A PROPOSAL FOR A PROCESS MODELING METHODOLOGY

by

Qingyu Wang

B.Sc., Xi'an Electronic Sci. & Tech. University, PRC, 1990

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

August 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.

Department of Faculty of Commerce and Business Administration

The University of British Columbia
Vancouver, Canada

Date Sept 10, 2002

DE-6 (2/88)

ABSTRACT

Process models are often used to describe organizational processes in contexts such as information system development, introduction of ERP (enterprise resource planning) systems, or the redesign of business processes. A technique for describing a process is termed a Process Modeling Language (PML). Current PMLs suffer from several shortcomings. First, different languages emphasize different aspects of processes, and thus might have problems in terms of their expressiveness. Second, many PMLs are intended to support information systems development, rather than model general business processes. As such, they may not have a complete set of clear concepts to represent real world processes. Third, often, PMLs provide a notation, but do not have rules to guide the construction of a process model.

This thesis develops a theory-based process modeling methodology. The research starts by observing five PMLs: EPC (Event-controlled Process Chains), IDEF0 (Integrated Definition 0), IDEF3 (Integrated Definition 3), UML (Unified Modeling Language) Activity Diagrams and the EDPM (Event-Driven Process Model) Method. Based on analyzing the constructs of these PMLs, the thesis identifies and defines a set of generic PML constructs and uses them to compare the five PMLs. It then analyzes the notion of a process based on Bunge's ontology (as adapted by Wand and Weber, and termed BWWP) and develops an ontology-based process model (termed OBPM). The analysis leads to two outcomes: a set of rules to map generic PML constructs to BWWP constructs, and a set of ontology-based integrity rules for building process models. The first outcome is used to evaluate the five PMLs in terms of completeness and clarity. The evaluation helps to clarify the basic

concepts of each process modeling methodology and identify possible weaknesses. The second outcome is used to generate an ontology-based process modeling algorithm. This algorithm can be used to guide the construction of process models, where the integrity rules should be followed.

Together, the generic process model, the mapping and integrity rules, and the algorithm provide for a theory-based process modeling methodology. The methodology provides the basic concepts, procedures and rules to build a generic process model, which covers all aspects of processes: functional, behavioral, organizational and informational. The integrity rules enable the checking of process models for semantic correctness. The algorithm provides a modeling procedure (based on the mapping rules) to create a process model for a given situation. Finally, the mapping rules and the algorithm can be used to point our missing information in models based on PMLs currently in use.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

INTRODUCTION

In order to show the essentials of complex problems, we need to set up *models* to abstract from the real world, neglect those aspects that we are not interested in, and highlight those which are of interest.

The power of models comes from its ability to simplify the real-world system they represent, and make predictions about that system based on the facts within the models.

The real-world system the thesis aims to model is *business processes*. Business attains its goals via its business processes. According to Davenport and Short (1990), a business process is a set of logically related tasks that use the resources of an organization to achieve a defined business outcome. It is a set of sequenced activities leading to some outcome such as a product or a service for a customer. Examples of typical business processes could be: Proposal development, Sales tracking, Order processing, Claims processing, Assembly, and so on.

Process models are created for business users to represent their perspectives of how the business is conducted. These models form a basis for information system development such as Enterprise Resource Planning (ERP) system and Customer Relationship Management (CRM) system. A good business process model is also critical for understanding and improving business processes effectively. This could help in achieving the benefits of many new management philosophies such as Total Quality Management (TQM), Activity-based Costing (ABC), Business Process Re-engineering (BPR), and Work Flow Management (WFM) as the decision-making under these management strategies are heavily relying on the definitions of business processes.

A *Process modeling language* (termed as PML) is a technique used for describing, documenting and analyzing a businesses process.

1.1 State of the problems

The existing process modeling languages have problems in terms of their expressiveness. The identified problems are:

(1) As different languages emphasize different aspects of business processes, thus they might have problems in terms of their expressiveness. For example, they may have some constructs overlapping each other, or some necessary constructs are simply missing in the PMLs.

(2) Many PMLs are intended to support information systems development, rather than model general business processes, therefore, they might not have complete and clear concepts to represent the real world process. For example, most PMLs construct process models by looking at the information flow such as input or output data rather than investigating the facts regarding how concrete objects interact with each other during the processes.

(3) Most PMLs provide a notation, but do not have formally defined rules to guide the construction of a process model. For example, the PMLs construct a process model based on the procedures obtained from traditional process modeling heuristics or observations. They do not have formal rules to follow.

1.2 Thesis Objectives

The purpose of the thesis is to develop a theory-based process modeling methodology, which is intended to support both information systems as some PMLs do (such as EPC, IDEF0, etc.) and general business processes in real world. Therefore, it does not just emphasize the specific information processing aspects of business processes, but also

presents the organizational, functional and behavioral aspects of processes. The methodology should have the following features:

(1) Cover all the aspects of a business process rather than specific aspects.

(2) Include clearly defined constructs that are derived from ontological concepts.

(3) Provide a procedure for constructing a process model in a given situation.

(4) Have a set of integrity rules to enable checking that the model is semantically correct.

The thesis achieves it objectives by the following steps:

- Compare five currently used PMLs and identify the *generic PML constructs*.
- Associate the process modeling languages with BWW Ontology (an ontology presented by Bunge and extended by Wand and Weber), and find the *PML-BWWP construct mapping rules* between the basic concepts of each other.
- Generate a group of *ontology-based integrity rules* that should be followed to build a theory-based process model.
- Use the mapping rules as a guideline to *evaluate* the currently used five PMLs in terms of their expressiveness.
- Use the integrity rules to generate an *algorithm on building an Ontology-based Process Model (OBPM).*

1.3 Related work

In the information system requirements modeling area, several modeling languages have been analyzed based on the theory of ontology up to date. According to Green and Rosemann (2000), the ontological analysis work has been done as follows: traditional modeling languages by P. Green in 1997, structured language such as Data Flow Diagrams by Wand and Weber in 1989, data-centered languages such as Entity-Relationship Diagram

by Wand and Weber in 1989 and 1993, relational model by Sinha and Vessey in 1995, NIAM by Weber and Zhang in 1996, OO model by Sinha and Vessey in 1995, Parsons and Wand in 1997. Some ontological analysis to PMLs has been done by Peter Green and M. Rosemann (2000). They analyzed the ARIS architecture with BWW Ontology, found the mapping rules between the concepts of each, and evaluated the architecture in terms of ontological completeness and ontological clarity.

However, to the best of our knowledge there is no ontological study done on generic process modeling language constructs so far. Additionally, we did not see any previous work that attempts to find the theory-based integrity rules for constructing an ontology-based process model.

1.4 Thesis Outline

Chapter two presents a general review of five currently used PMLs by introducing their basic constructs and modeling rules. It is followed by an example of several business processes in a middle-sized real world manufacturing enterprise. The five PMLs are applied in the example to compare their expressiveness in different aspects.

Chapter three identifies the generic PML constructs that are necessary in a process modeling language based on the observation and modeling experience in the previous chapter. And, the five PMLs are compared with each other in terms of each generic PML construct to find the strengths and weakness of each PML.

Chapter four investigates the theoretical foundations of the PMLs by looking at BWW Ontology concepts. Process-related BWW Ontological concepts are organized as *BWWP* concepts. By analyzing how things communicate from an ontological perspective, a set of *PML-BWWP construct mapping rules* and a set of *ontology-based integrity rules* are found.

Chapter five focuses on the application of the first set of rules: PML-BWWP construct

mapping rules. It evaluates the five PMLs using the rules and finds their problems in terms of their ontological completeness and clarity. This evaluation helps to clarify the basic concepts of the process modeling methodology.

Chapter six focuses on the application of the second set of rules: Ontology-based integrity rules. A conceptual modeling algorithm on how to establish an "Ontology-based Process Model" (OBPM) is generated, and an example has been used to illustrate the procedures of running the algorithm. The OBPM algorithm defines the procedures on building a process model for the methodology, and the ontology-based integrity rules show how to check that the model is semantically correct.

Chapter seven summarizes the thesis on its contribution, limitations and future research.

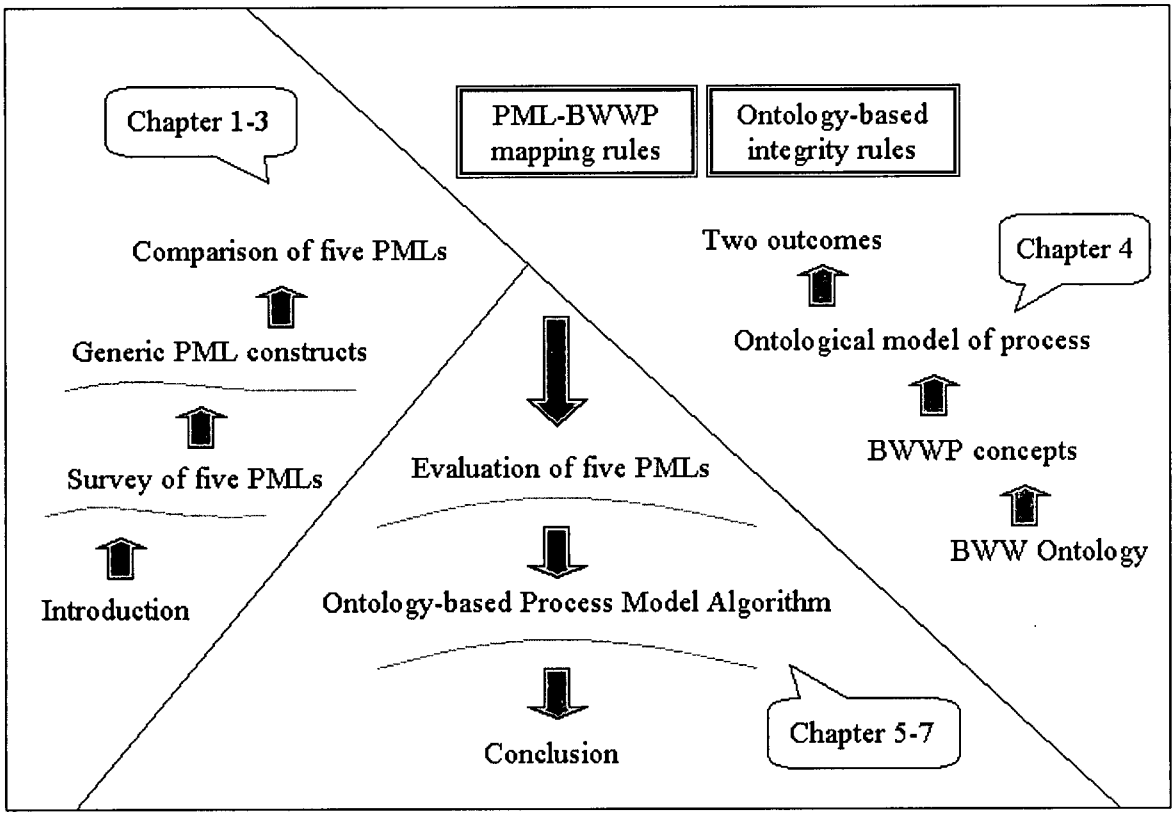


Figure 1-1: Thesis outline

Chapter 2

A SURVEY OF PROCESS MODELING LANGUAGES

Information system requirements-modeling languages have progressed from simple flowcharts, to data flow diagrams, to entity-relationship diagrams, to object-oriented schemas and integrated process modeling systems. Process modeling languages (PML) are the main components in the integrated process modeling system.

There are different categories of PMLs used for various purposes of business process management applications. According to Huckvale and Ould (1995), most PMLs fall into four categories based on the different application fields:

- Functional PML

The languages focuse on representing what operations are being performed and what dataflow connects them. For example, EPC, IDEF0, IDEF3 Process Schematics and Transition Schematics, UML Activity Diagram and EDPM method are all functional PMLs.

- Behavioral PML

PMLs in this category focus on representing the sequencing, feedback loops, iteration, decision-making and triggering conditions when operations are performed. For example, EPC, IDEF3 Process Schematics and Transition Schematics, UML Activity Diagram and EDPM method are behavioral PMLs.

- Organizational PML

This kind of PMLs concentrates on representing where and by whom activities are performed, plus physical communication mechanisms and storage media. For example, EPC, IDEF0, UML Activity Diagram and EDPM method are organizational PMLs.

- Informational PML

Informational PMLs focus on representing the entities such as documents, data, artifacts and products generated or manipulated by a process, including their structures and inter-relationships. For example, EPC, IDEF0, IDEF3 Transition Schematics and EDPM method are all informational PMLs.

As each modeling method and its associated architectural representations are designed to focus on one or some characteristics of a process, a variety of PMLs are necessary to meet the need of different kinds of applications. According to Zachman, there is not a "right" method or a "wrong" method, and each of them is additive and complementary to another (Mayer, 1995).

2.1 Introducing five currently used PMLs

The five PMLs introduced in this section are EPC, IDEF0, IDEF3, UML Activity Diagram, and EDPM Method. They are chosen to be investigated in the thesis because they cover all the four types of process modeling languages together. Table 2-1 shows the classification of the five chosen PMLs.

Aspect / PML	Functional	Behavioral	Organizational	Informational
EPC	It has "function" and "info in/out" constructs.	It has logical connectors.	It has "organizational unit" construct.	It has "info in/out" constructs.
IDEF0	It has "function" and "input/output" constructs.	It has logical connectors.	It has "mechanism" construct.	It has "input/output" constructs.
IDEF3	It has "UOB" construct.	It has logical connectors.	----	It has "object" construct.
UML AD	It has "activity" construct.	It has logical connectors.	It has swimlanes.	----
EDPM	It has "activity/operation" and "input/output" constructs.	It has logical connectors.	It has "agent" construct.	It has "input/output" constructs.

