedural knowledge, problem-solving knowledge, workspace, activity agent, and activity coordinator. In addition to these entities, two new ones are proposed: Goal and Service. (These two constructs are explained in more detail in the next section.)

One MOAP is always attached to a role/responsibility, which is held/implemented by a user or organizational unit (OU) (Woo and Lochovsky, 1991). The separation between activities and agents (i.e. user or actor) performing them makes it easy to cope with personnel changes (Kappel et al., 2000). Consider the purchasing process example. The Business Department and Finance Department are OU MOAPs. The Employee, Departmental Manager, and Corporate Accountant are represented as User MOAPs.
4.1.2 Goal

A goal is what a MOAP targets to achieve, and can be treated as the MOAP’s intention and the ‘reasons’ underlying work activity which answer the ‘Why’ and ‘What-If’ questions (Yu and Mylopoulos, 1994) (i.e., why we need the process in the first place, what if one of the steps in a process is eliminated.)

We propose that the goal construct be included in MOAP because of the following observations (Kueng and Kawalek, 1996) in business process redesign:

- Human action is driven by goals.

- We need to be able to state what we want to achieve so that we are then able to define the necessary activities that a business process should encompass.

- A clear understanding of goal is essential in the management of the process of selecting the best design alternative.

- A clear understanding of goals is essential for it to be possible to evaluate the operating quality of a business process.

- A clear expression of goals makes it easier to comprehend the organizational changes that must accompany a business process redesign.

Another interesting point is that people’s inability to achieve goals helps explain why people generate exceptions and thus how processes emerge (Katzenstein and Javier Lerch, 2000). Thus including ‘goal’ in the MOAP model would also help identify exceptions and facilitate workflow evolutions. Exceptions can occur not only
because people do not do thing right, but also because people are not able to accomplish goals.

The goal in a MOAP structure serves a number of purposes:

- As the intention and motivation of the MOAP or the user/OU it represents;
- As the guide and target to design services and workflow activities;
- As the criteria for the handling of other MOAP requests;
- As the criteria to define exceptions and workflow evolution;

In an organization context, there are different levels of goals, e.g. organization-wide goals, OU-level goals, and individual goals. It also needs to be noted that there could be conflicts between the goals of different OUs and individual MOAPs. How to model and how to solve conflicts are beyond the scope of this thesis. We assume that goal conflicts, if any, are already solved.

In the above purchasing process example, the organization-wide goal of the company could be increasing net value for shareholders, the Business Department’s goals could be building good relationship with customers or promoting sales revenue, and the Finance Department’s goals could be balancing the company’s cash flow and making payment in two business days.

4.1.2.1 Representation of Goal

From a teleological point of view, a goal is a collection of future states of affairs, where state is a set of variables and values that those variables have at a given instant.
(Weir, 1984). Similar, and albeit more specific, definitions can be found in business process modeling and artificial intelligence literature. For example, a goal is ‘an end that needs to be achieved, for the sake of meeting some customer requirements (Narendra, 2000)’. A goal can also be specified as ‘the desired state of the process domain’ (Craven and Mahling, 1995).

Since this thesis tries to show how the proposed approach (which may not be the best one) can handle exceptions, the details of modeling goals and the detail representation of goals are, therefore, not needed to demonstrate the approach. For the purpose of this thesis, we give an informal definition of goal as a set of future states of business objects that a MOAP is going to achieve by executing a set of tasks or requesting services from other peer MOAPs. We use a simple EBNF-style notation to represent goals:

```ebnf
goal ::= "GOAL"
    <state>
    <bizObject>
    "END.GOAL"
```

For example, a goal of customer service could be ‘purchase orders handled’, which can be represented as:

```
GOAL
    STATE Handled;
    BIZOBJECT PurchaseOrders;
    END_GOAL
```

The implementation of `<state>` is simplified as a word and full matching of the word satisfies the goal. A goal must be refined and decomposed until it is clear what specific actions must be performed to achieve the specified goal (Craven and Mahling, 1995). In our MOAP-Wf, goal is decomposed and organized in a goal-tree:
the higher level goals can be achieved by achieving lower level sub-goals. If a sub-goal can not be further decomposed, we call it 'leaf-goal', which can be achieved either by a specific task or by service by other MOAPs or users.

The following figure shows a general goal-tree diagram:

![Goal Tree Diagram](image)

**Figure 4-2 Goal Tree Diagram**

There are two basic types of relationships between sub-goals: AND-relationship and OR-relationship. We assume that these two relationships are sufficient for demonstrating the proposed approach in this thesis and do not consider more complicated ones such as Exclusive OR (XOR) and N-out-of-M, which can be composed by the two basic relation types in some 'workaround' ways such as selection of transition conditions (van der Aalst, 2002).

An AND-relationship means that all the sub-goals should be achieved in order to achieve the parent goal. For an OR-relationship, achieving one of the sub-goals suffices the achievement of parent goal. As shown in Figure 4-3, for the purchase process example, all the sub-goals of Approved(PR), Purchased(Item) and Paid(invoice) should be achieved to successfully complete the whole process (achieving the top-level goal Supplied(Item)).
We assume that a goal tree can always be normalized. A normalized goal decomposition tree has the following characteristics (Li et al., 2001):

- Logic relationship exists and only exists among direct sub-goals of the same (parent) goal.

- Homogeneity of logic relationship in the goal sets of the same level, i.e. there is only one type of relationship at the same level.

- Heterogeneity of logic relationship in goal sets of adjacent levels goal tree.

For example, if $G_0 = (G_1 \text{ AND } G_2) \text{ OR } (G_3 \text{ AND } G_4)$, then the decomposition will be normalized as $G_0 = Ga \text{ OR } Gb$, where $Ga = G_1 \text{ AND } G_2\text{, }Gb = G_3 \text{ AND } G_4$.

### 4.1.3 Service

Service is the state transformation of an object from an ontological point of view. A Service comprises a series of actions performed by an object with the purpose of satisfying a request (Hui, 1997; Wand and Woo, 1999). For example, a purchasing staff handles 'purchase requisition', a corporate accountant handles 'request for payment'.
The relationship among goal, service, procedural knowledge (activities) and problem-solving knowledge are depicted as: first, services are guided by goals. How one service is defined and provided should comply with the related goal. We assume that it is a one-to-one relationship, i.e. one goal guides one service and one service ‘serves’ exactly one goal. The second assumption is that one service is implemented by at least one task (activity, function) in the procedural knowledge, or by services from other MOAPs. Lastly, services use problem-solving knowledge if necessary. One of the main components of problem-solving knowledge is business rules (cf. section 1.5).

Service is also the interface or channel for inter-MOAP communications, e.g. receiving requests and delivering results. To publish and advertise its service, a MOAP:

- Sets the conditions of requests that other requesting MOAPs must follow. Some examples of the conditions include data required, who can send such requests and validation rules or constraints such as data/time etc.

- Advertises with other MOAPs its capability, which specifies what the MOAP can deliver. The capability can be loosely defined as a replication of goal. One difference is that goal is internal to MOAP itself.

From an OOEM point of view, such communication is done via the changing of interface attributes (Wand and Woo, 1999). A MOAP’s capability and the set of conditions can be viewed as ‘interface attributes’, which are accessible by other MOAPs.
4.1.4 Procedural Knowledge

Procedural knowledge is well-thought-out, well-organized, well-tested and well-adapted (Woo and Lochofsky, 1991), and can be regarded as 'default' execution of a workflow process. Procedural knowledge can described in terms of task/subtask hierarchy tree and the tasks should be able to be executed successfully in a specified sequence without encountering unexpected circumstances (Woo and Lochofsky, 1991).

In our MOAP-Wf model, we extend the procedural knowledge by including some other components and giving a concrete description of the original ones. The components of procedural knowledge include: task, pre-set of sub-goals, and MOAP selection.

4.1.4.1 Tasks

Task (or Activity) is a piece of work that forms one logical step within a process. Typically task is the smallest unit work scheduled during a workflow process enactment (WfMC, 1999). A similar definition is given by that tasks are the elementary or atomic work units that collectively achieve the workflow goal (Casati, 1998). We follow the concept of task atomicity that a task is the smallest unit and is to be executed to achieve one leaf-goal in the above goal hierarchy tree.

Core components of a task include pre-conditions, post-conditions and functions (Woo and Lochofsky, 1991).
Pre- and post-conditions govern when a task should be started, completed or terminated. More specifically, Pre-conditions determine under which circumstances the task should be executed (Woo and Lochovsky, 1991). Post-conditions describe those things that must be true after the activity is executed (Woo and Lochovsky, 1991). Termination conditions determine the circumstances under which an activity should be stopped during execution.

Pre-conditions can be loosely matched to the ‘input’ of IPO (input-process-output) model, and post-conditions to ‘output’. In this thesis, we use the same format as that of goal to describe post-conditions, i.e. business object with certain state. A task is capable of achieving a certain goal when one of the post-conditions matches the goal, i.e. the task leads to a ‘state’ that is the same as the state described by the goal. For the purchase process example, one pre-condition of task ‘make payment’ could be ‘Is_Sufficient(Funds)’, and one post-condition could be ‘paid(invoice)’. Obviously, this task can achieve the goal ‘paid(invoice)’.