Table 2-1: Classification of the five PMLs

The purpose of this survey is to find what constructs are commonly used by the five PMLs, and what are bearing different implications on different PMLs. After practicing in

modeling some business processes in an example, this chapter suggests some observations on the implications of generic PML constructs.

2.1.1 Event-controlled Process Chains method (EPC)

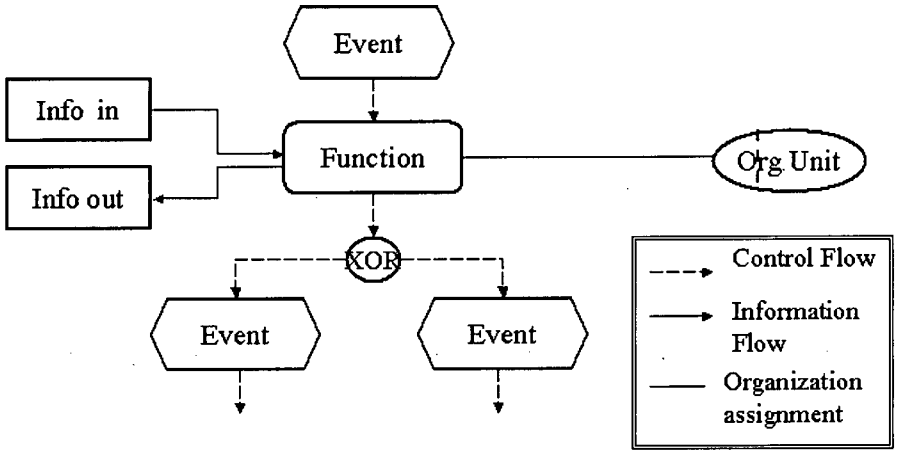


Figure 2-1: EPC basic constructs

Event-controlled Process Chains method is the modeling tool adopted by SAP R/3 system based on the Architecture of Integrated Information Systems (ARIS). SAP R/3 is an enterprise-wide, integrated information system which is playing a leading role in the Enterprise Resource Planning (ERP) market.

EPC method has the following basic constructs: *Organizational unit, Event, Function, Information in, Information out,* and *Logical connector.*

2.1.2 IDEF0

The Integrated DEFinition (IDEF) methodology is a suite or family of methods that supports a paradigm capable of addressing the modeling needs of an enterprise and its business areas. IDEF technologies have grown over the past four to five years, the Department of Defense is a prime user of the technologies, and some of the largest U.S. corporations have adopted the IDEF technologies for competitive advantage (Mayer, 1995).

IDEF0 is a "*Function Modeling Method*" of the IDEF modeling method family. It is

used to produce a "function model", which is a structured representation of the functions, activities or processes within the modeled system or subject area.

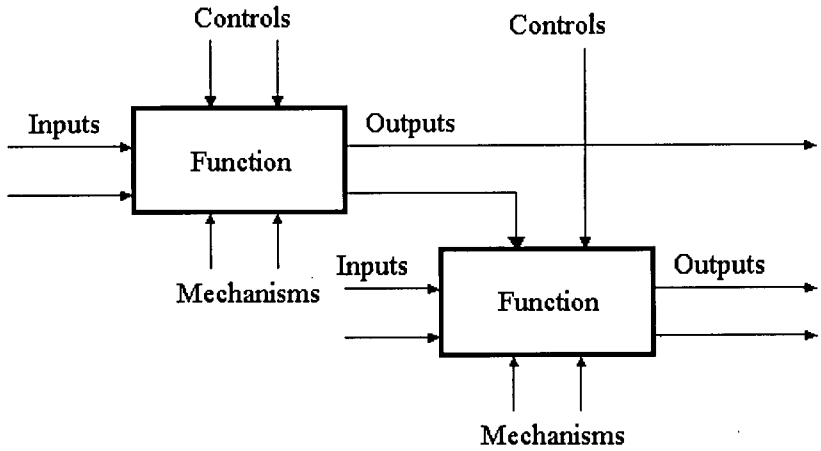


Figure 2-2: IDEF0 basic constructs

IDEF0 was developed for the US Air Force Program for Integrated Computer Aided Manufacturing (ICAM) for the purpose of increasing manufacturing productivity by systematic application of computer technology. Although IDEF0 was originally intended for use in system engineering, it has evolved and contains the necessary notations to support business applications.

IDEF0 basic constructs are: *Function*, *Controls*, *Mechanisms*, *Inputs*, *Outputs* and *Logical connector*.

2.1.3 IDEF3

IDEF3 is a "*Process Description Capture Method*" of the IDEF modeling method family. It was created specifically to capture descriptions of sequences of activities. It provides a structured method by which a domain expert can express knowledge about the dynamic operation of a particular system or organization. This includes capturing the state situations of the objects that participate in the process and objects that support the process, as well as the precedence and causality relations between events and processes (Mayer, 1995).

An IDEF3 Process Description is developed using two knowledge acquisition strategies: a *process-centered* strategy and an *object-centered* strategy. Two types of IDEF3 semantics parallel the two knowledge acquisition strategies. IDEF3 *Process Schematics* show a process-centered view of a scenario, and *Object Schematics* display the object-centered information. The Object Schematics that display an object-centered view of a single scenario are called *Transition Schematics*, and those that display the object-centered view of multiple scenarios are simply called *Object Schematics*. Transition Schematics that display additional objects and object relations to provide context-setting information are called *Enhanced Transition Schematics* (Mayer, 1995).

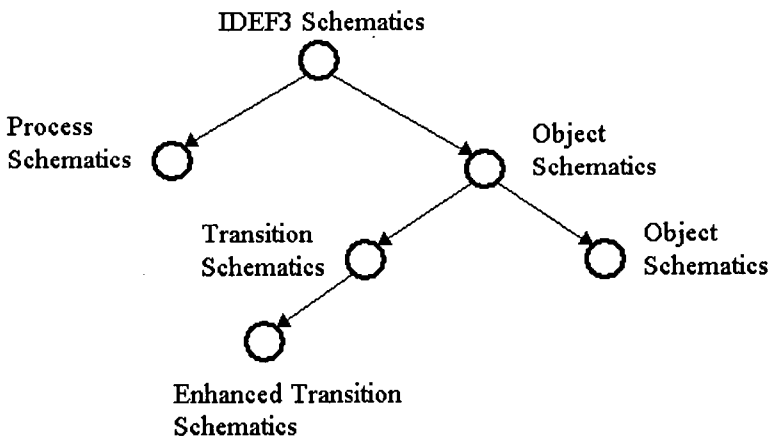


Figure 2-3: IDEF3 Schematics

Basic constructs of IDEF3 are: *Unit of behavior (UOB)*, *Object's state* and *Logical connector*.

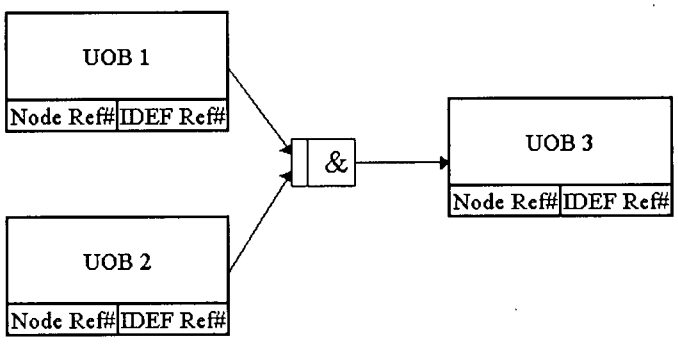


Figure 2-4: IDEF3 Process Schematic basic constructs

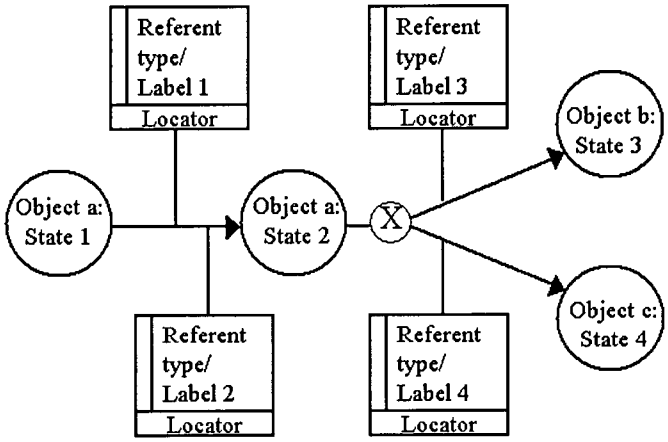


Figure 2-5: IDEF3 Transition Schematic basic constructs

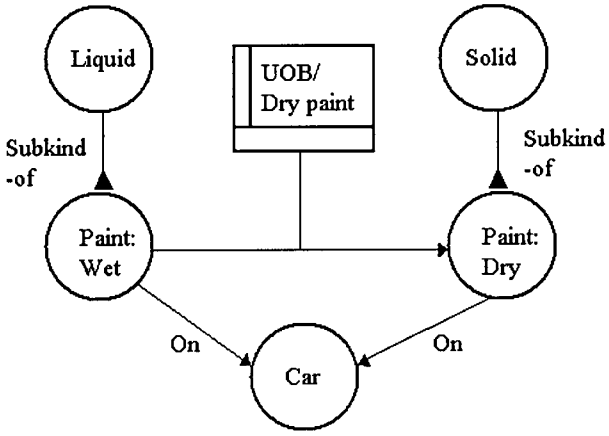


Figure 2-6: IDEF3 Enhanced Transition Schematic basic constructs

2.1.4 UML Activity Diagram

The latest Unified Modeling Language (UML) specification describes Activity Diagrams as a mechanism to capture business workflows, processing actions, and use-case flows of execution (OMG Unified Modeling Language Specification Version 1.3., 1999). A UML Activity Diagram documents the logic of a single operation or method, or the flow of logic of a business process. It shows the sequential flow of activities in a process, which could be one after another or in parallel. [1]

The basic constructs are: *Activity*, *Agent*, *Condition*, and *Logical connector*.

[1] Note: Choosing UML Activity Diagram is just for illustration of one PML in this thesis. The analysis and evaluation to the Activity Diagram does not apply to other diagrams in UML, and it is not intended to evaluate UML.

The concept of agent in Activity Diagrams is expressed by swimlanes.

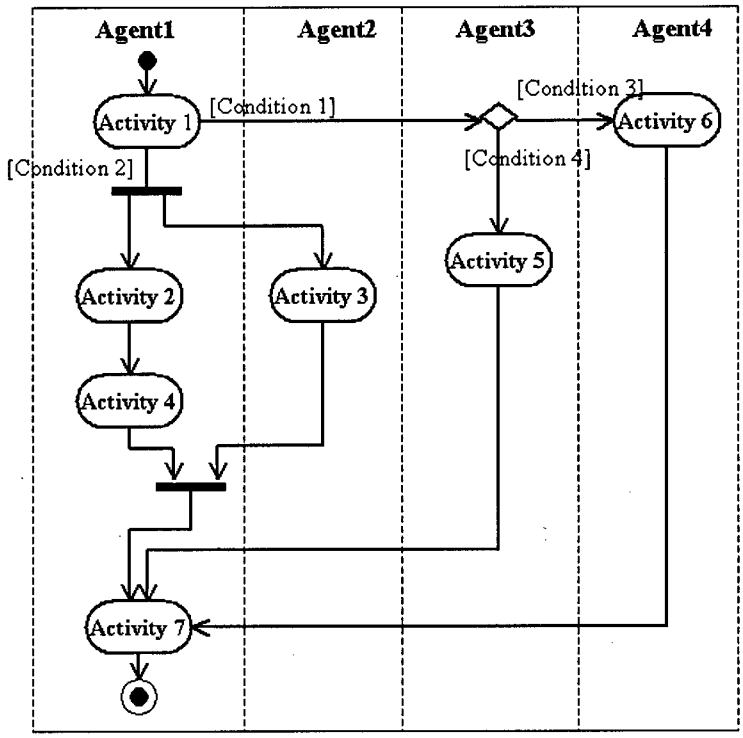


Figure 2-7: UML Activity Diagram basic constructs

2.1.5 EDPM Method

The Event-Driven Process-Modeling (EDPM) method introduced here was developed by Jacob Steif based on the notation of Martin and Odell (1992).

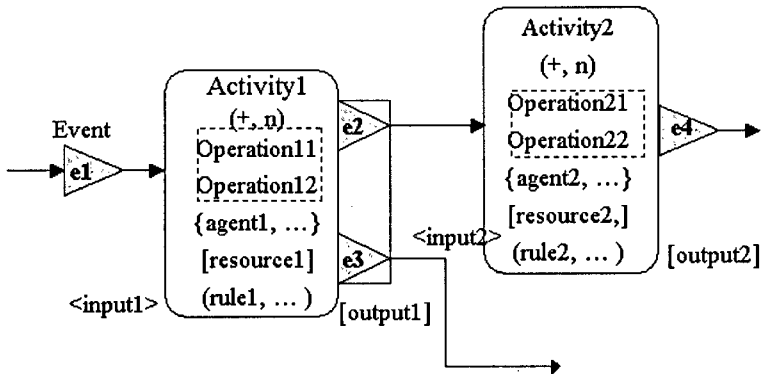


Figure 2-8: EDPM Method basic constructs

The basic constructs are as follows: *Agent*, *Activity*, *Operation*, *Event*, *Resource*, *Input*, *Output*, *Logical connector* and *Business rule*.

The logical relations between events and activities are shown in Figure 2-9.

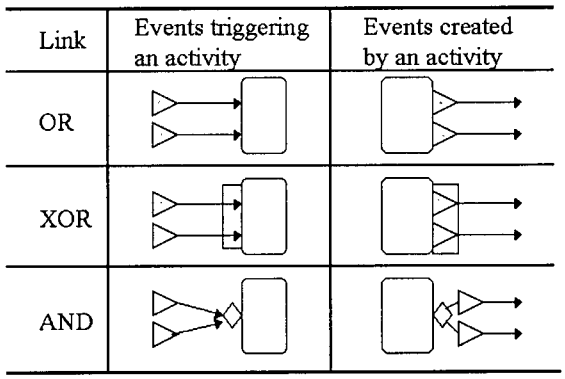


Figure 2-9: Logical connectors in EDPM Method

2.2 An example of business processes

To understand how to use each PML introduced above, we need to go through a real-world process example and model it with the five PMLs. This is an example of typical business processes in a middle-sized manufacturing enterprise.

The enterprise assembles water purification systems from components purchased locally or internationally, and sells them to the local market. Its supply chain model adopts the "Build-To-Order" strategy, and the manufacturing or purchasing starts only when customer places orders and their payment is made. This strategy enables the enterprise to keep low inventory level and make quick response to customers' requests. For enterprise privacy and the easy reference in later chapters, the enterprise' name is omitted, and the example is to be referenced as "*WaterSystem*" in later parts.

There are five processes in the example: 1. Order taking 2. Order processing 3. Assembly 4. Purchasing 5. Arrival. For simplicity, only "Order processing" process which includes most complicated cases and covers all the aspects of a business process will be illustrated here modeled by different PMLs, and the complete business process models could be found in Appendix I.

2.2.1 Process Description

The triggering events of this process are "*customer payment made*" generated from "Order taking" process and "*assembled products ready*" coming from "Assembly" process. When the first triggering event occurs, the Account Receivable department will collect customer's payment into the company's bank account. The section clerk might use bank machines or ask a bank teller for help.

Starting event	Triggered operation	Ending event	Resource used	Agent	Data
Payment made	Collect payment	Payment collected	Bank machine, bank teller	A/R	Customer file
Payment collected	Check order in warehouse	Order ready XOR order not ready	–	Warehouse	–
Order not ready	–	–	–	Warehouse	–
Order ready OR assembled product ready	Ship products	Order shipped	Forklift, elevator	Warehouse	–
Order shipped	Rearrange warehouse	Warehouse rearranged	Forklift	Warehouse	–
Warehouse rearranged	–	–	–	Warehouse	–
Order shipped	Send invoice	Invoice sent	Mail	Sales dept	–
Invoice sent	Close order	Order closed	Computer	Sales dept	–
Order closed	-	-	-	Sales dept.	–

Table 2-2: Definition of "Order processing" process

After payment is collected, the Warehouse department clerk will check if the order is ready for shipping. If not ready, she might wait for notification from the manufacturing department. If the order is ready in warehouse OR the assembled products are ready for shipment (which is the second triggering event), the Warehouse department will ship the order with help of a forklift and an elevator. When the order is shipped, two things need to be done: The Sales department should send invoice to the customer by mail and close the order, and the Warehouse department should rearrange the warehouse space for new products and components.

The sample process covers all aspects of a business process. For example, the organizational aspects could be "A/R department", "Warehouse department", and "Sales department". The functional aspects could be "collect payment", "ship products" and "send invoice". The behavioral aspects could be the rule that, when an order is ready or the assembled products are ready, the warehouse department will ship products. The informational aspects could be a customer file and a bank machine, which are used when A/R department collect payment.

2.2.2 Model the process with PMLs

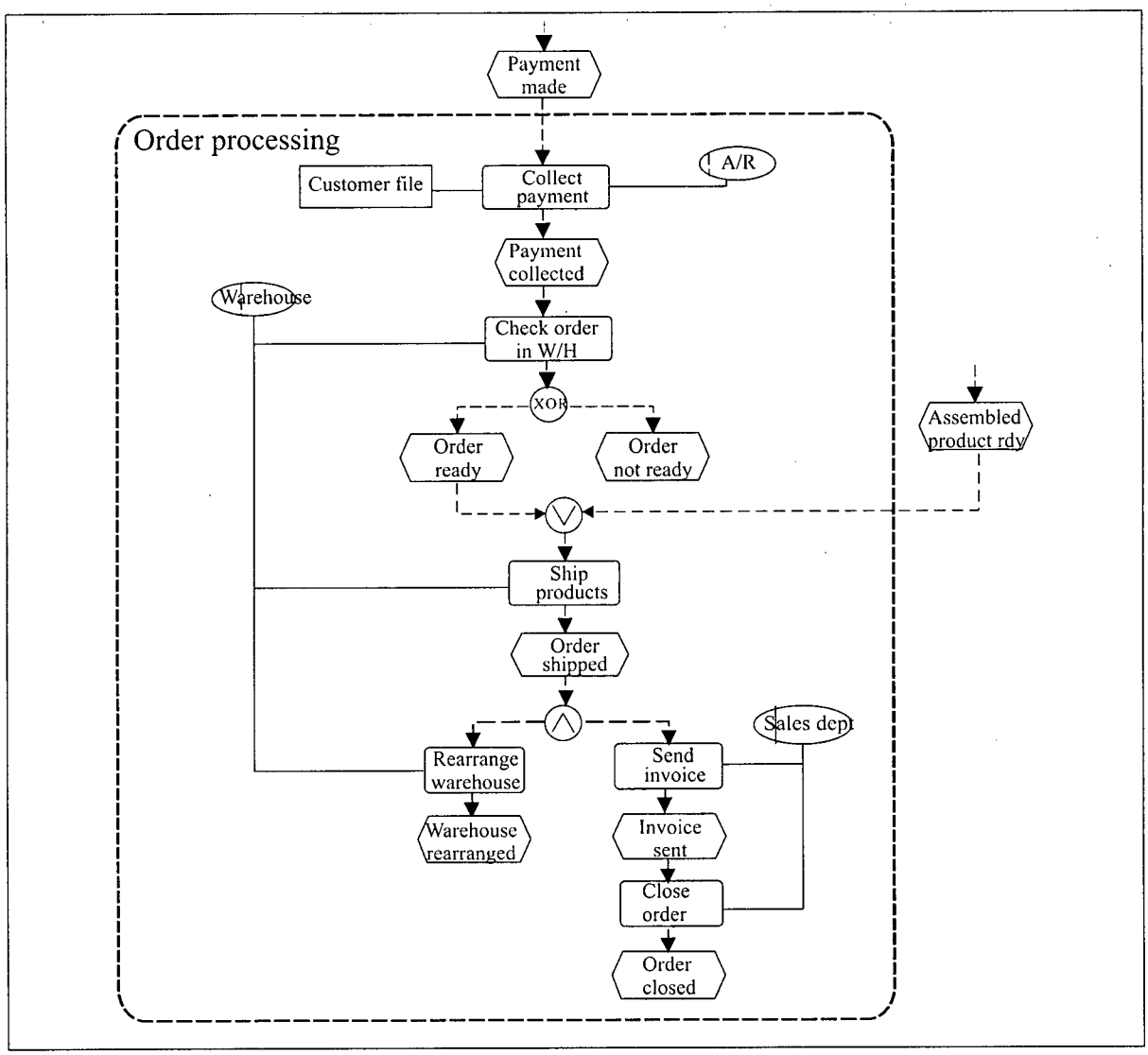


Figure 2-10: EPC model of "Order processing" process

In order to compare the expressiveness and the ease-of-use of each PML, the "Order processing" process is modeled by the five PMLs. For simplicity, IDEF3 Enhanced Transition Schematic model, which focuses on the static state of a system, is omitted. The models are displayed in Figure 2-10, 2-11, 2-12, 2-13, 2-14 and 2-15.