A minimum representation of task can be shown as:

```plaintext
<task>::= 'TASK"
    <Pre-conditions>
    <Post-conditions>
    <Functions>
    "END_TASK"
```

### 4.1.4.2 Pre-Set of Sub-Goals

Pre-set of sub-goals can be regarded as ‘default’ ones that the MOAP is going to achieve under normal circumstances, i.e. when there is no possible failure or exception. This is also the ‘first try’ of the MOAP. Only when encountering failure or
exceptions, will MOAP try other alternatives, e.g. other sub-goals with OR-relationship, by using problem-solving knowledge (cf. next section).

For example, the pre-set of goals of the purchase process controller MOAP could be:

```
PRE_SET_SUBGOALS
MEDGOAL Approved(PR)
MEDGOAL Purchased(Item)
MEDGOAL Paid(invoice)
END_PRE_SET_SUBGOALS
```

Because the decomposition is an AND-relationship, all the sub-goals are required to achieve overall goal of Supplied(Item).

4.1.4.3 Pre-Conditions of Goals

By decomposing goals into sub-goals, we can have a clear picture of what needs to be done before the overall goal can be achieved. Super-goals depend on sub-goals. This dependency may also exist between sub-goals. One sub-goal may be achievable only if some other sub-goals have already been achieved.

For a goal dependency, if Goal1 depends on Goal2, Goal2 becomes Goal1's pre-condition, which must be a ‘true state’ before Goal1 can be achieved. In the purchase process example, the dependencies can be shown as:

```
Purchased(Item): has_precondition(Approved(PR))
Paid(invoice): has_precondition(Purchased(Item))
```

These dependencies/pre-conditions can also lead to goal ordering – the sequence of goals that a MOAP must follow when pursuing the goals. In the above example,
the ordering of goals can be inferred as: Approved(PR) → Purchased(Item) → Paid(invoice).

4.1.4.4 MOAP selection

Similar to the Agent Selection rule in Kappel et al (2000), MOAP Selection rules determine which sub-goal should be executed by which MOAP and how the candidate MOAP should be decided.

The selection is capability-based, i.e. a MOAP is selected according to its 'abilities' to execute tasks or to achieve certain sub-goals. As we discussed in the previous section, we assume that a MOAP’s capabilities are published and made available to other peer MOAPs through its service advertising.

For the purchase process example, the sub-goal Approved(PR) can be achieved by the Business Department Manager; sub-goal Purchased(Item) can be achieved by the Purchasing Department; and Paid(invoice) is to be achieved by the Finance Department.

4.1.5 Problem-Solving Knowledge

Problem-solving knowledge provides assistance when the procedural knowledge is insufficient to perform an organizational activity (Woo and Lochofsky, 1991). It describes a way (or ways) for solving a particular class of problems (Wielinga et al., 1993).

In our MOAP-Wf, we group problem-solving knowledge into the following categories:
4.1.5.1 Goal Decomposition

As we discussed in previous sections, each MOAP has its own goal that can be decomposed into sub-goals. The top level goal is a separate component of the MOAP, while in procedural knowledge there is a pre-set of default sub-goals that the MOAP will initially try to achieve.

In addition to the top level goal and pre-set of sub-goal, the complete goal decomposition hierarchical tree is stored in problem-solving knowledge. The notion is that the sub-goals other than the default pre-set ones are only needed when there is failure or exception, which makes the top level goal unachievable.

New sub-goals can be added to the hierarchical tree if necessary when the MOAP tries to solve any failure or exception.

4.1.5.2 Problem-Solving Tasks

Problem-solving tasks are those tasks that are not included in procedural knowledge. These tasks are executed to achieve those sub-goals other than the pre-set ones. Another type of task in this category is that has not yet been approved to be stable (Woo and Lochofsky, 1991) to achieve default sub-goals in procedural knowledge.

A problem-solving task is defined in the same way as one in procedural knowledge using the traditional IPO model.
4.1.5.3 Business Rules

Business rules articulate the policies that guide a business process in its effort to achieve its goals (Keddara, 1999). Business rules usually determine who should do what tasks, how, and when the tasks should be done. We distinguish business rules from other components because business rules change frequently over time and modeling business rules separately makes it easier and more flexible to cope with such changes.

Based on their functional scope, business rules can be classified into three categories according to their 'ruling' scope: organizational-wide rules, process-level rules, and individual MOAP-level rules.

- **Organization-wide rules** are those that should be observed by all organizational units, processes, and individuals.

- **Process-level rules** are local to process and govern the process’ execution.

- **MOAP-level rules** are local to individual MOAPs and govern how a MOAP handles tasks.

In general, the above 3-level business rules support inheritance: process-level rules inherit from organization-level rules and individual MOAP-level rules inherit from process-level rules. In this way, changes of lower level rules do not affect upper level rules while changes of upper level rules take effect in all lower level components.
The business rules in problem-solving knowledge deal with those conditions and MOAP section criteria that deviate from that of procedural knowledge. We identify two types of business rules according to their ‘ruling’ target:

- **Prioritization of Sub-Goals**: If the default sub-goal in procedural knowledge cannot be achieved, the business rules decide which sub-goal in problem-solving knowledge should be tried, and if there are more than two sub-goals in an OR-relationship, which sequence of sub-goals should be selected.

- **Pre-Conditions of Sub-Goals**: this type is identical to that of procedural knowledge. Pre-conditions define what states should be true before a sub-goal can be achieved.

### 4.1.6 From OOWM Object to MOAP

As stated before, Object-Oriented Enterprise Modeling (OOEM) (Wand and Woo, 1999) is the theoretical foundation of both Object-Oriented Workflow Model (OOWM) (Hui, 1997) and the proposed MOAP-based workflow system architecture in this thesis. The modified MOAP structure can be regarded as an extension and enhancement of OOWM Object by incorporating procedure knowledge and communication capability.
A MOAP is differentiated from an OOWM Object with:

- A Goal component;
- Separation of normal tasks from exception execution tasks.
- An activity agent component.

### 4.2 MOAP-based Workflow Model

#### 4.2.1 Overview

The MOAP discussed in Woo and Lochovsky (1991) and this thesis is independent and autonomous, and performs activities according to its own business rules. On the other hand, a workflow process usually involves different participants or agents of an organization. One MOAP may only perform one task and all the MOAPs involved need to communicate and cooperate with each other to complete the process. To bridge the gap between MOAP's fully-distributed nature and workflow process'
requirements of communication and cooperation, we introduce a special type of agent: Controller MOAP, which controls and is the owner of a workflow process. There is a requirement to monitor and manage the overall business process, because although the control and resources of the constituent sub-part are decentralized, there is often a need to place constraints on the entire process (Jennings et al., 1996).

However, there is another issue to be addressed: to what extent the Controller MOAP should be 'controlling'. On one hand, the Controller MOAP is responsible for enforcing the business rules that govern one process and overseeing the execution of the process by other MOAPs. On the other hand, each MOAP decides the course of action independently and autonomously. This independency and autonomy could also facilitate exception handling by isolating and solving exceptions locally within a MOAP.

To address the above issues, we propose a leveled and hierarchical workflow model with decentralized task execution by MOAPs.

4.2.2 MOAP-based Workflow Model

In this section, we introduce the MOAP-based workflow system architecture, which serves as a basis for the exception handling mechanism presented in the next section.

Figure 4-5 illustrates the MOAP-based Workflow Model (MOAP-Wf).
In the MOAP-Wf, a typical workflow will work in the following way:

1. One MOAP or a User triggers the process by sending a request to the Controller MOAP;

2. The Controller MOAP then evaluates business rules, checks its own Procedural Knowledge, and decides the actions required to satisfy the request;

3. The Controller MOAP sends the requests to other MOAPs (e.g. MOAP 1 to MOAP n);

4. After all requests have been replied to and the results have been received, the Controller then sends the final result to the original requesting MOAP.

When executing one step of the process, each individual MOAP can also send requests to other MOAPs, for example MOAP n can request service from MOAP n+1. The MOAP being requested may become the second level Controller.
(Controller MOAP 2 in Figure 4-5), which will be in charge of the second level of the process (or sub-process). Another possibility is that there is no Controller in Level 2 workflow and whatever the MOAP does is not under control (e.g. MOAP \(n+1\) in Figure 4-5). In this way, the whole workflow can be recursively defined.

4.2.2.1 Controller MOAP

We use the Controller MOAP to clearly identify what does and does not need to control for a workflow process. This concept is similar to the 'controller object' in OOWM (Hui, 1997). The Controller MOAP can also be a solution to OOEM’s limitation of ‘not captur[ing] the execution order of work’ (Hui, 1997).

![Diagram](image)

**Figure 4-6: An Example of Business Views**

As shown in Figure 4-6, the Controller object in OOWM can be regarded as from the process point of view: monitoring all the controller requests to and from objects, orchestrating all objects to complete the process. In our proposed workflow architecture, we follow the same concept although the construct and mechanism of orchestration might be different.
More specifically, a Controller MOAP in MOAP-Wf is in charge of:

- Overseeing and controlling the whole process. Its goal reflects the whole process' goal. As we discussed in previous sections, MOAP is a distributed system and we need a mechanism to bridge the gap between MOAP's distributedness and the overall business process objective. By adding the Controller MOAP with a goal representing the goal of the business process, we try to make sure that actions and interactions of MOAPs can achieve the objective of a workflow process and resolve exceptions.

- Evaluating business rules and requesting services from other individual MOAPs. Each individual MOAP handles one step of the process.

- Serving as a 'central communication point' that coordinates among MOAPs.

By requesting services from other MOAPs, the Controller MOAP only specifies when and what the receiving MOAPs need to react to the request. How the services are provided is under the receiving MOAPs' own control. The receiving MOAPs, or service providers, have full autonomy to decide the detailed procedures according to their own business rules.