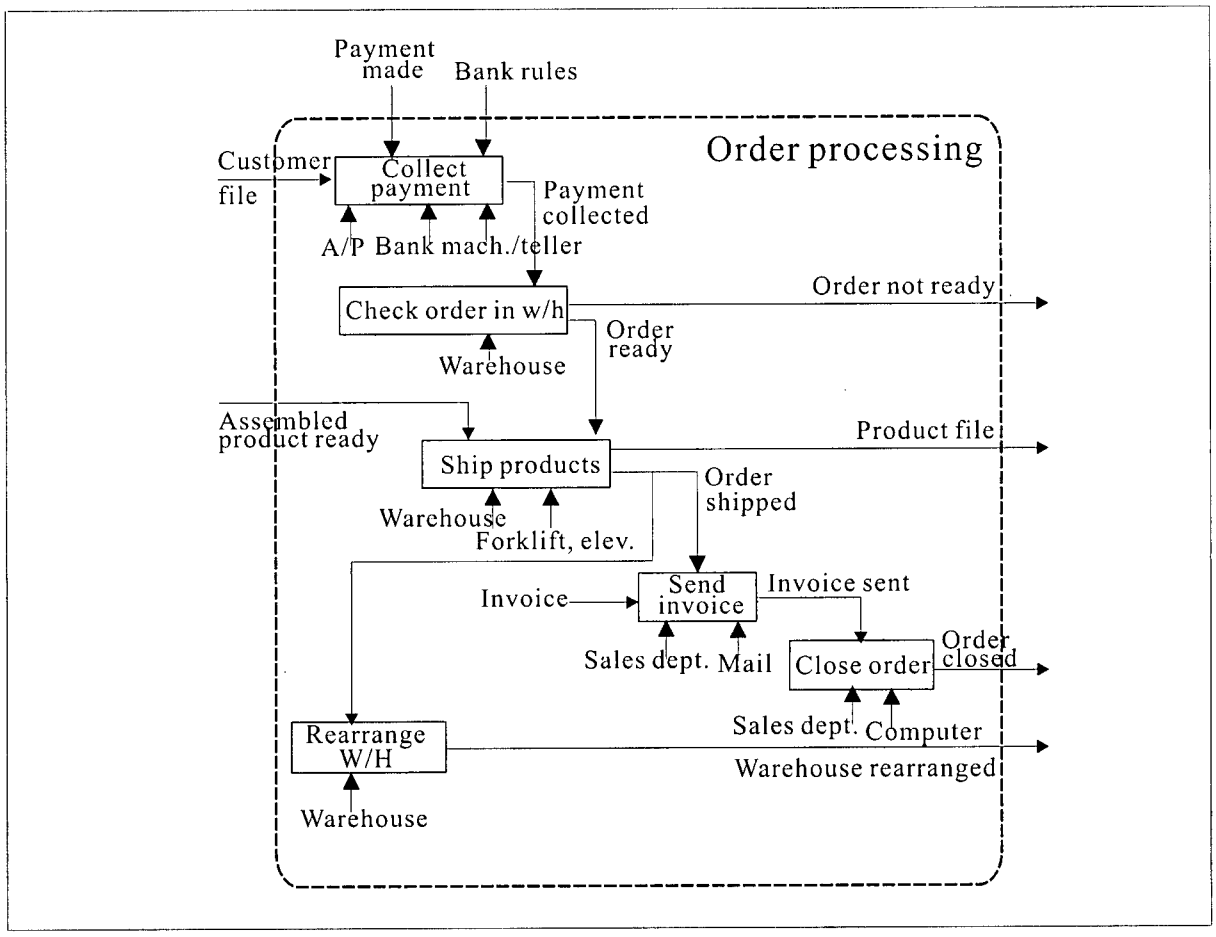


Figure 2-11: IDEF0 model of "Order processing" process

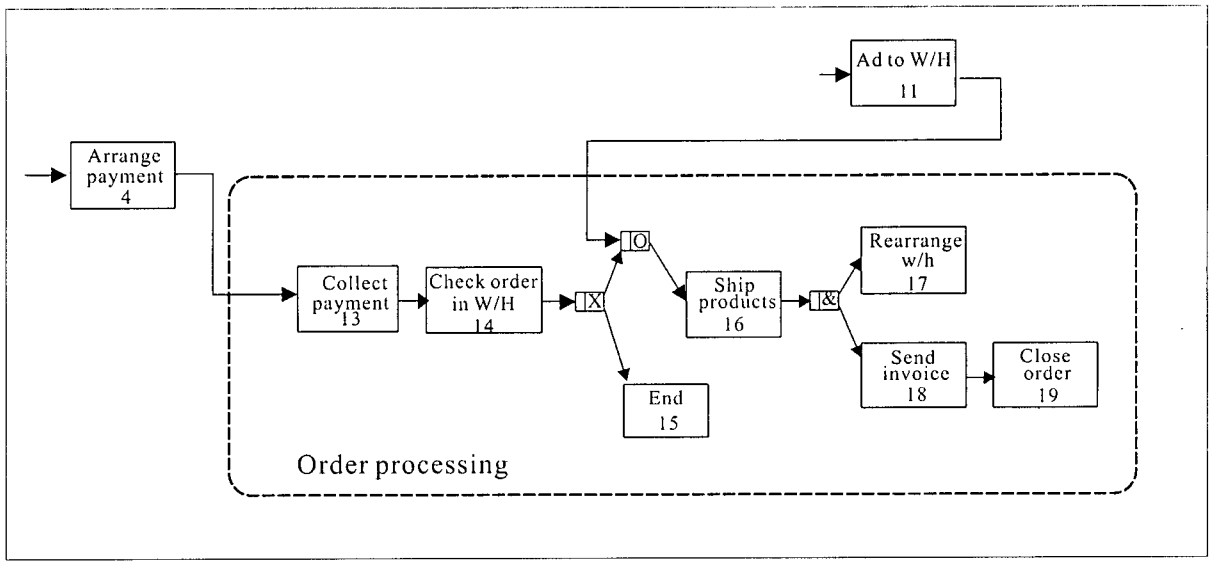


Figure 2-12: IDEF3 Process Schematic model of "Order processing" process

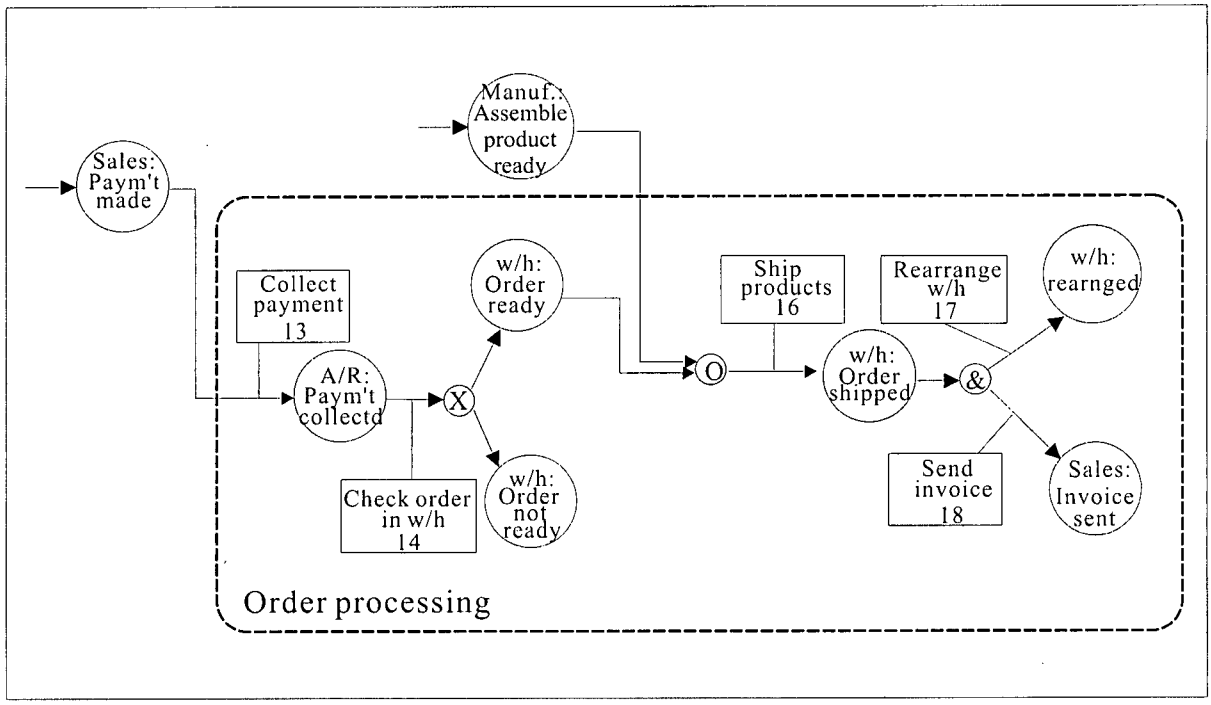


Figure 2-13: IDEF3 Transition Schematic model of "Order processing" process

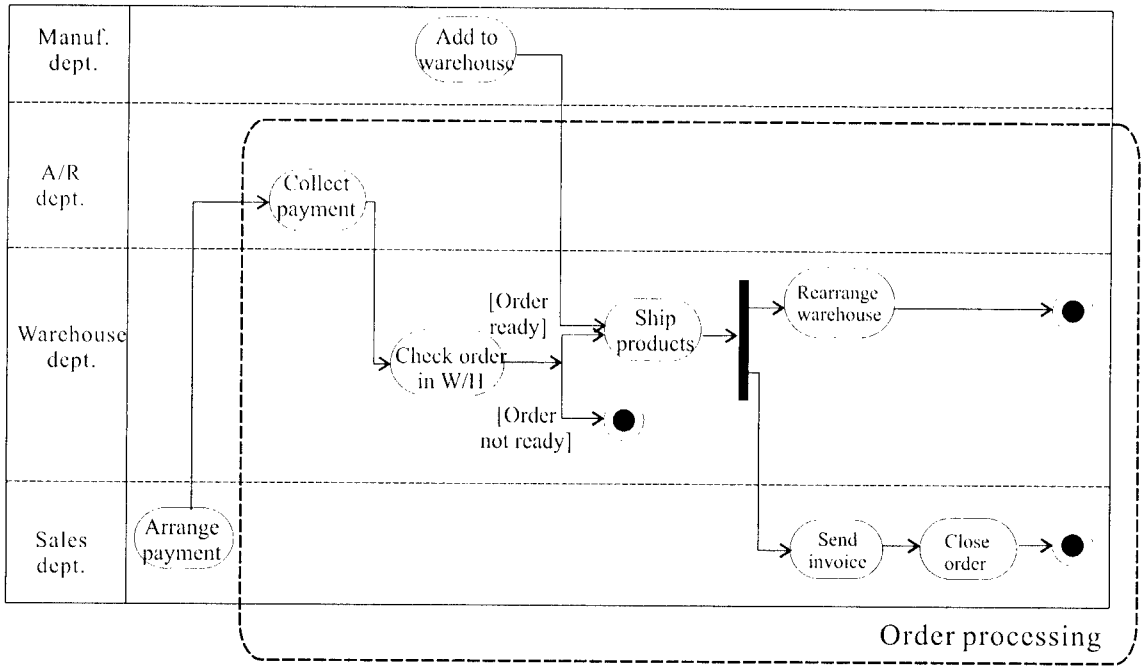


Figure 2-14: UML Activity Diagram model of "Order processing" process

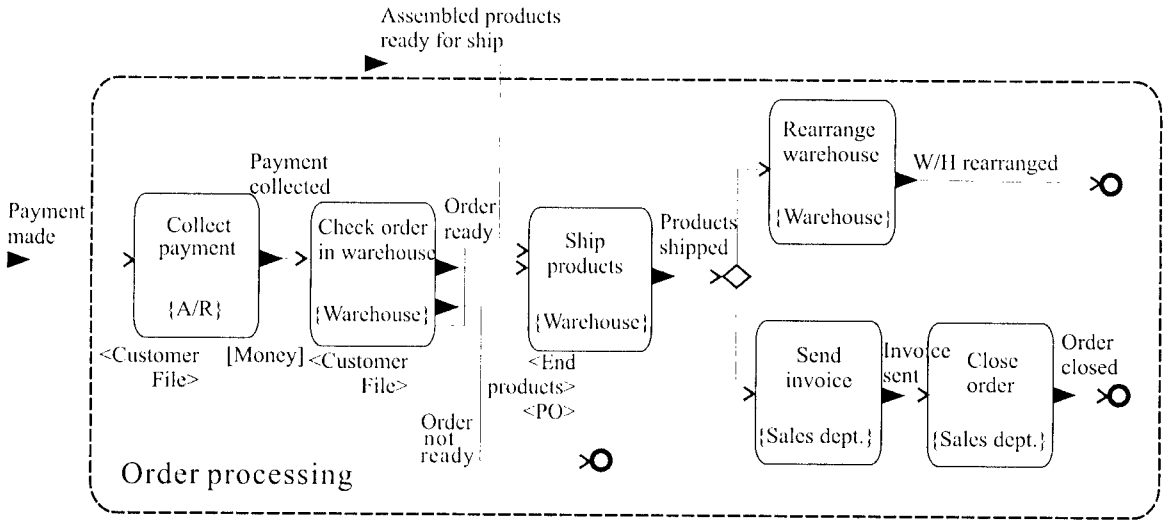


Figure 2-15: EDPM Method model of "Order processing" process

Chapter 3

GENERIC PML CONSTRUCTS

Based on previous observations, it is clear that a *process model* defines the dynamic features of a system. It abstracts from the uninteresting details of the real-world process, and only describes the facts about the following: the participants in the process, the activities triggered by an event or events, the events that trigger an activity or activities, the order of activities to be performed, the order of events occurred, the resource utilized in an activity, and the input and output objects of an activity.

This chapter attempts to summarize the generic PML constructs that are necessary to represent a good process model, and compare the five PMLs in terms of those constructs.

3.1 Generic PML constructs

Based on the five PMLs' notations and the observations on their modeling practice, the basic constructs which are needed to model a business process are summarized and defined below:

PML construct	Explanation
Process	A process is a sequence of activities or operations leading to a desired outcome or goal. It may involve one or many participants, which interact with each other by executing activities and operations. When all participants stop their state-changes, the process is over.
Agent	An agent is a person or an organizational unit that participates in the process. It is the active participant of the process that executes activities and operations. An agent could execute an activity or operation and keep changing until it is stable. It can also activate other participants to start changing.
Non-agent or Resource	Another kind of participant in a process is a non-agent or a resource, which is not able to activate any participants including itself during the process. It might be a resource which is utilized by an agent in an activity or an operation, or an object which is transformed or produced by the activity or operation.

- To be continued to the next page

PML construct	Explanation
Activity	An activity is the elementary task an agent should perform in a process. The execution of an activity is atomic, which means that once the activity starts it will finally complete and the agent will not change its state unless it is triggered to change again.
Operation	An operation is a component of an activity. An activity is divided into operations so that the agent's intermediate states could be captured. These intermediate states might trigger other agents to change, or might cause the agent itself to continue its state-change.
Event	An event is a notable occurrence at a particular point of time, which causes the state of an agent or a non-agent to change.
State	The state is regarded as the values of all properties of a participant at a particular point of time.
Data	Data represents the state of an agent or non-agent. For example, the data of product quantity represents the state of a non-agent "product". The data which is necessary for the function to process is the input data, and the data which results from the execution of a function is output data.
Logical connector	The logical connectors could be considered as the definition on the preconditions of an activity to be initiated, or the post conditions when an activity is completed. They define the rules between activities (operations) and events. There are three kinds of logical connectors: "AND", "OR" and "XOR".
Business rule	Business rules provide the law or rules within an activity or operation that the agent should follow. If this activity or operation is decomposed, the business rules will be eventually expressed by logical connectors between the refined activities or operations.
Input	Inputs are the data or objects that are transformed by the operation or activity into output.
Output	Outputs are the data or objects produced by the operation or activity.

Table 3-1: Generic PML constructs

3.2 Comparison of the five PMLs

Identifying the generic PML constructs from the five PMLs shows that each of them has different focus in modeling the dynamic world. Also, some of them are similar in constructs and rules while some are complimentary to each other.

3.2.1 Generic PML constructs in EPC

Identifying the generic PML constructs from EPC gives the following table:

PML construct	Explanation in EPC
Agent	The construct of agent in EPC is named "*Organization unit*". It is directly associated with the construct of "*Function*" in the notation, indicating who performs this function.
Resource	EPC does not capture the resources used in a process.
Activity	Construct of activity in EPC is named "*Function*" in its notation, which is used to express the action the agent executes.
Operation	The construct of "*Function*" is not differentiated into activity and operation in EPC. There is no way in EPC to find which function is an activity, and which is just an operation.
Event	EPC explicitly captures the event occurrence in a dynamic system. Events are created by a function and will trigger the next function if the active agent does not achieve its goal. Or some events may trigger another agent to start change. EPC uses events to control the chain of functions.
State	EPC does not explicitly keep track of each agent's state which keeps changing within a function. However, the state between two activities or operations is implied in the construct of "*Event*".
Data	EPC clearly present data by the constructs of "*Information in*" and "*Information out*". "Information in" represents the data that is only referred in a function, and "Information out" represents the data that might be newly generated or updated in the function.
Logical connector	The logical relationships between events and functions are expressed clearly: AND, OR, and Exclusive-OR.
Business rule	EPC does not include this construct in the model.
Input/output (object)	EPC does not consider the concrete objects that are input to the function or produced by the function.

Table 3-2: Identifying the generic PML constructs from EPC

Generally speaking, EPC method covers almost every necessary aspect in a process model except the concepts of resource, business rule and input/output object. Missing these concepts causes this method lack of information. Users are not able to know which participants are utilized as resources when an organizational unit completes a function, which rules the organizational unit must follow to complete a function, and what input objects are transformed into what output objects.

3.2.2 Generic PML constructs in IDEF0

Identifying the generic PML constructs from IDEF0 gives the following table:

PML construct	Explanation in IDEF0
Agent	The agent in IDEF0 is expressed in "Mechanisms" construct. The other part of Mechanisms is resource.
Resource	IDEF0 also represents the concept of resource in its "Mechanism" construct. This will cause confusion when people read the diagram and attempts to identify who performs the function and who is only utilized in the function.
Activity	The concept of activity is expressed by "Function" construct in IDEF0.
Operation	IDEF0 does not make difference between an activity and an operation. Both of them are all in "Function".
Event	In IDEF0, triggering events are hidden in the construct of "Controls", because the triggering events control which functions are to be performed. New events could be found in the construct of "Outputs" in IDEF0, as the new events are created by the function and transferred with other output objects together to the next stage. Therefore, an output of one function could be a control of another function.
State	There is not a state construct in IDEF0 method.
Data	Input data is part of IDEF0's "Inputs" construct, and output data is part of IDEF0 "Outputs" construct.
Logical connectors	IDEF0 does not provide rich symbols to represent the logical conjunctions of "AND", "OR", and "XOR". When two events are needed to trigger a function, its "Controls" construct only shows that the two events reach a function together, but it does not clearly show the logical relationship between the two events.
Business rule	IDEF0 puts the concept of business rule in the "Controls" construct. As "triggering event" is also expressed in the "Controls" construct, it is not clear to the model readers.
Input/output (object)	Input object is part of IDEF0's "Inputs" construct, and output object is part of IDEF0's "Outputs" construct.

Table 3-3: Identifying the generic PML constructs from IDEF0

From the above analysis, IDEF0 method has the following confusions in its constructs:

The construct of "*Controls*" includes (1) PML-business rule (2) PML-triggering event.

The construct of "*Mechanisms*" includes (1) PML-agent and (2) PML-Resource.

The construct of "*Inputs*" includes (1) PML-input data and (2) other physical input

objects.

The construct of "*Outputs*" includes (1) PML-output data, (2) PML-new events and (3) other concrete output objects.

3.2.3 Generic PML constructs in IDEF3

As there are two types of IDEF3 schematics (*Process Schematic* and *Object Schematic*) and they could be used in parallel in the process knowledge acquisition, the constructs of both schematics are analyzed together in the following table:

PML construct	Explanation in IDEF3
Agent	There is no agent construct in IDEF3 Process Schematics, but it has a construct "*Object*" in Object Schematics. An object could be any participant in the process that need to be modeled. The strength of this is that the model is not restricted to model the behavior of agents only. It also captures the states of non-agents. However, mixing agents with non-agents can lead to lack of clarity.
Resource	There is no resource construct in IDEF3. Every participant is modeled as an "Object" in IDEF3 Object Schematics.
Activity	Activity is named "*UOB* – Unit of behavior" in IDEF3.
Operation	There is no distinction between an activity and an operation in IDEF3.
Event	IDEF3 does not capture the information of events.
State	The Object Schematic that displays an object-centered view of a *single* scenario is called *Transition Schematic*. In this schematic, IDEF3 keeps track of an object's states change.
Data	No information of data is captured.
Logical connector	Each schematic in IDEF3 has its own set of logical connectors.
Business rule	Business rules are not mentioned in IDEF3.
Input/output (object)	Not available in IDEF3.

Table 3-4: Identifying the generic PML constructs from IDEF3

The two types of schematics in IDEF3 are normally used together to model complicated process requirements. It also includes a static model of the objects in the Enhances Transition Schematics.

3.2.4 Generic PML constructs in UML Activity Diagram

Identifying the generic PML constructs from UML Activity Diagram gives the following table:

PML construct	Explanation in UML Activity Diagram
Agent	The agent construct in Activity Diagram is expressed by swinlanes, each of which represents the behavior of one agent. When there is no swimlanes, the assumption is that the whole process is to be executed by the same agent, which does not need to be expressed in the diagram.
Resource	Not available.
Activity	The activity construct in Activity Diagram is "*Activity*".
Operation	It does not make any difference between an activity and an operation.
Event	Not available.
State	Not available.
Data	Not available.
Logical connector	Activity Diagrams use "Condition" to specify the conditions, which clearly shows the Exclusive-OR logical relationship. It also has its way to express "AND" and "OR". See Figure 3-1.
Business rule	Not available.
Input/output (object)	Not available.

Table 3-5: Identifying the generic PML constructs from UML Activity Diagram

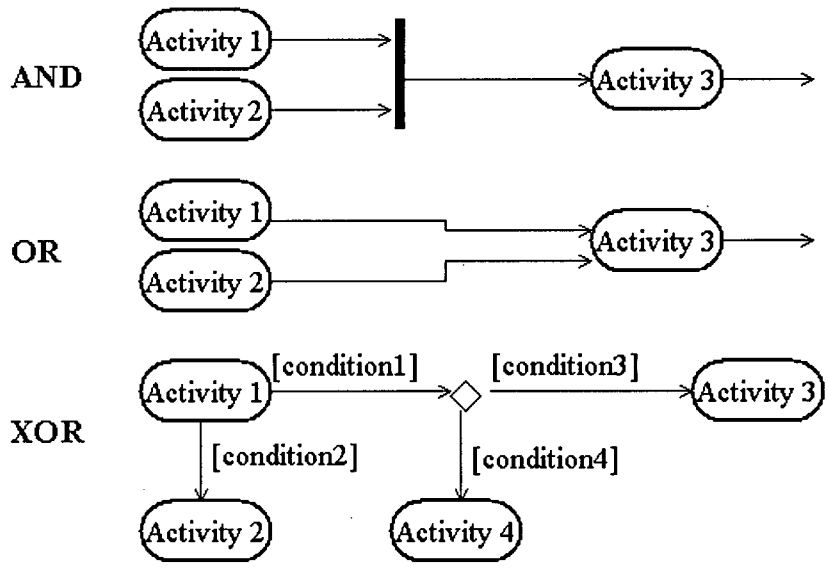


Figure 3-1: Logical conjunctions in UML Activity Diagram

Overall speaking, a UML Activity Diagram is comparatively simple to use but not as clear as other PMLs. An analyst could use Activity Diagrams to start the requirement analysis, and when the process is getting clearer, other PMLs with richer constructs could be considered.

3.2.5 Generic PML constructs in EDPM Method

EDPM Method is comparatively complete in terms of the number of constructs.

PML construct	Explanation in EDPM Method
Agent	The construct of agent in EDPM Method is "*Agent*".
Resource	Resource is expressed as "*Resource*" in EDPM Method.
Activity	The construct of activity is "*Activity*" in EDPM Method.
Operation	EDPM Method has the construct of "*Operation*" which enables an activity to be decomposed.
Event	The construct of event is "*Event*" in EDPM Method.
State	No available.
Data	Data is expressed as part of "*Input*" and "*Output*" in EDPM Method.
Logical connector	There are logical connectors as AND, OR and XOR in EDPM Method.
Business rule	Business rules are captured as "*Rule*" in EDPM Method.
Input/output (object)	Input and output objects are expressed as part of "Input" and "Output" in EDPM Method.

Table 3-6: Identifying the generic PML constructs from EDPM Method

3.3 Summary of the comparison

The differences among the five modeling languages in terms of generic PML constructs are illustrated in Figure 3-2.

PML construct	EPC	IDEF0	IDEF3	UML A.D.	EDPM
Agent	Org.Unit	Mechanisms	Object	By swimlanes	{agent}
Resource	N/A	Mechanisms	N/A	N/A	[resource]
Activity	Function	Function	UOB	Activity	Activity
Operation	Function	Function	UOB	Activity	Operation
Event	Event	Controls Outputs	N/A	N/A	Event e1
State	N/A	N/A	Object state	N/A	N/A
Data	Info in Info out	input output	N/A	N/A	<input> [output]
Logical Connector	∧ ∨ XOR	Not explicitly expressed.	& O X		
Business Rule	N/A	Controls	N/A	N/A	(rule)
Input/output object	Info in Info out	input output	N/A	N/A	<input> [output]

Figure 3-2: Comparison of the five PMLs

Chapter 4

THEORETICAL FOUNDATIONS

A conceptual model such as a business process model is created for the purpose of capturing the knowledge of a real-world domain. Therefore, the concepts that are used in the modeling languages should be able to completely and clearly represent the constructs in the real-world domain.

All the five PMLs introduced in previous chapters are used in describing business process models. However, each of them might have weakness in expressing the real-world system behaviors. Despite of the fact that most of their graphical symbols are intuitively defined and are easy to understand, they might have deficient, excessive, overloaded and redundant constructs that might lead to a vague model of the real-world process. And, this final model might either describe the business processes incompletely, or the representation might be not clear enough.

4.1 Objective of this chapter

The objective of this chapter is to examine the theoretical foundations of PMLs by the following steps:

- Introduce the well-defined BWW Ontology and its fundamental premise.
- Introduce the basic BWWP concepts, which are the process-related BWW Ontological concepts.
- Analyze the process model from the ontological point of view, and identify the ontology-based integrity rules.
- Map PML constructs to BWWP concepts, and find the PML-BWWP construct mapping rules to form a clear semantics for PMLs.

This chapter follows the notion of *ontological expressiveness* of modeling languages (Wand and Weber, 1993). Mapping from a modeling language to ontology is attempting to interpret the language constructs by ontological concepts, and mapping from ontological concepts to a modeling language is attempting to represent the real world concepts by modeling language constructs. The objective of this chapter is to complete the interpretation mapping analysis, which is to map PML constructs to BWWP concepts.

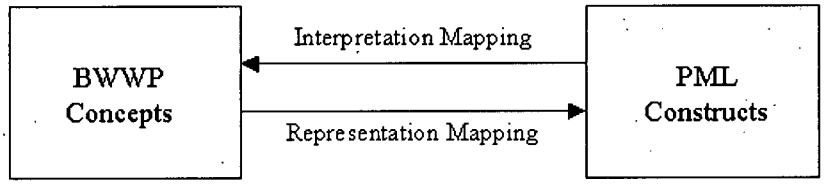


Figure 4-1: Two-way mappings

As EPC modeling language is widely used in the field of business process modeling, its graphical symbols are adopted as a representation for a generic PML in most parts of the following discussions. However, due to the fact that some general constructs such as *resource, input/output concrete object* and *business rule* are missing in this language, it will not be used when discussing these constructs.

4.2 Choosing BWW Ontology

The intention of the thesis is to build a general model of a process. To do this, we use the theoretical foundation of ontology. Ontology is a well-established theoretical domain within philosophy dealing with models of reality. It indicates what exists in the real world, and what rules are followed for things to interact in the real world. Wand and Weber have adopted an ontology presented by Mario Augusto Bunge (1977, 1979) and applied it to the modeling of information systems (Wand and Weber, 1989, 1990, 1993). The thesis will refer to this ontology as the BWW Ontology. Bunge's ontology is chosen as the foundation for our work due to the following strengths:

- It is well formulated in terms of set theory.
- It has been applied in a number of studies related to information systems modeling and object-oriented concepts, thus providing a benchmark for evaluating other models (Green and Rosemann, 2000; Wand and Weber, 1993).
- It has been used to suggest an ontological meaning to object concepts (Wand, 1989).
- It has been used to generate predictions about conceptual models that have been corroborated empirically (Gemino, 1999; Weber and Zhang, 1996).

4.3 The fundamental premise of BWW Ontology

The fundamental premise on BWW Ontology is that any modeling language must be able to represent all relevant concepts in the real world that might be of interest to users (Wand and Weber, 1993, 1995). Otherwise, the modeling language is regarded as incomplete from ontological point of view. This might, in turn, lead to incomplete or ambiguous models. If the model is incomplete, the final computerized information system might not adequately reflect the real-world domain. However, even if the model is complete, there still might exist the ontological clarity problems in the model.

Two major situations that might occur when a modeling language is analyzed (Wand and Webber, 1993) are displayed in following:

1. *Ontological completeness*: For each ontological concept in the investigated domain, if there is at least one construct in the modeling language that could correspond to it, the language is regarded as "ontologically complete".

2. *Ontological clarity:* For the ontologically complete languages, there might exist one or more of the following deficiencies from the perspective of ontological clarity:

- *Construct Excess:* There exists at least one element in the modeling language that does not have a counter-part in ontology.

- *Construct Overload:* There exists at least one element in the modeling language that has multiple interpretations in ontology.
- *Construct Redundancy:* There exist at least two elements in the modeling language that share one ontological interpretation.

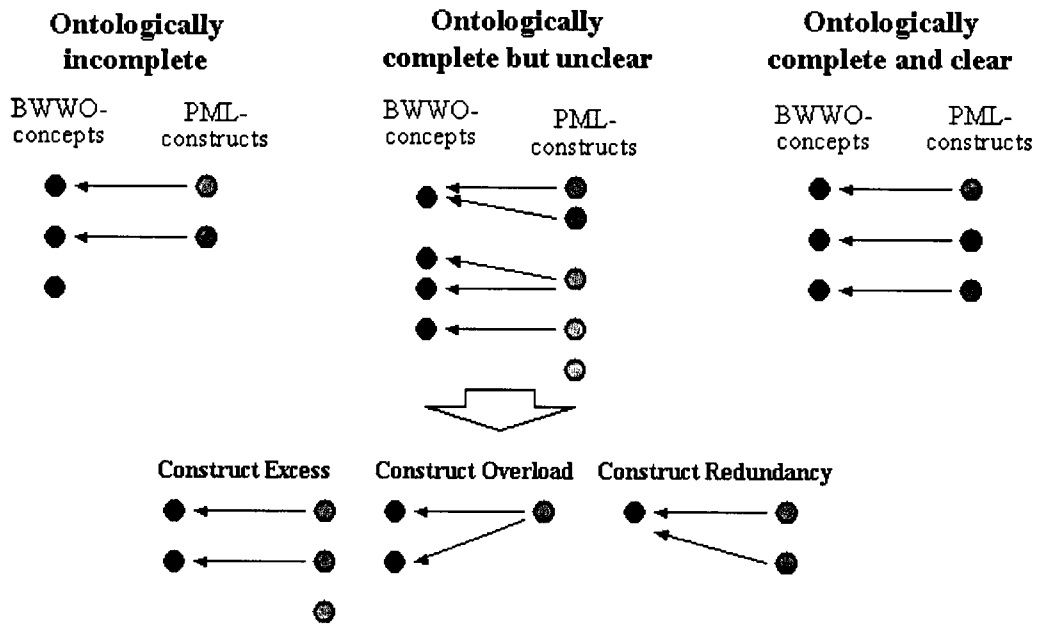


Figure 4-2: Ontological completeness and ontological clarity

Ontologically incomplete models will lead to inadequate information systems which may partially represent the reality, while ontologically unclear models could generate faulty information systems which might be meaningless, ambiguous or redundant.