With decentralized services structure, a workflow system should be more capable of handling exceptions in that:

- Each service-providing MOAP share the responsibility with the Controller MOAP for exception handling. While the Controller has to make sure the
process is completed, each service-providing MOAP is to solve problems if any occur in the process of handling service requests.

- The scope that is affected by exceptions is isolated and limited to local MOAP itself. Thus any problems encountered by one MOAP will have no impact on other MOAPs.

4.2.2.2 Leveled Hierarchy

Another major characteristic of the MOAP-Wf is the leveled hierarchical structure. As we introduced in the previous section, a service-providing MOAP can further request service from other MOAPs. The requesting MOAP itself becomes the second level controller.

Conceptually, the top-level Controller MOAP is an organization-wide MOAP, which deals with only one process – e.g. creating value for shareholders.

In this leveled hierarchy, a Controller MOAP only communicates with those MOAPs of the same level and does not send requests to, monitor or control those of other levels. Thus the independency and autonomy are maintained. The hierarchy also enables multiple levels of abstraction, with higher level MOAPs not caring about the execution details of lower level MOAPs from which they request services.

The leveled hierarchy also enables multiple levels of exception handling. An exception is first handled at the level at which it occurs. If the exception is not solved, the correspondent MOAP generates an error and replies to the MOAP one level up. The latter then tries to solve the problem based on its own rules. This multiple-level
exception handling continues until a solution is found or the topmost level Controller-MOAP is reached.

4.3 Summary

In this chapter, we introduced the MOAP architecture that was modified based on the one proposed in Woo and Lochovsky (1991) in order to meet the requirements of workflow management. We then proposed a MOAP-based Workflow Model (MOAP-Wf) that incorporates the controller concept of OOWM and is organized as a leveled hierarchy to facilitate exception handling.
5 Handling Exceptions in a MOAP-based Workflow System

In the previous chapter, we introduced the MOAP-Based Workflow Model and gave an overview of how a workflow is executed. In this chapter, we further elaborate the detailed Execution Mechanism. Based on the Execution Mechanism, we then explore how exceptions can be handled in the MOAP-based Workflow Model.

In order to illustrate how exceptions are handled in the proposed MOAP-Wf system, we will restate and revise the workflow example in Chapter 4. The example illustrates a computer purchasing process at a company.

The purchase process is initiated by a user who submits a purchase requisition (PR) to his/her Departmental Manager. The Departmental Manager checks to see if the user does have the need for a new computer, if it is within the department's capital expenditure budget, and other constraints, e.g. company policies. After approving the PR, the departmental manager passes it to the Purchasing Dept.

Upon the approved PR, Purchasing Dept first decides if more detailed specification/requirements for the computer are needed. If yes, the Purchasing Dept will send a request to the IT Dept to get such specifications. The Purchasing Dept then prepares and sends a Purchase Order (PO) to a qualified vendor. Depending on the
availability of vendors and products, the Purchasing Dept might need to initiate a 'vendor selection and approval' process.

When the computer arrives, the Purchasing Dept requests that IT Dept verify quality and/or quantity. After the verification process, the computer is handed over to the user and the Purchasing Dept sends a payment request to the Finance Dept. After an Accountant checking the required authorization, payment terms and conditions set in the PO, and verifies the completeness of required documents, the Cashier of the Finance Dept. makes final payment to the vendor and the purchase process is completed.

5.1 The Execution Mechanism

5.1.1 The Controller and Execution Schema

A Controller MOAP starts to execute a process when it receives a request from a MOAP or a User. Figure 5-1 shows the execution mechanism. Usually the Activity Agent of the requesting MOAP puts its request in to the Controller MOAP’s Workspace and the Activity Coordinator of Controller MOAP will pick up and handle the request (Woo and Lochovsky, 1991). (The workspace component is not shown in Figure 5-1.)

After picking up the request from workspace, the Activity Coordinator then uses the following steps to complete the process:
**Step 1:** Compares the request and its service and determines if the request can be processed.

If the request is beyond its capability, the Activity Coordinator simply replies to the requesting MOAP and rejects the request. Under normal conditions, the request should be valid and can be handled by the Controller MOAP because the requesting MOAP has a list of other MOAPs and their capabilities.

**Step 2:** Based on the pre-set of sub-goals, the activity coordinator then generates an activity path from procedure knowledge.

In this step, several decisions should be made for that workflow process:

- A set of sub-goals (usually the same as the pre-set at this time) required to achieve the overall goal;

- For each sub-goal, what task or service is needed;

- For each service, the MOAP who will be providing that service.

As we discussed in the previous section, tasks are not specified in a rigid sequence, but rather are set up to support a goal (Craven and Mahling, 1995), and so are the services. This mechanism provides more flexibility in that the MOAP is ‘free’ to define or select tasks as needed on-the-fly instead of them having to be ‘pre-defined’. The relationships among different components are as follows: first, sub-goals suffice parent goal according to the sub-goal relationship. However, only top goal of a MOAP directly guides the service, which in turn satisfies requests.
Secondly, one sub-goal (leaf-goal in a goal tree) may be achieved by one or many tasks or services.

After this step, the goal and necessary activity path are defined and written to the workspace ready for execution. The process’ goal and activity path can be described by a value pair:

\[
\text{Goal: } \{G_1, G_2, \ldots, G_n\} \\
\text{Task/Service: } \{T_1, T_2, \ldots, T_n | S_1, S_2, \ldots, S_n\}
\]

![Image of diagram](image-url)

**Figure 5-1: Execution Mechanism of MOAP-Wf**

*Step 3:* Task execution according to the activity path.
Some tasks are executed automatically by software programs. Others may need to be done either by other MOAPs or users. The communication with other MOAPs is handled by the Activity Agent.

Under normal conditions, the Activity Agent has the knowledge to direct requests to the right MOAPs. Otherwise the exception handling mechanism will be triggered (c.f. next section). Upon receiving results, the Activity Agent then hands over the results to the Activity Coordinator.

**Step 4:** Completion of the process and a return of the results to the requesting MOAP or User.

The activity coordinator controls and monitors the execution of tasks to make sure that the specific goal is achieved. After handing over results to the requesting MOAP or User, the process is completed.

5.1.2 MOAP-Based Workflow Example

The leveled workflow model for the purchasing process is shown in Figure 5-2. In the first level, the Controller MOAP only coordinates with and requests services from the Dept. Manager, Purchasing, and the Finance Department. The Controller MOAP does not have control of nor does it have information of how each service-provider MOAP implements.

At the second level, Purchasing coordinates with the IT Dept and Vendor to provide ‘purchasing’ service, i.e. to get the required items. The Finance Dept provides payment service by further requesting services from the Accountant and the Cashier.
All the details of the second level workflow are encapsulated from the first level
Controller MOAP.

![Figure 5-2 MOAP-Wf Structure for Purchasing Process](image)

The Controller MOAP for the purchasing process can be depicted as in Table 5-1.

A formal representation of knowledge is out of the scope of this thesis; rather we
focus on utilization of different types of knowledge by MOAP. As such, we use plain
English to describe the internal structure of a MOAP and omit the detailed
descriptions and definitions of each knowledge item. For example, tasks and
functions are listed without detailed pre- and post- condition descriptions. It should be
noted that the sub-goals in Procedure Knowledge are a subset replica of that in
Problem Solving Knowledge. As we discussed in the previous chapter, those sub-
goals in Procedure Knowledge are well-organized and have proven to be stable, and
are to be achieved by the MOAP under normal conditions. Sub-goals in Problem
Solving Knowledge include those that exist only for ‘problem-solving’ purposes.
For illustration purposes, we only show the internal architecture of the Purchasing Dept. while omitting that of other functional MOAPs. The internal architecture of Purchasing Dept. is shown in Table 5-2.

### Table 5-1 Controller MOAP – Purchasing Process

<table>
<thead>
<tr>
<th>Goal</th>
<th>Supplied required items</th>
</tr>
</thead>
<tbody>
<tr>
<td>Service</td>
<td>Handle purchase requests</td>
</tr>
<tr>
<td>Procedure Knowledge</td>
<td></td>
</tr>
<tr>
<td>Pre-Set of Sub-Goals</td>
<td></td>
</tr>
<tr>
<td>Condition</td>
<td>Sub-Goals</td>
</tr>
<tr>
<td>None</td>
<td>Approved(PR)</td>
</tr>
<tr>
<td>Approved(PR)</td>
<td>Purchased(item)</td>
</tr>
<tr>
<td>Purchased(item)</td>
<td>Paid(invoice)</td>
</tr>
<tr>
<td>Tasks or Functions</td>
<td>None /* This means that there is no atomic or executable task for this controller MOAP, which requests services from other MOAPs to achieve the overall goal.*/</td>
</tr>
<tr>
<td>MOAP Selection</td>
<td></td>
</tr>
<tr>
<td>Sub-Goal</td>
<td>MOAP</td>
</tr>
<tr>
<td>Approved(PR)</td>
<td>Dept Manager</td>
</tr>
<tr>
<td>Purchased(item)</td>
<td>Purchase Dept</td>
</tr>
<tr>
<td>Paid(invoice)</td>
<td>Finance Dept</td>
</tr>
<tr>
<td>Problem Solving Knowledge</td>
<td></td>
</tr>
<tr>
<td>Business Rules</td>
<td></td>
</tr>
<tr>
<td>Condition</td>
<td>Sub-Goal</td>
</tr>
<tr>
<td>Approved(PR)</td>
<td>Purchase(item)</td>
</tr>
<tr>
<td>Purchased(item)</td>
<td>Paid(invoice)</td>
</tr>
<tr>
<td>Goal Decomposition and Reasoning</td>
<td></td>
</tr>
<tr>
<td>Goal</td>
<td>OR</td>
</tr>
<tr>
<td>Supplied required item</td>
<td>Supplied New Item</td>
</tr>
<tr>
<td></td>
<td>Supplied temporarily</td>
</tr>
<tr>
<td>Supplied New Item</td>
<td>AND</td>
</tr>
<tr>
<td>AND Approved(PR)</td>
<td>Purchased(item)</td>
</tr>
<tr>
<td></td>
<td>Paid(invoice)</td>
</tr>
<tr>
<td>Supplied temporarily</td>
<td>AND</td>
</tr>
<tr>
<td>AND Got identical used items</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Purchased(item)</td>
</tr>
<tr>
<td>Problem-Solving Tasks</td>
<td>None</td>
</tr>
</tbody>
</table>