4.4 Original BWW Ontological constructs

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

Category	Original Construct of BWW Ontology
Static model of an individual	1. Thing, composite thing
	2. Property, intrinsic property and mutual property, hereditary property and emergent property
	3. State, conceivable state space, state law
	4. Class, kind, natural kind

- To be continued in the next page

Category	Original Construct of BWW Ontology
Dynamic model of an individual	1. Event, conceivable event space 2. Transformation, transformation law 3. History
Static model of a system	1. Coupling 2. System 3. System composition, system environment, system structure, system decomposition, subsystem, level structure
Dynamic model of a system	1. Stable state and unstable state 2. Internal event and external event 3. Well-defined event and poorly defined event

Table 4-1: Original constructs of BWW Ontology

There ontological meaning will be explained in section 4.5.

4.5 Process-related BWWP constructs

Some of the original BWW Ontological concepts are closely related with the concept of process, and those process-related BWW ontological concepts are referred to as *BWWP* constructs. The BWWP constructs include the original BWW Ontological constructs of *thing (simple thing and composite thing), property (intrinsic property and mutual property, hereditary property and emergent property), state (stable state and unstable state), event (internal event and external event), transformation,* and *law (state law and transformation law).* The new concepts added in BWWP construct set are: *actor, non-actor, actuator, propagator,* and *process.*

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

The world is made up of substantial *things*. Things possess *properties* that are either *intrinsic* or *mutual*. A property could be *intrinsic* if it is possessed by the thing itself (e.g. color of a car), or it could be *mutual* if it is commonly possessed by two or more things (e.g. distance between two cars).

Things can combine to form a *composite* thing. Composite things possess *emergent* properties that are not possessed by any component (e.g. the speed of a car is the emergent property of a car that is not possessed by

any components of a car). The properties that are only possessed by components in a composite thing are *hereditary* properties (e.g. the horsepower of car engine).

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

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

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

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

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

For example, let a thing be in a stable state s1. An interaction with other things can change its state, say to s'. We shall call <s1, s'> an external event. Now, if the state s' is unstable, then according to the assumptions the thing will keep changing its state until it reaches a new stable state s2. Note, s2 is actually defined by the lawful transformations of the thing. The event <s', s2> is an internal event. The internal event does not depend on the interaction with other things but only on the lawful transformations that are intrinsic characteristics of the thing itself (Wand and Weber, 1990b).

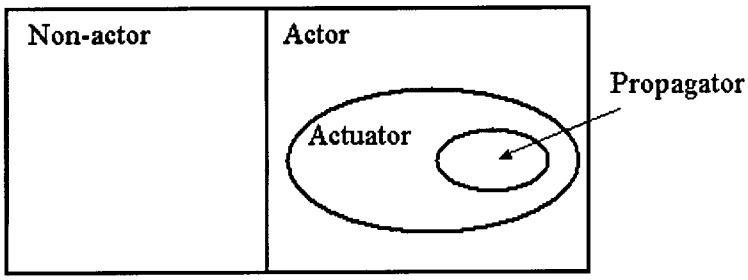
Besides the original BWW ontological concepts introduced above, some new concepts are added in BWWP. They are described as follows:

A thing could be differentiated into an *actor* or a *non-actor*. An *actor* is a thing that changes the state of itself or at least one other thing (actor/non-actor). A *non-actor* is a thing that does not change the state of any thing including itself. The state of a non-actor can only be changed by an actor. For example, when a clerk walks, her state is changed by herself and she is an actor. When a clerk uses a telephone set to make a call,

the clerk changes the state of herself and also the state of the telephone set, so the clerk is an actor. The telephone set is a non-actor as it is not able to change the state of itself, nor is it able to change any other things. Its state could only be changed by the clerk.

An actor could be an *actuator* if it changes the state of at least one other thing (actor or non-actor). For example, if the clerk just walks, she is not an actuator because she does not change the state of any other things. When she makes telephone calls, she is an actuator because she changes the state of at least one other thing: the telephone set.

An actuator could be a *propagator* if it changes the state of at least one other actor. For example, if the clerk makes a phone call and a worker is activated by her to move a forklift, the clerk is a propagator because she changes the state of another actor: the worker.



BWWP-Thing

Figure 4-3: BWWP constructs related with `thing`

A process involves a set of actors and non-actors who interact one another. The process is *activated* when at least one actor is changed from stable state to unstable state. One or more events could trigger a process. The process is *completed* when all actors and non-actors are in stable states.

4.6 Ontological model of a process

An ontological model of process is the description of a business process using ontological concepts rather than PML concepts.

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

4.6.1 A system of four things

To capture the essentials of an ontological process model, an ontological analysis is needed for a generic scenario of a system that is composed of several things (including actors

and non-actors, simple or composites). The ontological analysis here is performed by considering the state-change happened to each thing in the system. The scenario being used here is described in Figure 4-4. For simplicity, we do not distinguish the concepts between *property* and *attribute*.

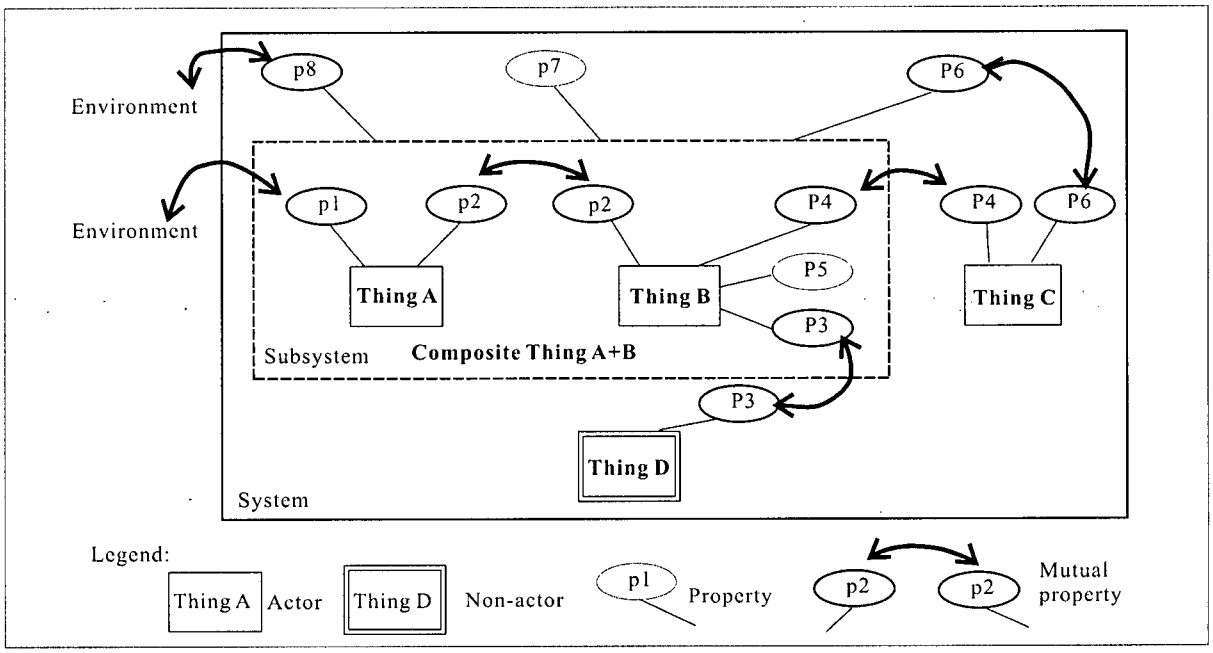


Figure 4-4: A system of four things

There are four individual things: A, B, C, and D. A, B and C are actors, and D is a non-actor which only interacts with B. A has properties: p1, and p2. B has properties: p2, p3, p4, and p5. The mutual property of A and B is p2. C has properties p4, p6. The mutual property of B and C is p4. Non-actor D has property p3. It does not have any other properties. The mutual property of B and D is p3. Composite thing A+B is also an actor, it has emergent properties p6, p7 and p8. P6 is the mutual property of C and A+B.

4.6.2 Ontological analysis

Figure 4-5 displayed the interactions between the four things from BWWP ontological perspective. For clarity, all the stable states are labeled as *s1, s2, s3, ...* and all unstable states are labeled as *u1, u2, u3, ...* Based on the definitions of BWWP concepts, the

analysis in this section attempts to find the ontological principles of thing's interaction in a process, and demonstrate them by the example. The ontological principles found in this section will be summarized as a set of ontology-based integrity rules later. The detailed analysis is presented as follows:

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

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

(2) The first change in the system is triggered by the system environment. This change is effected by a thing in the environment to change the mutual property of a thing in the system.

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

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

Demonstration: A thing in the environment interacts with actor A by changing the value of mutual property p1. Actor A interacts with actor B by changing the value of mutual property p2. Actor B interacts with non-actor D by changing the value of mutual property p3. Actor B interacts with actor C by changing the value of mutual property p4. Actor C interacts with composite actor A+B by changing the value of mutual property p6. Composite actor A+B interacts with a thing in the environment by changing the value of mutual property p8.

(4) When an actor modifies its intrinsic properties, it does not affect any other things but only changes the actor itself.

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

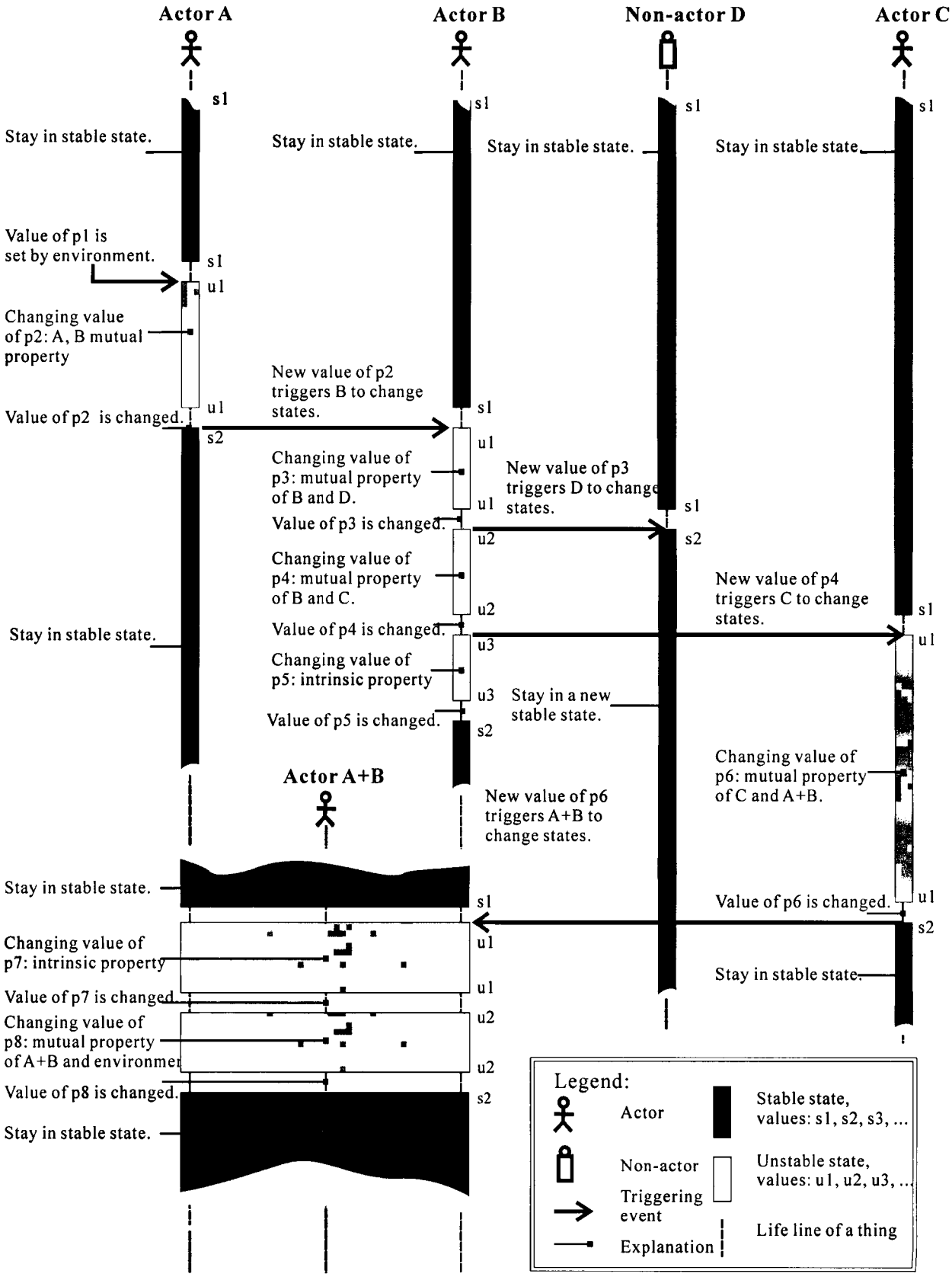


Figure 4-5: Ontological analysis to the four things' interaction

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

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

(6) A non-actor is not able to change any thing including itself. Therefore, all its properties are mutual with actors, and its intrinsic properties are of no interest in the model. Its state-changes are always caused by actors through changing the values of the mutual properties, and they are always from a stable state to another stable state. [2]

Demonstration: Non-actor D does not have any intrinsic properties. D changes from a stable state to another stable state.

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

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

(8) A composite thing and its components are observed from different standpoints.

Demonstration: When composite actor A+B is changing, we have no idea what happens in its components A and B. Similarly, when actor A and B are changing, we have no idea what happens in composite thing A+B.

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

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

[2] Note: If we are interested in state transitions of a non-actor, we could look at it as an actor. In this case, its intrinsic properties can be included in the model.

4.6.3 State curves

A state curve diagram in Figure 4-6 is used to analyze the concepts of *external event*, *internal event* and *transformation* in a process.[3] Like in the previous section, a demonstration in the example will follow the principles found in the analysis.

(1) In order to change, things must interact with its environment. Therefore, each thing must be activated by an external event. An external event occurs by virtue of an interaction with the thing's environment.

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

(2) The before-state of an external event is always stable. An actor has its external event end with an unstable state so that it could keep changing itself or another thing. A non-actor has it external event end with a stable state because it is unable to change any thing including itself.

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

(3) Internal events occur by virtue of transformations in a thing. For an actor, all the subsequent events following an external event are internal events. As an actor has at least one transformation to reach stable state again, there is at least one internal event during an actor's change. Similarly, as a non-actor is not possible to have transformation, it does not have any internal events.

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

---

[3] Note: A state curve shows how an actor changes its state over time. The x-axis indicates the time, which is considered discrete when a state-change needs to be shown. The y-axis indicates the value of properties (i.e., state). For an unstable state, the value on y-axis varies in some extent showing that the state is not as stable as a straight line. However, the variance on y-axis is regarded as "no change" if it does not surpass a threshold. When the value on y-axis varies over a threshold, it is regarded as a state-change. For a stable state, the value on y-axis does not have any variance.

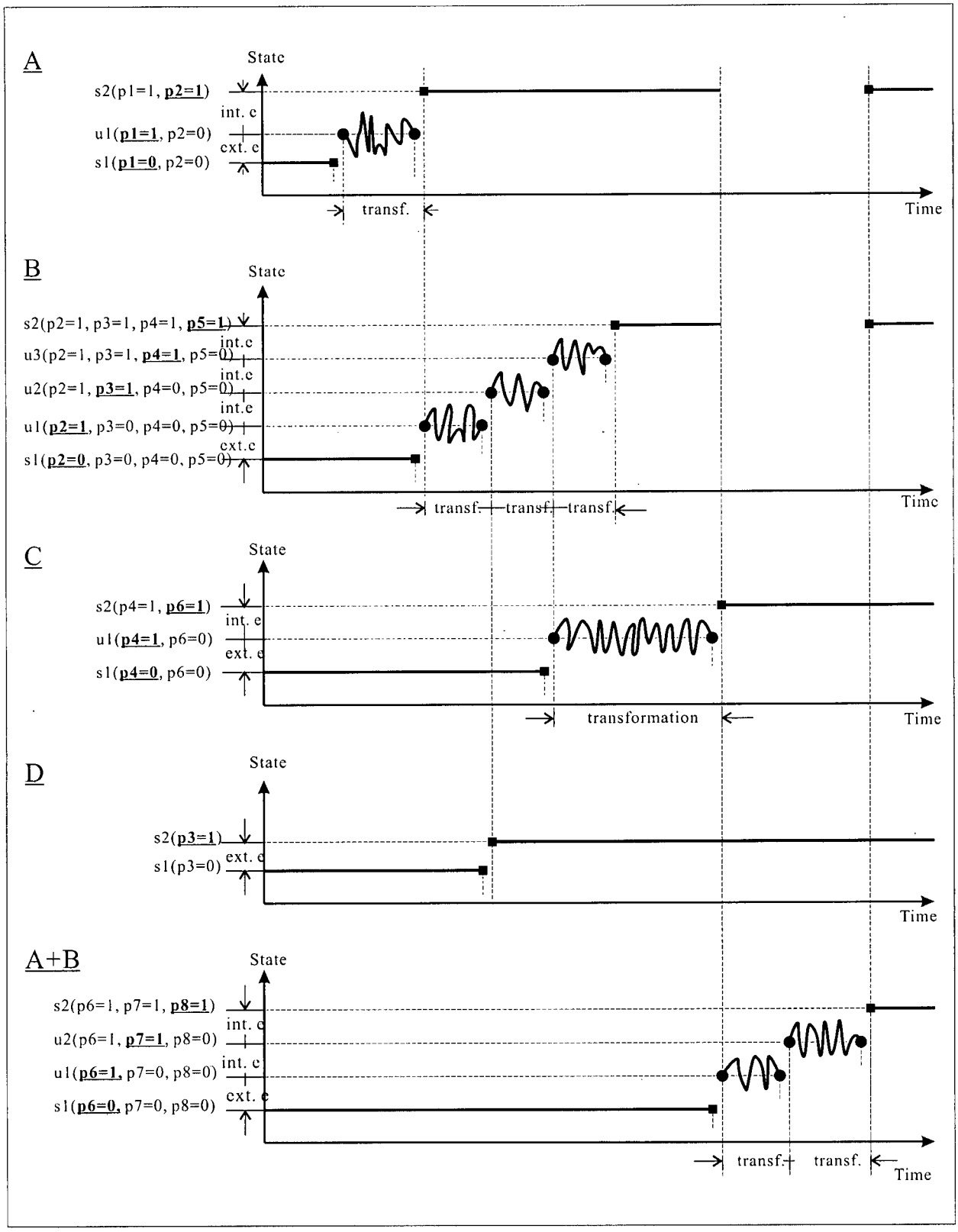


Figure 4-6: State curves of each thing

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

Demonstration: A, B, C and A+B undergoes transformations D doesn't.

(5) After an actor starts to change, whenever it modifies one property, it undergoes one transformation. When an actor modifies all its properties, it undergoes one sequence of transformations.

Demonstration: After A starts to change (p1 is modified by the environment), A modifies another property p2 and undergoes a transformation. It also undergoes a sequence of transformation that comprises only one transformation. After B starts to change (p2 is modified by A), B modifies p3, p4 and p5. Each modification leads B to a transformation. When all of p3, p4 and p5 are modified, B undergoes a sequence of transformations. Same to C and A+B.

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

Demonstration: The only transformation of A starts from an unstable state: u1. It ends with a stable state s2. All the transformations of B start from an unstable state: u1, u2 and u3. They end with different states of u2, u3 and s2 respectively.

(7) The concept of external event, internal event and transformation also applies to composite thing.

4.7 PML-BWWP construct mapping rules

After doing a detailed ontological analysis to a business process, the counterparts of PML constructs in BWWP can be described. In the following, we attempt to look at a process from both BWWP and PML perspectives, and find the mapping rules between them. To avoid confusions between the two types of concepts, the prefixes of BWWP- and PML- are used.

The PML concepts will be analyzed in the following order: *Agent, resource, data, event,*

state, activity/operation, logical connector, business rule, input object and output object.

4.7.1 PML-Agent

A PML-agent is normally defined as "a person or an organizational unit who executes the process activities." There are two kinds of participants in a process. The active participants who have the ability to take actions in the process are considered as agents, and the passive participants whose states are totally affected by agents' behaviors are PML-non-agents. For example, the agents could be human beings and organizations such as a company, a department, a political parties and a team, and non-agents could be resources used by agents during the process such as a document, a computer program, and some tools. The behavior of agents is the main focus of PMLs.

The construct in BWW Ontology that defines the entity to be modeled is "thing", which is defined as a substantial individual that is perceived to exist physically. A PML-agent could be either a person or a material organizational unit, which is also composed of persons and other substantial individuals. For example, an organization such as a company, a political party, a department, a team, etc. is the object on behalf of which that human being participates in the process.

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

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

In the WaterSystem example (the complete example is presented in Appendix I), the agents are "Sales dept.", "Warehouse dept.", "Manufacturing dept.", "Purchase dept.", and "Accounting dept.". All of them are substantial entities, and their actions cause other

participants change.

4.7.2 PML-Resource

A PML-resource is a substantial thing that is utilized or referenced by an agent to assist the execution of the activity. The difference between a PML-*resource* and a PML-*agent* is that a resource is unable to change the state of any participant including the resource itself.

The state of a resource could only be changed by one or more agents. If its state is temporarily changed and will be changed back when the activity is over, this resource is called "*carried forward*" resource. In the WaterSystem example, in the process of "Arrival" when the agent "Warehouse dept." performs the activity "Add to warehouse", it utilizes a forklift as a tool. This forklift will be changed back to its previous state when the activity is over. Therefore, it is a "carried forward" resource. On the other hand, if the state of a resource is changed permanently during the activity, the resource is called "*consumable*" resource. In the example, in the process of "Assembly" when agent "Manufacturing dept." executes the activity "Assemble product", it consumes electric power.

In BWWP concepts, a *non-actor* is defined as a thing that is not able to change the state of any thing including the non-actor itself. Hence, we have the following rule:

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

4.7.3 PML-Data

In PMLs, data is a special kind of resource that also can be utilized, referenced or modified when an agent is performing a task. But the difference is that data is conceptual rather than material. Data would be a representation of the states of other substantial things. For example, the data of a product's diameter could be regarded as the value of a property "diameter" of the thing "product".

An agent who involves in an activity may access this state information by sharing the

representation in some way between the agent and the data owner. This access is the interaction between two physical things: agent and the represented thing. The changes to data, in turn, could represent the outcomes of interactions where the represented thing changes states. For example, if an agent "worker" changes the data "product file", it represents an outcome of interaction between a worker and a product that the worker has changed the product's properties such as "diameter" or "length", which means, the state of product is changed.

However, the represented thing behind data is not of interest to most PMLs. We do not discuss what data represents in this paper, and a deeper meaning of data could provide new insights in future research.

Mapping rule 3: PML-data maps to BWWP-state of the represented thing.

Data could also be differentiated into two types:

- *Read only data* – the data that do not change during the activities and can be reused many times without modification by the same or different agents. In this case, the represented thing is not changed during the activity.
- *Read and write data* – the data that could be read by an agent and modified later on by the same agent during the execution of a particular activity. This reflects that the state of the represented thing is changed.

4.7.4 PML-Event

A PML-event is a notable *occurrence* at a particular point in time, which causes an agent or a resource to start change or continue to change. It is the cause or the trigger for an activity or an operation to be invoked.

From ontological perspective, BWWP-event is defined as a *state-change* of a thing caused by the interaction with some things in the environment, or by the transformations of

the thing itself.

It is clear that there exists a difference between PML-event and BWWP-event concepts. PML-event is actually the starting point of a thing's state-change while BWWP-event is the state-change itself. Therefore, they cannot be simply mapped to each other.

When attempting to find the ontological counterpart for PML-event, it is convinced that a PML-event which occurs to an agent is actually a vector of values of the agent's properties at the time when an expected fact occurs, which expresses the implication of BWWP-state concept. Hence, we have the following mapping rule:

Mapping Rule 4: PML-event maps to BWWP-state of an actor.

The PML-event that invokes next PML-operation could be named "*triggering event*". Similarly, the PML-event which is the result of a PML-operation could be named "*resulting event*".

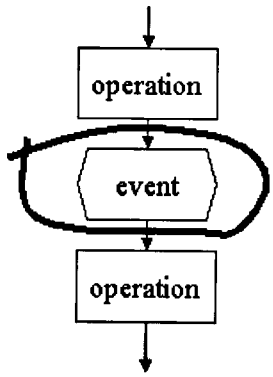


Figure 4-7: BWWP-state in PML

There are two types of PML-events:

- Type one: For a PML-agent, when one activity (or a sequence of operations) is completed, it reaches a stable state and will never change until a new event arouses it. The PML-event at the end of an agent's activity could be mapped to BWWP-stable state of the actor (PML-agent).

Mapping rule 5: The PML-event at the end of an agent's activity maps to BWWP-stable state of

the agent (BWWP-actor).

In "Order Processing" process of the WaterSystem example, the event "Order closed" is the BWWP-stable state of the agent "Sales dept.", and the event "Paid vendors" is the BWWP-stable state of agent "Account payable dept.".

- Type two: If the PML-event triggers an agent to change, it means that the agent is in unstable state and its change starts. Similarly, the PML-event which locates between an agent's two consecutive PML-operations (if any) is also an unstable state, because there is an operation following it and the agent will continue to change to another state later on.

Mapping rule 6: An agent's all PML-events except the final event map to BWWP-unstable states of the agent (BWWP-actor).