69
### Table 5-2 MOAP – Purchasing Dept

<table>
<thead>
<tr>
<th>Goal</th>
<th>Purchased required item</th>
</tr>
</thead>
<tbody>
<tr>
<td>Service</td>
<td>Purchase items for business as required</td>
</tr>
<tr>
<td>Procedure Knowledge</td>
<td></td>
</tr>
<tr>
<td>Pre-Set of Sub-Goals</td>
<td></td>
</tr>
<tr>
<td>Condition</td>
<td>Sub-Goals</td>
</tr>
<tr>
<td>None</td>
<td>Got specification from dept. in charge</td>
</tr>
<tr>
<td>Got specification from dept. in charge</td>
<td>Found qualified vendor</td>
</tr>
<tr>
<td>Found qualified vendor</td>
<td>Issued(PO)</td>
</tr>
<tr>
<td>Issued(PO)</td>
<td>Shipped(item)</td>
</tr>
<tr>
<td>Shipped(item)</td>
<td>Verified(item)</td>
</tr>
<tr>
<td>Tasks or Functions</td>
<td></td>
</tr>
<tr>
<td>Select vendors from database</td>
<td></td>
</tr>
<tr>
<td>Place purchase order</td>
<td></td>
</tr>
<tr>
<td>Deliver items to users</td>
<td></td>
</tr>
<tr>
<td>MOAP Selection</td>
<td></td>
</tr>
<tr>
<td>Sub-Goal</td>
<td>MOAP</td>
</tr>
<tr>
<td>Got specification from dept. in charge</td>
<td>IT Dept</td>
</tr>
<tr>
<td>Verified(item)</td>
<td>IT Dept</td>
</tr>
<tr>
<td>/*quantity and quality */</td>
<td>Vendor</td>
</tr>
<tr>
<td>Shipped(item)</td>
<td></td>
</tr>
<tr>
<td>Problem Solving Knowledge</td>
<td></td>
</tr>
<tr>
<td>Business Rules</td>
<td></td>
</tr>
<tr>
<td>Conditions</td>
<td>Sub-Goal</td>
</tr>
<tr>
<td>Received item from vendor</td>
<td>Verified(item)</td>
</tr>
<tr>
<td>Found qualified vendor</td>
<td>Issued(PO)</td>
</tr>
<tr>
<td>Issued(PO)</td>
<td>Shipped(item)</td>
</tr>
<tr>
<td>Goal</td>
<td>AND</td>
</tr>
<tr>
<td>Purchased(item)</td>
<td>Got specification</td>
</tr>
</tbody>
</table>
Suppose an employee submits a purchase requisition (PR) for a new computer. Upon receiving a request, the Controller MOAP decides the process execution path by evaluating business rules to see if the request is within the scope of its goal and service. The process execution path is dynamically generated from goal/task/service evaluation and mapping. In this case, the path can be sub-goal 1: Approved(PR); sub-goal 2: Purchased(Item); and sub-goal 3: Paid(invoice). These sub-goals are to be achieved with services from other MOAPs.

After the execution path is generated, the Controller MOAP then sends requests to related MOAPs according to the MOAP/Capability list (in the Activity Agent component) and MOAP selection criteria in Procedure Knowledge. In other words, it

<table>
<thead>
<tr>
<th>Goal Decomposition and Reasoning</th>
<th>Received item from vendors</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Verified(item)</td>
</tr>
<tr>
<td>Got specification</td>
<td></td>
</tr>
<tr>
<td>OR</td>
<td>Got specification from dept. in charge</td>
</tr>
<tr>
<td>OR</td>
<td>Got specification from historical data</td>
</tr>
<tr>
<td>Received item from vendors</td>
<td></td>
</tr>
<tr>
<td>AND</td>
<td>Found qualified vendors</td>
</tr>
<tr>
<td></td>
<td>Issued(PO)</td>
</tr>
<tr>
<td></td>
<td>Shipped(item)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Problem-Solving Tasks</th>
<th>None</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activity Agent</td>
<td>MOAP List</td>
</tr>
<tr>
<td></td>
<td>IT Dept</td>
</tr>
<tr>
<td></td>
<td>User</td>
</tr>
<tr>
<td></td>
<td>Vendor</td>
</tr>
<tr>
<td>Workspace</td>
<td>Temporary Data</td>
</tr>
<tr>
<td></td>
<td>Purchase requests</td>
</tr>
<tr>
<td></td>
<td>Specification</td>
</tr>
<tr>
<td>Data Storage</td>
<td>List of vendors</td>
</tr>
<tr>
<td></td>
<td>List of purchase orders</td>
</tr>
</tbody>
</table>

71
first sends requests to Business Department Manager, and then based on the result received from Purchasing Department, sends a request to Finance Department for invoice payment.

After the Finance Department makes the payment, the Controller MOAP sends notification back to the employee and closes the purchase process.

It should be noted that how the Purchasing Department and the Finance Department handle the requests is unknown to the Controller MOAP. The Controller MOAP has no knowledge about what divisional rules apply and what steps are followed by each MOAP. For example, the Business Department Manager may check the employee's budget and year-to-date spending before approving the purchase requisition. These detailed steps are out of the 'control' of the Controller MOAP. Thus the autonomy and encapsulation of each MOAP are observed. Accordingly, each MOAP is responsible for handling exceptions, if any, at its level.

5.2 The Mechanism of Exception Handling

5.2.1 Overview

Generally speaking, one of the most important objectives in the handling of an exception is to isolate it and limit its impact it has as much as possible. This strategy is supported by our proposed MOAP-Wf with a leveled control structure.

In MOAP-Wf, each MOAP is responsible for the execution of its own level of tasks and handling exceptions if any. All the task executions of one MOAP are encapsulated and hidden from other MOAPs and its higher level Controller MOAP.
Exceptions in one MOAP do not affect other MOAPs’ operations, unless, for example, the MOAP returns errors to requesting MOAPs.

As stated in Woo and Lochovsky (1991), the bias of the MOAP architecture is towards trying to use procedural knowledge whenever possible, and, failing that; problem knowledge, activity agent (for external communication), or a user. We use also this concept of precedence in our proposed exception handling mechanism.

The general steps of exception handling are: 1. using the knowledge (procedural and problem-solving); 2. Soliciting help from other MOAPs (by activity agent); and 3. Consulting users. If, after the third step, the exception remains unsolved, then the whole process fails. We focus on the first two steps because they are to be handled by the system. Figure 5-3 gives an overview of the exception handling mechanism:

![Figure 5-3: Overview of Exception Handling Mechanism](image)

At different steps, different mechanisms are used. At the first step, we use Goal-Based Reasoning, while at the second step, OASIS Blackboard is also used.
5.2.1.1 Goal-Oriented Reasoning

Goal-oriented reasoning starts from a goal. Upon processing failures or exceptions, the MOAP first finds out if there are other options to carry out tasks and achieve its desired goal. The identified options are usually based on procedural knowledge. After identifying possible solutions, the MOAP then examines the effects of each solution until a desired one is found.

For each MOAP, goals are represented by a hierarchical tree in which super-goals are decomposed into sub-goals. Sub-goals are further decomposed into leaf-goals that can be achieved by certain atomic tasks or by the services from other MOAPs. During workflow execution, each MOAP find out which goal can be achieved by which task or service from other MOAPs by means-ends reasoning. Tasks are not specified in a rigid sequence, but rather are set up to support a goal, including handling new activities and exceptional situations (Craven and Mahling, 1995). The role of a goal in exception handling can be summarized as follows.

1. **Multiple Level Exception Handling**

Within the goal hierarchical tree, the top level goal is achieved in a bottom-up fashion from the leaf-goals. Upon discovering any exceptions, the MOAP will first try to solve them at the current level where they occurred. If not successful, the immediate upper level goal is not achieved. The MOAP then moves up and tries to solve the problems at this next level (Figure 5-4).
2. Decoupling of Goal and Task/MOAP Facilitates Exception Handling

Decoupling allows a MOAP have multiple alternatives to achieve one goal. If one task or MOAP fails to deliver the required results, the MOAP can try other similar tasks or request services from other MOAPs. With the goals as criteria, a MOAP is ‘free’ to choose different courses of action, as long as the actions can achieve the desired goals and satisfy business rules and constraints. Unlike traditional process modeling which is more rigid and difficult to change, the MOAP-Wf is more flexible and different process variants can be generated dynamically (Figure 5-6).

5.2.1.2 OASIS Blackboard

It is possible that none of existing problem-solving knowledge and procedural knowledge will solve the problem and meet the desired goals. In this case, the Controller MOAP will try to ask for assistance from other MOAPs.

We apply the OASIS Blackboard and Form concepts that were proposed by Martens and Woo (1997). OASIS (Organization Activity Support and Information System) is fully decentralized and distributed in that each task has a separate
blackboard from which the task starts to execute. In MOAP-Wf, the Workspace is used as a blackboard and the Activity Agent is in charge of soliciting help from other MOAPs in the case of an exception.

If, based on predefined parameters such as time, no MOAP can handle the request, the Activity Agent then assumes that the task has failed. In this case, the MOAP will return errors or contact and pass the unsolved exception to the user.

5.2.2 The Logical Steps

Based on the discussion in previous sections, we now formalize the workflow execution and exception handling in the following logical steps, which serve as the implementation design for the exception handling mechanism described in the next section. The logical execution and exception handling steps are shown in Figure 5-7.

Stage 1: Planning

In this stage, the MOAP initializes some required parameters for a new workflow process instance and prepares for execution. The parameters include input data and specification of preferences of the workflow instance. The general steps are as follows:

1. Goal Selection and Initialization

Under normal circumstances, the pre-set sub-goals in procedural knowledge will be selected and initialized. We classify a goal into three states: WAITING, SUCCESS, and FAILED. A goal in WAITING state is currently initialized and the MOAP is waiting for the execution results of related tasks or for a response from other MOAPs.
SUCCESS means the goal has been achieved and FAILED indicates that the selected task or MOAP failed to deliver the desired results.

2. Means-End Analysis

In this means-end analysis step, the MOAP evaluates and identifies what tasks or services from other MOAPs can achieve those sub-goals initialized in the previous step. The MOAP thus maintains a mapping between tasks/services (the 'means') and sub-goals (the 'ends'). The sub-goals are usually the smallest units that are not able to be further decomposed and can be immediately achieved by tasks or services.

We use a simple value pair to represent the mapping:

\[
\text{means\_ends(task | service, goal)}
\]

3. Scheduling

By scheduling, the MOAP decides the sequence in which goals are to be realized. This sequence is usually generated from pre-conditions of sub-goals stored in procedural knowledge or problem-solving knowledge.
We use a queue structure to represent the schedule. The goal in the front of the queue is tried first and removed (dequeued) if it is successfully achieved.

\[
\text{schedule(}G_1, G_2 \ldots G_n) \\
\text{dequeue(schedule, } G_1 ) \quad \text{/*if } G_1 \text{ is achieved */}
\]

**Stage 2: Execution**

After certain initialization and preparation steps in the planning stage, actual process execution starts. Two steps involved in this stage are task execution and service requesting. If pre-conditions are met, e.g. no goal dependency or resource sharing conflicts, these two steps can happen in parallel.
The two steps are represented as:

```c
execute(task)
request(service, moap)
/* request service from MOAP moap. */
```

**Stage 3: Evaluation**

The final stage is evaluation, in which the results returned by a task or MOAP are evaluated according to the related sub-goals.

If the desired results are achieved, the state of the sub-goal is updated and set as 'SUCCESS'. At this time, the overall top level goal is also evaluated. If the top goal state is also 'SUCCESS', then the whole workflow process closes, otherwise another execution-evaluation loop starts and tries to achieve the next sub-goal.

The evaluation steps are represented as:

```c
IF post-condition(task) = sub-goal
    Update_state(sub-goal, 'SUCCESS');
IF state(top-goal) = 'SUCCESS'
    Close(workflowProcess);
```

**Stage 4: Handling Exceptions**

This stage is an optional one that is to be triggered by failures and exceptions in both the execution and evaluation stages. At different stages, there are different types of failures and exceptions, which are classified as follows:

**Type 1 – Change of Workflow Sequence**

This type of change or exception is usually caused by changes in the preconditions and constraints of a workflow process, which in turn change the sequence of goal accomplishments.
One example is where the vendor changes the terms from on credit to prepayment, which results in the change of workflow sequence, i.e. the (pro-forma) invoice needs to be paid before the Purchase Dept can receive the item from the vendor.

**Type 2 – Evaluated Goal Failure**

An evaluated failure is one in which a goal is not achieved because one or more sub-goals fail depending on the sub-goal relationship. If the sub-goal relationship is ‘AND’, failure of any one sub-goal can scuttle the parent goal. However, if the sub-goal relationship is ‘OR’, the parent goal can only be evaluated to be ‘failed’ if all sub-goals are not achieved.

For example, an exception is raised by the Purchasing Dept when no supply (or capable vendor) is found in the marketplace. In this case, the goal PurchasedItem of Controller MOAP is failed. The failure of goal PurchasedItem consequently causes the parent goal SuppliedNewItem to fail. Upon finding such an exception, the Controller MOAP will try other alternatives. For example, it might find an equivalent used item to satisfy the user’s request (pursuing the goal SuppliedTemporarily).

**Type 3 – Non-Response/Unavailable MOAP Node**

This type of exception happens when a MOAP does not respond to a request in a timely fashion or a MOAP is not available. In both cases the requesting (calling) MOAP will try to find other capable MOAPs to provide services. The former case is similar to the time control concept of OOWM, in which the Controller object will
assume the whole workflow failed if it does not receive a reply from request recipient object by a preset deadline.

One example is that the Dept. Manager is not available to approve the PR. So the Controller MOAP sends a request to the General Manager instead.

**Type 4 – Reported Task/Service Failure**

This type of failure occurs at the execution stage and can be further categorized into two subtypes: service failure and task failure. A service failure occurs when a MOAP does respond to service requests, but fails to meet the requirements of service, i.e. fails to achieve the desired goal. Service failure is external to the requesting MOAP. To solve such an exception, the requesting MOAP will try to find other capable MOAPs or execute capable tasks if any, to handle the request.

An example would be if the IT department (MOAP) fails to provide specifications for the required items, the Purchase Dept. will try to get relevant historical data (task) from corporate business systems.

On the contrary, a task failure happens within, i.e. internal to, a MOAP node when a task execution fails to achieve the desired goal. In this case, the MOAP will try other capable tasks (means). The example assumes that the required items are not in the user’s or the department’s budget. Another step is added to the workflow process, Modifying Budget, to solve the exception. This step is done before the Dept. Manager approves the PR. All other steps will remain unchanged.

It should be noted that a Type 4 exception may also imply a Type 2 one. A task/service failure may ultimately fail a goal. However, the key difference between
the two types is that Type 4 is immediately after the execution, while Type 2 exception is failing the top level goal in the evaluation stage.

In summary, depending on the underlying causes, several problem-solving methods can be used, i.e. executing another task, requesting service from another MOAP, achieving another sub-goal as a replacement, soliciting help from other MOAPs, and finally consulting users.

The exception handling function can be illustrated as follows:

```
//Exception Handling Function
BEGIN_EXCEPTION_HANDLING
SELECT CASE method:
1: means = means_end(?<>currentMeans, currentGoal);
   //try other means (task or service to achieve
   //current goal
2: currentGoal = nextGoal(subGoals);
   //set currentGoal as another sub-goal in an
   //OR decomposition which is stored in
   //problem-solving knowledge;
3: blackboard(subGoals);
   //The MOAP will broadcast
   //its desired sub-goals in form of blackboard
   //to solicit help from other MOAPs (that may
   //not be in the MOAP's list.
4: consult(user);
   //as the last step, if exception remains, the
   //MOAP will consult users.
END SELECT
END_EXCEPTION_HANDLING
```

Depending on the method chosen, exception handling may follow different paths as shown in Figure 5-5. If the solution is to try other means (above-mentioned Method 1), the process will continue with the execution stage. However, if the
solution is to try other goals (Method 2), new planning and scheduling might be necessary, i.e. the process will continue with the planning stage.

5.2.3 Comparison of ADOME-WfMS and MOAP-Wf

In this section, we demonstrate the exception handling mechanism by using the above example. As we discussed in Chapter 3, ADOME-WfMS is one of the most comprehensive research works that we have surveyed. We do a comparison study by applying the ADOME-WFMS model on the same exceptions. Some modifications are made to match the above example and the one provided in Chiu 2000.

5.2.3.1 Comparison of Normal Processes

Figure 5-6 (modified based on Chiu 2000) shows an example of an activity diagram for a purchasing process based on ADOME-WFMS model.

In ADOME-WFMS, an activity is recursively decomposed into sub-activities and tasks, which are the smallest units. (The notation is similar to the one of process-subprocess-task in this thesis and other research work. We use ‘activity’ and ‘process’ interchangeably for this comparison study.)

The requisition workflow (i.e. purchasing process) is composed of the following sub-activities: purchase request, procurement, payment, and receive/check goods. There are two possible execution paths depending on the terms of the purchase order: credit or cash on delivery (COD). During execution, one of the paths is followed and enforced by the Activity Executor.
As a conventional activity-based model, the ADOME-WFMS treats an activity as a composition of multiple inter-dependent tasks that need to be coordinated, scheduled and executed. The coordination, scheduling and execution are centered on the Activity Executor of the system.

In MOAP-Wf, we adopt a different approach as described in previous chapters. Our approach is goal- and agent- oriented. We model a workflow as a decomposition hierarchy of goals, and the workflow is enacted by cooperation between MOAPs, which are intelligent and autonomous agents from an artificial intelligence perspective. The following figure shows the modeling and enactment in MOAP-Wf.
A: goal decomposition

```
Supplied item

Approved PR  Purchased item  Paid invoice
```

Legend:

- Goal
- Goal-subgoal

B: workflow enactment

```
User
PR
Controller MOAP

Dept Manager
Request to approve
Request to purchase

Purchasing Dept
Finance Dept

Request to pay
```

Legend:

- MOAP
- request

Figure 5-7 A Purchase Process in MOAP-Wf.

In Figure 5-7 we show only one level of goal decomposition, the top level goal of Supplied Item can be achieved if all of sub-goals - Approved PR, Purchased Item, and Paid Invoice - are accomplished. Part B of the figure shows that each of the sub-goals can be achieved by three MOAPs: Dept. Manager, Purchasing Dept., and Finance Dept., respectively.

Compared to ADOME-WFMS, our MOAP-Wf model is more dynamic and flexible:

- Instead of following a fixed activity path, MOAPs decides a course of action dynamically at runtime depending on the status of goal achievement and resources available for accomplish the goals. From this perspective, MOAP-Wf is 'descriptive' while ADOME-WFMS is 'prescriptive' (Jablonski, 1994).
Because of the prescriptiveness of ADOME-WFMS, all of the workflow process and subsequent activities and tasks must be well known and fixed in advance. The only way to handle an exception is to pre-define certain compensation tasks (exception handlers). The number of handlers is limited and it is very difficult, if not impossible, to identify all the possibilities of workflow execution. Since all the activities/tasks are rigidly bound together, it is not easy to make changes to the process definition. If an exception cannot be solved, a new process definition becomes necessary. Defining a new process triggers other issues like how to migrate running instances to new definitions.

Being descriptive, MOAP-Wf does not ‘prescribe’ detailed processes/tasks in advance. Each step to execute relevant tasks and to accomplish its goals is decided at runtime by each MOAP. Exceptions can be solved by a MOAP trying different tasks, sub-goals or requesting services from other MOAPs.

- In ADOME-WFMS, the Activity Executor controls everything, including all activities and tasks. In MOAP-Wf the Controller MOAP only controls the immediate level of sub-goals. Each MOAP is free to decide how its goal should be achieved. For example, the Purchasing Dept. has full control of its own goal ‘Purchased Item’, which defines how an item should be purchased, how to find vendors and how to pick one from the choices.

Because of the separation of responsibility, MOAP-Wf has improved exception handling capability. Possible exceptions are decomposed and
distributed among multiple MOAPs for handling. And the 'success rate' of solving exceptions can therefore be improved.

### 5.2.3.2 Comparison of Exception Handling

In ADOME-WFMS, handlers must be pre-defined to solve any failures or exceptions. This approach thus requires that exceptions be identified and modeled into the workflow. A typical exception handler itself is modeled as a sub-activity and is in the form of an event-condition-action rule (or ECA rule).

Table 5-3 Exception Handling Example in ADOME-WFMS

<table>
<thead>
<tr>
<th>Activity/Task</th>
<th>Exception Condition</th>
<th>Exception Handling Resolution</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Budget Check</td>
<td>The software that checks budget raises 'Budget_Exceeded'.</td>
<td>The task's re-execution mode is 'critical' and there is no specific handler. The whole 'purchase request' activity fails, thus the whole workflow fails.</td>
</tr>
<tr>
<td>2. Procurement</td>
<td>Vendor changes payment term of 'credit'.</td>
<td>Since there is another replaceable branch, the activity executor would attempt the 'COD' branch instead.</td>
</tr>
</tbody>
</table>

As shown in Table 5-3 (adopted from Chiu 2000), a resolution is defined for each possible exception. Any exceptions other than those pre-defined ones will fail the whole workflow.

In the following two sections, we illustrate how the above two exceptions are handled in MOAP-Wf.

**Exception 1: Budget Exceeded (Reported Failure)**

In ADOME-WFMS the Activity Executor is in control. The approach in MOAP-Wf is more flexible in that each MOAP is responsible for execution and exception
handling. The exception 'Budget_Exceeded' is handled by Dept. Manager instead of the Controller MOAP. The exception handling process is shown in Figure 5-8.

![Diagram of exception handling process](image)

**Figure 5-8 Handling ‘Budget_Exceeded’ Exception in MOAP-Wf.**

After the execution of task ‘Check Budget’, the goal of ‘Approved PR’ is not achieved (the state of the goal is WAITING). By doing a means-end analysis (as illustrated in previous sections), the Dept. Manager then tries another task – ‘Modify Budget’. After this step, the result is evaluated that the budget is met and goal is achieved. It should be noted that ‘Modify Budget’ is only one example. There could be multiple solutions that the Dept. Manager can decide among dynamically at workflow runtime. These new tasks can be added to the execution without affecting others.

**Exception 2: Procurement (Change of Sequence)**
Because each MOAP decides the course of action (i.e. workflow steps) dynamically at runtime, handling the second sample exception (vendor changes payment terms) is straightforward. In this case Purchase Dept. is responsible for solving the problem since without payment it cannot achieve the goal of 'Purchased Item'. Originally the Purchase Dept. does not need to care about payment because all purchases are to be made on credit. The actual workflow is depicted in Figure 5-9.

A: goal decomposition

```
+-----------------+
| Supplied item   |
+-----------------+
    | and            |
    | Approved PR    |
    | Purchased item |
    | Paid invoice   |
        | Prepaid       |
```

B: workflow enactment

```
User -> PR -> Controller MOAP
     Request to approve
     Dept Manager
     Request to purchase
     Purchasing Dept
     Request to pay
     Finance Dept
```

Legend:
- Goal
- Goal subgoal

Figure 5-9 Handling ‘Payment Terms Change’ Exception in MOAP-Wf

To solve the problem, the Purchasing Dept. will send a payment request to the Finance Dept. so as to achieve the sub-goal ‘Prepaid’. This, in turn, helps achieve the overall goal of ‘Purchased Item’. It should be noted that the accomplishment of sub-goal ‘Prepaid’ will automatically achieve the Controller MOAP’s goal, ‘Paid Invoice’
(the dotted arrow in Figure 5-9). So the Controller MOAP will no longer need to coordinate with the Finance Dept. because the goal ‘Paid invoice’ is already achieved.

5.2.3.3 Summary of Differences

In the following table, we summarize some of the differences between ADOME-WFMS and MOAP-Wf that we have proposed in this thesis.

<table>
<thead>
<tr>
<th>Features</th>
<th>MOAP-Wf</th>
<th>ADOME-WFMS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Workflow model</td>
<td>Goal-based, Agent-based</td>
<td>Activity-based</td>
</tr>
<tr>
<td>Execution</td>
<td>Goal-oriented, Multiple levels of control (Controller MOAP and other MOAPs.)</td>
<td>Event-driven, Single centralized control (Activity Executor)</td>
</tr>
<tr>
<td>Process/task path</td>
<td>Dynamically decided at runtime</td>
<td>Pre-defined at build-time</td>
</tr>
<tr>
<td>Type (Jablonski, 1994)</td>
<td>Descriptive</td>
<td>Prescriptive</td>
</tr>
<tr>
<td>Exception Handling</td>
<td>Solution decided dynamically based on goal (means-ends analysis)</td>
<td>Exceptions and handlers must be define a priori</td>
</tr>
<tr>
<td>Recursive exception handling (exceptions encountered by handler)</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Knowledge management (AI perspective)</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Capability of handling some unexpected exceptions</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Workflow type</td>
<td>Dynamic workflow (Du and Elmagarmid, 1997)</td>
<td>Core process + Exception process</td>
</tr>
</tbody>
</table>
5.3 Summary

In this chapter, we introduced the exception handling mechanism of MOAP-Wf. First, we elaborated the workflow execution schema under normal conditions. We focused on the utilization and application of different types of knowledge that MOAPs hold. We then explored the exception handling mechanism based on goal-based reasoning and the OASIS blackboard. The exception handling mechanism is modeled as four integrated stages – planning, execution, evaluation, and handling exceptions. Different types of exceptions were also identified. Lastly, we did a comparison of our MOAP-Wf model and the ADOME-WfMS by applying some exception cases.
6 Simulation of MOAP-Wf

In this chapter, we design a simulator to show how the MOAP-Wf architecture can be implemented to execute normal workflow processes and handle exceptions. We demonstrate this simulator by using some workflow cases. There are several advantages of using simulation (Mohan et al., 2002): 1). This type of simulation is often sufficient to give the users a feeling of the system and let them judge whether any functionality is missing, wrong, or unnecessary. 2). Simulation prototypes represent the concepts to the developers of how the business process might be implemented. 3). Users’ evaluation of the prototype can point out alternative courses for a business process, new missing process steps, previously undetected exception conditions, or new ways to visualize information.

6.1 Development Platform

The workflow simulator is developed as a web-based application using:

- Visual Basic Scripting Version 5.0
- Active Server Page 3.0
- Microsoft Window 2000 Server with Internet Information Server
- Microsoft Access 2000 Database

We adopt this web-based approach because:
• Web client programs (browsers) are available for all popular computing platforms and operating systems, providing access to information in a platform independent manner (Bentley et al., 1997).

• Browsers offer a simple user interface and consistent information presentation across these platforms (Bentley et al., 1997).

• Object-oriented features are supported by Visual Basic Script.

• Database connections are supported by VBScript and Active Server Page.

6.2 System Design

The MOAP-Wf Simulator is composed of two parts: functional modules and a database. Functional modules do actual computations and the database maintains all workflow definitions, including MOAP, goal decomposition and task definitions. The results of the workflow processes are also stored in the database.

With database support and functional modules, this simulator has several advantages as compared with the FlexFlow simulation tool (Mohan et al., 2002):

FlexFlow tool does not use real data and contains little or no real functionality (Mohan et al., 2002). On the contrary, our simulator does have real functionality, one example being the logic algorithm. Real data, albeit very simple, is stored in the database.

In the FlexFlow simulation tool, the information that appears in response to a client’s request is faked or static, and report contents are hard-coded (Mohan et al.,
Our simulator responds to requests and generates reports dynamically from the database. A brief introduction to FlexFlow is presented in Appendix A.

### 6.2.1 Database Design

The workflow database is the central repository of the simulator and holds all the MOAP definitions, procedural knowledge and problem-solving knowledge. A formal detailed representation of MOAP’s knowledge is out of the scope of this thesis. For the purpose of the simulator, we limit the scope to goal decomposition and simple task definition, an example being which task can achieve what goal. We do not consider further details and other constructs introduced in previous chapters.

The entity-relationship diagram of the database is shown in Figure 6-1.

![Figure 6-1 Entity-Relationship Diagram](image)

When it starts up, the simulator will connect to the database and perform initialization if necessary. After a workflow process is completed, the status of that process is written back to the database. The status could be ‘waiting’, ‘success’ or
‘failed’. As such, the database does not maintain additional information about
detailed workflow steps, like what has been done and when during workflow
execution. These detailed workflow steps are handled as ‘session’ data, which is
maintained in system memory and is disposed of when the process is finished or the
simulator is shut down.

6.2.2 Functional Modules

The simulator has the following functional modules: MOAP module, exception
module, and initialization module. The MOAP Module is the main function of the
simulator. The Exception Module simulates different types of exceptions. The
Initialization Module reads and writes MOAP and process data to and from the
database. The mapping of modules to simulated functions is shown in Table 6-1.

<table>
<thead>
<tr>
<th>Module (sub-module)</th>
<th>Simulated Workflow Function</th>
<th>Web Pages</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOAP Module</td>
<td>Main function of MOAP-Wf</td>
<td>Flow.asp</td>
</tr>
<tr>
<td>Evaluation sub-module</td>
<td>To evaluate goal state based on those of sub-goals</td>
<td>Evaluate.asp</td>
</tr>
<tr>
<td>Goal selection sub-module</td>
<td>To decide next goal to work on based on current workflow state and pre-conditions of each goal.</td>
<td>findNextGoal.asp</td>
</tr>
<tr>
<td>Task/MOAP Selection sub-module</td>
<td>To find task or MOAP that is capable of achieving certain goals.</td>
<td>findCapableTaskMoap.asp</td>
</tr>
<tr>
<td>Execution sub-module</td>
<td>To execute selected task or send request to capable MOAP.</td>
<td>executeTask.asp, sendRequest.asp</td>
</tr>
<tr>
<td>Exception Module</td>
<td>To simulate workflow exception.</td>
<td>Exception.asp</td>
</tr>
<tr>
<td>Initialization Module</td>
<td>To read MOAP and process information from database and initialize web session.</td>
<td>Initialize.asp</td>
</tr>
</tbody>
</table>
All the modules are written in Visual Basic Scripting language. The algorithm and control flow of these modules is presented in Appendix B. Appendix C illustrates the page transition of the web-based simulator application. Figure 6-2 shows an example of the Evaluation Module.

```vbnet
if session(str).item(goal) = "AND" then
    'if current goal's subgoalrelation = AND
    isSuccess = true
    isFail = false
    For i = 0 to ubound(arrGoal)
        vgoal = arrGoal(i)
        state = session(moap).item(vgoal)
        isSuccess = isSuccess AND (state = "Success")
        isFail = isFail OR (state = "Failed")
    Next
End if

'--------------------------------------------------------------------------
if session(str).item(goal) = "OR" then
    'if current goal's subgoalrelation = OR
    isSuccess = false
    isFail = true
    For i = 0 to ubound(arrGoal)
        vgoal = arrGoal(i)
        state = session(moap).item(vgoal)
        isSuccess = isSuccess OR (state = "Success")
        isFail = isFail AND (state = "Failed")
    Next
End if

'--------------------------------------------------------------------------
if isSuccess=true then
    session(moap).item(goal) = "Success"
end if
'--------------------------------------------------------------------------
if isFail=true then
    session(moap).item(goal) = "Failed"
end if
```

Figure 6-2 Example Code of Evaluation Module

### 6.3 Simulation of MOAP-Wf

In this section, we demonstrate the handling of different types of exceptions (cf. Chapter 5) with the MOAP-Wf simulator. A workflow process is started when a user
submits a new request (as shown in Figure 6-3), which is written to the database and its state is set as ‘Waiting’.

Upon receiving a user’s request, the Controller MOAP starts to process it based on its procedural knowledge, including goal decomposition and pre-conditions. In this example, two sub-goals are found and accomplishment of one of them can achieve the top goal – SuppliedItem. This is because the sub-goal relation is ‘OR’. If the sub-goal relation is ‘AND’, all sub-goals must be achieved. The Controller MOAP will try to achieve goal SuppliedNewItem because it is the default goal according to procedural knowledge (as shown in Figure 6-4).
The goal SuppliedNewItem is further decomposed into sub-goals: ApprovedPR, PurchasedItem, and PaidInvoice. All these sub-goals should be achieved because the relation is ‘AND’. Because ApprovedPR is a ‘LEAF’ goal, it can be achieved by an atomic task or service from other MOAPs. By searching its task and MOAP list, the Controller MOAP then sends the request to DeptManager, which is one capable MOAP.

Upon receipt of the request, DeptManager follows the same steps as Controller MOAP does, i.e. decomposing goals, executing tasks, or requesting services from other MOAPs if necessary. In this case, an atomic task CheckBudget is found in its procedural knowledge. The DeptManager will then try to do this task. We assume that there is a ‘blackbox’ application that can actually complete the function of
CheckBudget. For this simulator, we allow the user to choose possible results so as to 'simulate' it. The execution of a task can be either 'Completed Successfully' or 'Failed' (Figure 6-5).

If task CheckBudget fails because the cost exceeds the available budget balance, DeptManager will look into its problem-solving knowledge and find other capable tasks by means-end analysis. In this example, task ModifyBudget is found and then executed (Figure 6-6). This demonstrates the handling of exception case Type 4 – Reported Failure (task failure).
IF
MoapWorkflow
Demo
Overview | Process | MOAP | Exception | About

Current Moap | DeptManager
Current Goal | ApprovedPR

ONE of the task(s) can be executed to achieve the goal:

- ModifyBudget
- CheckBudget

('←' = Task to be executed.)

If DeptManager is not available, the request will be sent to the next capable MOAP, which is GeneralManager in this example. Figure 6-7 depicts the handling of this Type 3 exception case – Non-Response/Unavailable MOAP Node.
The following capable MOAP(s) has(have) been found:

- GeneralManager
- DeptManager

('→' = MOAP to be sent request.)

GeneralManager will also follow the general steps of goal decomposing, task executing, and service requesting. Upon successful completion of GeneralManager, the state of ApprovedPR is set to 'Success' and the Controller MOAP will continue with next goal, i.e. PurchasedItem (as shown in Figure 6-8).
The request is sent to PurchasingDept because it is the MOAP capable of achieving the goal of PurchasedItem. PurchasingDept also follows the steps of selecting sub-goals, deciding work sequence, and evaluating results based on a sub-goal relation. The goal PurchasedItem is decomposed into GotSpecification, ReceivedItem, and VerifiedItem.

The first sub-goal to be achieved by PurchaseDept is GotSpecification, which can be accomplished by requesting service from ITDepartment. If ITDepartment fails to provide specifications, PurchaseDept will try to get specifications from historical business data. The handling of such Type 4 exceptions – Reported Failure (service failure) is shown in Figure 6-9.
Under normal circumstances, goal ReceivedItem can be achieved by sub-goals: FoundVendor, IssuedPO, ShippedItem. However, exceptions could occur when the vendor changes payment terms; for example, from on credit to pre-payment. In this case, another sub-goal PaidInvoice is necessary for the accomplishment of ReceivedItem. So a request will be sent to FinanceDept by PurchasingDept (as shown in Figure 6-10).
When the process of PurchasedItem by PurchasingDept is completed, the result is sent back to Controller MOAP.

Under normal conditions, the next step of the Controller MOAP will be to request that FinanceDept achieves the goal PaidInvoice. However, in the above example, the goal is already accomplished by FinanceDept upon request from PurchasingDept. In this case, no additional request is necessary (as shown in Figure 6-11). This solves exception case Type 1 – Change of Sequence.
Figure 6-11 Simulation - Completion of Purchasing Process

After this step, the whole purchase process is completed. The state of the process is set to 'success' and the database is also updated.

Another possible exception for the Controller MOAP is that one or more of the sub-goals of SuppliedNewItem fail (the sub-goals as shown in Figure 6-8). In this case, the goal SuppliedNewItem will be evaluated as 'Failed'. To solve this exception (Type 2 – Evaluated Failure), the Controller MOAP will try other goals. A possible alternative is SuppliedTemp. The handling of the exception is shown in Figure 6-12.
6.4 Summary

In this chapter, we briefly discussed different options of simulator development. We chose a web-based solution because of its platform independency, wide availability, and support of object-oriented features. The functional modules and database design were then presented. Finally, we demonstrated the simulation of the MOAP-Wf with different types of exception cases that were identified in Chapter 5.
7 Conclusion and Future Research

7.1 Conclusion

The main objective of this thesis is to find out how workflow systems can be made flexible, adaptive and dynamic, and how these in turn handle exceptions and changes.

We first reviewed some definitions and classifications of workflow exceptions in the literature. Exceptions can derive from workflow schema or instances, expected or unexpected, and at the domain, process, resource or infrastructure level. A survey of related research work was done in order that we could have an understanding of current workflow systems and exception handling mechanisms. We found that most of the surveyed research work was based on a traditional rigid sequenced model and focused on handling exceptions at the system level by extending database transaction processing technologies. One common limitation of these is that they can only handle expected exceptions and generally do not cover unexpected exceptions.

Based on the findings of the survey, we then proposed to build a dynamic and adaptive workflow model by using the Micro-Organization Activity Processor (MOAP). MOAP is an agent architecture that supports organizational knowledge acquisition and communication. We extended it by introducing two more constructs: service and goal. Thus, in the proposed workflow model of MOAP-Wf, a workflow can be viewed as that MOAPs coordinate and cooperate with each other to achieve
certain business goals. A goal can be further decomposed into sub-goals until they can be achieved by executing certain tasks or by services from other MOAPs.

MOAP concept provides an 'organizational view' of a workflow. MOAPs are also fully distributed. For these reasons, we adopted the 'controller' concept of Object-Oriented Workflow Model (OOWM) by introducing a Controller MOAP in the proposed MOAP-Wf. The Controller MOAP casts an overall horizontal process view to and is the 'owner' of a workflow. The responsibility of Controller MOAP is to monitor and control the overall progress of a workflow by interacting with other 'functional' MOAPs. In order to maintain the autonomy of MOAPs and facilitate exception handling, the MOAP-Wf is constructed in hierarchical levels.

Based on the discussion of normal execution procedures, we proposed the exception handling mechanism of MOAP-Wf by using a purchasing process example. Exception handling in MOAP-Wf is a goal-oriented reasoning process, which includes steps of goal selection and analysis, means-end analysis, scheduling, executing, and evaluating. We also presented the algorithm of exception handling logic steps in high level language. We further identified four types of exceptions and failures that can be solved by the proposed approach. To demonstrate the advantages of MOAP-Wf, we did a comparison study of handling two exception examples by our proposed model and ADOME-WfMS.

Finally, we built a web-based workflow simulator with Visual Basic Scripting Version 5.0, Active Server Pages and a Microsoft Access database. The simulator is
based on our proposed MOAP-Wf and demonstrates the detailed steps of workflow execution and exception handling.

7.2 Contributions

One of the major contributions of this thesis is that it introduces and extends previous research work – Micro-Organization Activity Processor (MOAP) into workflow systems, a new domain that MOAP is applicable as architectural base. The MOAP-Base Workflow Model proposed an alternative to the traditional ‘process view’. The model not only considers the autonomy of agents from an enterprise view but also supports processes emerging out of agent interactions.

The introduction of the goal construct also presents some new thinking in business process and workflow modeling. Instead of modeling a workflow as rigid and interdependent activities, our goal-oriented approach focuses on realization of business goals. These goals are achieved by coordination and cooperation of independent and autonomous agents – MOAPs. The proposed MOAP-Wf is more flexible and adaptive than other approaches. Changes of activities or tasks might not change the overall workflow. We identified four types of exceptions and failures that can be solved with the proposed MOAP-Wf: change of workflow sequence, evaluated goal failure, non-response/unavailable MOAP node, and reported task/service failure.

Another major contribution of this thesis is the proposed exception handling mechanism. First, the hierarchical levels of the workflow model support isolation of exceptions and provide more alternatives to solve exceptions. Second, we applied
agent and artificial intelligence technologies, for example means-end analysis and scheduling, to exception handling.

7.3 Limitations and Future Research

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

First, our proposed MOAP-Wf is based on the assumption that business goals can be readily captured and modeled. The goal concept in this thesis is simplified while that of a real business context could be more complex. More research work needs to be done for comprehensive business goal modeling. For example, consideration should be carefully made in the goal-oriented reasoning process: how does one solution impact the goals of a MOAP? One solution might have a positive effect on one goal, but negative impact on another goal. Another point that requires future research is the use of goals and goal inheritance in hierarchical MOAPs. In this thesis, one goal is attached to a MOAP. However, there are some other places where goals can be used. For example, each service could have a goal, and each activity agent could have a goal. By expanding the use of goals, we can have a more complete MOAP architecture and, in turn, a more complete picture of business processes.

Secondly, the simulator we used to demonstrate the functions of MOAP-Wf is not a true workflow system per se. To implement a real MOAP-Wf system, we need to do more research on the technical aspect of building a MOAP agent in workflow context. For example, how is a MOAP's goal matched to a business goal, how is means-end analysis done within the intelligent agent, and how are the goal and knowledge
represented. Another issue needs to be addressed is how to improve system performance which might be affected by the flexibility and distributedness of MOAP.

Thirdly, in this thesis, we allow a MOAP participates in different workflow processes. However, there is one issue needs to be addressed: how business rules are inherited by each MOAP, e.g. how to solve the business rules conflicts among different processes. We also need to do more research on the communications among MOAPs, and between user and MOAPs. For example, how users interact with the system and MOAPs if necessary, and how lower level MOAPs provide sufficient information for upper level MOAPs to handle unsolved exceptions. Another point requires further research is that how a Contoller MOAP could know what to control and what not.

Finally, the proposed MOAP-Wf focuses on intra-organizational workflow. Although MOAP is capable of inter-organizational knowledge communication, further research is required to address some remaining issues, such as information exchange of business goals and communication protocol. An interesting topic could be how to build a web-based MOAP-Wf system by combining agent technologies and Web Service technology.


Chiu, K.W. 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.


Mohan, C. "Recent Trends in Workflow Management Products, Standards and Research," NATO Advanced Study Institute Workshop on Workflow


Appendix A – FlexFlow

This appendix, excerpted from Mohan et al (2002), gives a brief introduction to FlexFlow, a state machine based workflow system that was proposed by Mohan, Cohen, and Schiefer.

1. FlexFlow Process Model

FlexFlow models business processes as UML state diagrams. However, unlike UML, where state diagrams describe the behavior of objects, FlexFlow state diagrams describe processes. The following figure shows a FlexFlow state diagram:

![FlexFlow State Diagram](image)

Figure 7-1 FlexFlow State Diagram

A FlexFlow model includes the following constructs:

**Actions** - correspond to task logic being executed at the application server. They are atomic units of business work. Actions could appear in states and transition. An action can be used to interface to an external system.

**States** - correspond to stages in a business process. A state identifies a precise point within the process. In a given business process at a given state, the actions that
can be taken by various parties are completely defined by the set of outgoing transitions. A state may have an entry action, an action that is executed upon entering the state, and an exit action, an action that is executed upon leaving the state.

**Transitions** – represent the changes of the process states. A transition connects two states, a source state it exits and a target state it enters. A transition corresponds to an action that is taken in response to an event.

**Events** - are the named messages needing to get processed. An event can be an HTTP request or an incoming SOAP request. It can also be an event generated by another process.

**Guards** – are the conditions that need to be true before the action can be taken. In general, the guards can be rules.

**Contexts** – are data associated with a business process. A context consists of: 1.) session information that includes information about the user including roles and permissions. 2.) the data submitted by the user such as form entries and the data stored in the form such as identification of the process and the business object.

2. Visual Modeling & Simulation

The FlexFlow modeling tool includes a simulation component that allows development of a horizontal prototype. The prototype displays the facades of user interface screens from web application, possibly allowing some navigation between them. The modeling tool does not show real data and contains little or no real functionality. Information is faked or static. The purpose is to provide a process-
oriented navigation. The tool allows users to change the status of the current process by selecting one of the available actions on a simulation panel.

The authors argue that this type of simulation is often sufficient to give the users a feeling for the web application and lets them judge whether any functionality is missing, wrong, or unnecessary. The users' evaluation of the prototype can point out alternative courses for a business process and new ways to visualize information.
Appendix B – Control Flow of MOAP-Wf Simulator

Functional Modules

MOAP = the capable MOAP (requestee)  
Goal = the MOAP's top goal

Send request to the MOAP.

Type of Means

Means not found

Execute Task

MOAP = requester MOAP  
Goal = requester MOAP's subgoal (corresponding to the top goal of previous MOAP)

Status?

Success or Failed

Have Subgoals?

Yes (AND, OR)

Evaluate (based on subgoals)

Success or Failed

Status?

Waiting

Find parent goal

Find next subgoal to work on

Set requester MOAP's goal as 'success' or 'fail'

Find the requester MOAP (which requesting service from current MOAP)

Has Parent Goal?

Yes

Set the process as 'success' or 'failed' (top goal's state)

No

Is current MOAP controller?

Yes

No

Goal: Fail

Goal: Fail

Send request to the MOAP.

Type of Means

Means not found

Execute Task

Goal status

Fail

Goal status

Fail

Evaluate (based on subgoals)

Success or Failed

Status?

Waiting

Find parent goal

Find next subgoal to work on

Set the process as 'success' or 'failed' (top goal's state)
Appendix C – Page Transition Diagram of MOAP-Wf Simulator

![Diagram](image-url)