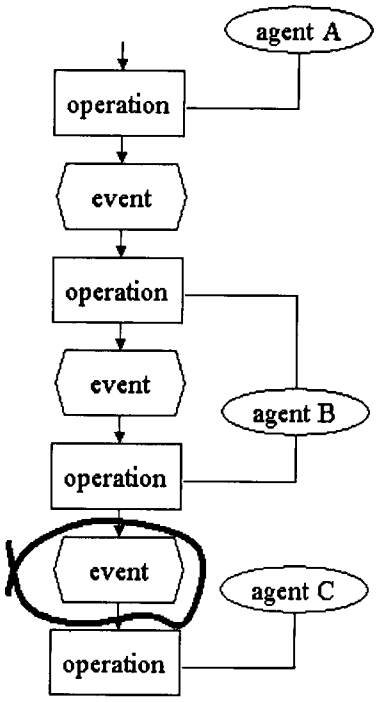


Figure 4-8: BWWP-stable state of actor B

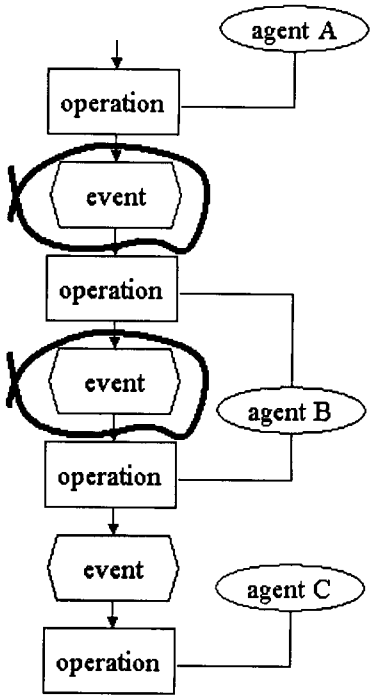


Figure 4-9: BWWP-unstable state of actor B

In the "Order Taking" process of the WaterSystem example, the event "Order placed" is a BWWP-unstable state for agent "Sales dept.". In the "Assembly" process, the event "Assembly done" is a BWWP-unstable state for agent "Manufacturing dept.". All these unstable states will keep changing till the agent reaches a stable state.

4.7.5 The sequence of "event→operation→event" in PML

The following question will be: what should be the counterpart for BWWP-event in PML.

As mentioned before, a BWWP-event is defined as a state-change of a thing, which is normally expressed as a pair of states <*s1, s2*>.

In PMLs, when an agent need to change states, it follows the procedures below: (1) A PML-event occurs. (2) A PML-operation starts. (3) A new PML-event occurs. The first PML-event gives the agent's state before the agent changes, and the second PML-event gives the agent's state after it changes. The PML-operation in between indicates the task that the agent need to do in order to change it state from one to another. Therefore, the sequence of "event→operation→event" in PMLs describes the "state-change" of the agent through an PML-operation, and it can be mapped to BWWP-event concept in ontology.

Mapping Rule 7: The sequence of "event→operation→event" in PML maps to BWWP-event.

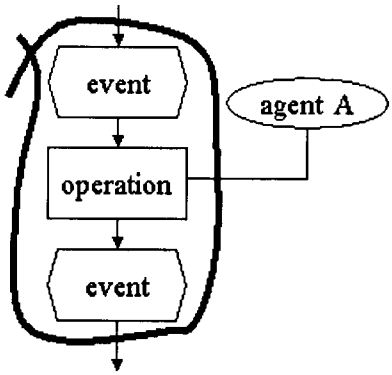


Figure 4-10: BWWP-event in PML

Here, we need to further differentiate the concepts of BWWP-external event and BWWP-internal event.

- A BWWP-external event to an agent describes an agent's state-change that is caused by the interaction with a thing in the agent's environment. Hence, the external event must

represent the state-change of an external thing.

Mapping rule 8: The sequence of "event→operation→event" which belongs to a thing in an agent's environment and activates the agent to start change is mapped to BWWP-external event of the agent (BWWP-actor).

In "Order processing" process of the WaterSystem example, the sequence "Order approved→Arrange payment→Payment made" which belongs to the state-change of another agent "Sales dept." is an external event for the agent "A/R dept." to start "Collect payment" operation.

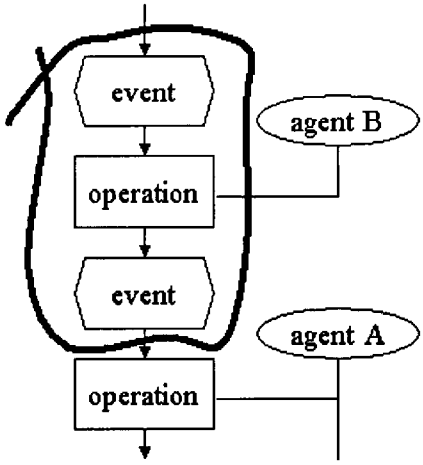


Figure 4-11: BWWP-external event for actor A in PML

■ A BWWP-internal event to an agent describes an agent's state-change that is achieved by its lawful transformation. Hence, the internal event must represent the state-changes of the agent itself, which are accomplished through its operations.

Mapping rule 9: For each sequence of "event→operation→event" of an agent, if the "operation" is performed by the agent, then the sequence maps to BWWP-internal event of this agent (BWWP-actor).

In the "Order taking" process of the WaterSystem example, the sequence of "Product available→Approve order→Order approved" is an internal event for agent "Sales dept.".

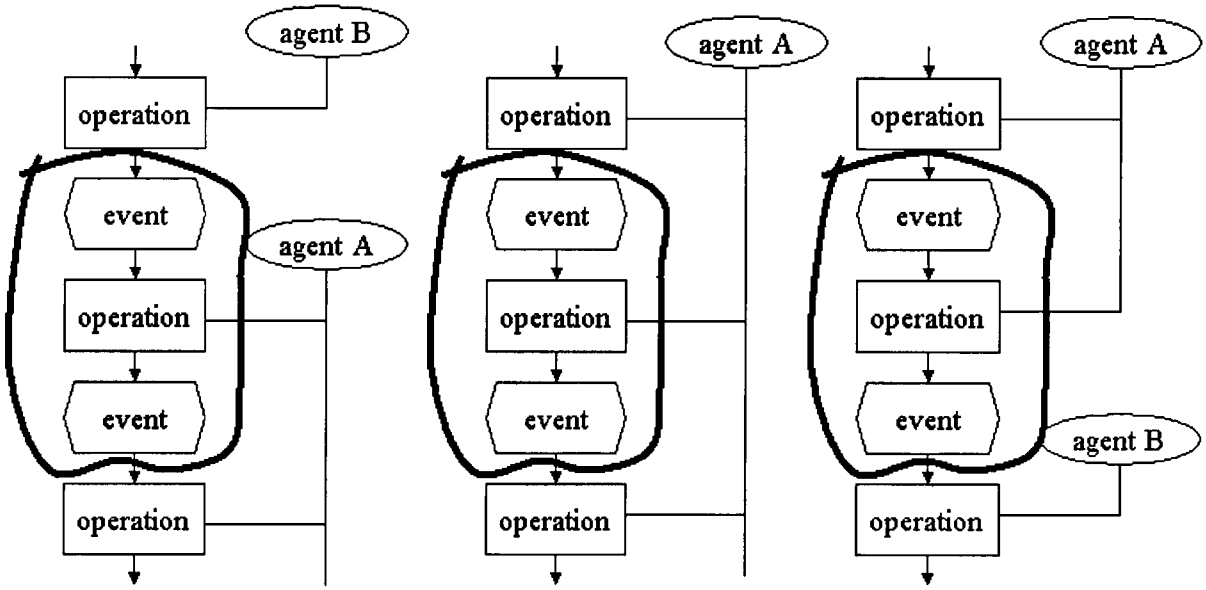


Figure 4-12: BWWP-internal event for actor A in PML

4.7.6 PML-State

A PML-state represents the values of all properties of a participant at a particular point of time when a PML-event occurs.

Similarly, a BWWP-state is defined as the vector of values for all attribute functions (or properties) of a BWWP-thing. It could be interpreted as a complete assignment of values to all n attribute functions $Fi\ (A,\ t)$ in the function schema, where n is the number of attribute functions, i is from 1 to n, A is the agent, and t is the time when triggering event occurs.

Therefore, we have the following mapping rule:

Mapping Rule 10: PML-state maps to BWWP-state.

As both PML-*event* and PML-*state* have the same ontological counterpart of BWWP-state, most PMLs only use one of the constructs in order to avoid ontological redundancy.

In the "Purchasing" process of the WaterSystem example, the agent "Purchasing dept." has the following properties:

Property p1: Purchase plan situation. Possible value: Purchase needed/ not needed.

Property p2: Order to vendor situation. Possible value: Order placed/Order not placed.

There are two states before and after the activity "Place order with vendors":

State s1: p1 = Purchase needed, p2 = Order not placed;

State s2: p1 = Purchase needed, p2 = Order placed.

4.7.7 PML-State of a joint agent

There might be a situation where two or more agents are involved in one PML-operation. In this case, the PML might need to capture the states of the joint agent that is composed of the individual agents. From BWW Ontological point of view, this joint agent is the BWWP-composite thing, whose states are the values of emergent properties that do not belong to any of the components.

Mapping rule 11: PML-joint agent involved in a PML-operation could be mapped to BWWP-composite thing, and PML-state of the joint agent could thus be mapped to BWWP-state of the composite thing.

For example, an activity is executed by a joint agent "A" that is composed to two individual agents "A1" and "A2". The states that PML need to keep track of are the BWWP-states of BWWP-composite thing A, which gives the values of the emergent properties of BWWP-composite thing A. These emergent properties belong neither to A1 nor to A2.

4.7.8 PML-Activity and PML-Operation

PML-activity and PML-operation represent the actual tasks that an agent carries out in order to change the states of itself or other participants. A PML-activity is an atomic action in a process, which means once it starts it will eventually reach an end. A PML-operation is a component of an activity. A PML-activity need to be divided into operations so that the agent's intermediate states could be captured. These intermediate states might trigger other agents to change, or might cause the agent itself to continue its change.

In this sense, both a PML-activity and a PML-operation can be regarded as a BWWP-transformation, which represents an agent's state-change over time. The difference is, an operation could be any transformation (which always starts from an unstable state) and an activity is a sequence of operations where the first follows a state-transition from stable to unstable (i.e. an external event), and the last ends with a stable state.

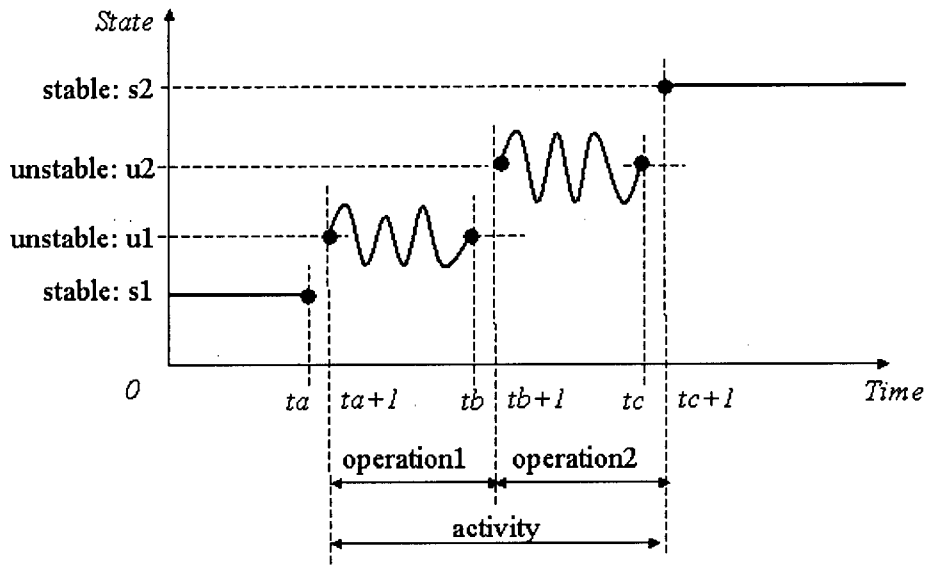


Figure 4-13: A state curve of two operations

Figure 4-13 presents a state curve of an agent, which start to change at time *ta+1* and its change stops at time *tc+1*. It is clear that the agent undergoes two operations: the first changes the state from unstable to unstable, and the second from unstable to stable. The whole sequence of operations forms an activity, where the first operation follows a transition from stable state s1 to unstable state u1 (this change represents an external event <s1, u1>), and the last operation ends with a stable state.

Mapping Rule 12: PML-operation maps to BWWP-transformation.

Mapping Rule 13: PML-activity maps to a sequence of BWWP-transformation, where the first follows a state-transition from stable to unstable (i.e. an external event), and the last ends with a stable state.

Therefore, PML-operation and PML-activity represent two different ontological concepts,

and A PML-operation is not simply a component of a PML-activity.

In the "Order processing" process of the WaterSystem example, the agent "Warehouse dept." undergoes three PML-operations: "Check order in warehouse", "Ship products" and "Rearrange warehouse". The sequence of the three operations could be regarded as one PML-activity named "Arrange shipment".

4.7.9 PML-Logical connector

A rule in a business process maps to the BWWP-law concept. A rule can be expressed by a logical connector or a business rule in general.

The logical connectors in PMLs are used to define the order of activities in a process by expressing the logical relationships between PML-events and PML-activities (such as in EPC, IDEF3 Transition Schematics and EDPM Method). Some PMLs use logical connectors to express the relationships of activities only (such as IDEF3 Process Schematics and UML Activity Diagram), while other PMLs (such as IDEF0) do not explicitly provide the detailed logical connectors.

Taking the most complicated case into consideration, the PML-logical connectors attempt to capture the causal relationships between events and activities in the following three categories:

(1) One PML-event → logical connector → two or more PML-operations

(2) Two or more PML-events → logical connector → one PML-operation

(3) One PML-operation → logical connector → two or more PML-events

The detailed ontological analysis could be seen in Appendix II "Ontological analysis of PML-logical connectors".

As a logical connector defines the rules between activities or operations, it might map to the concept of BWWP-*law*.

BWWP-law has two types: *state-law* and *transformation-law*. A BWWP-state law restricts the states of a thing to a subset that is deemed lawful because of a natural law or human law (here, when considering business process, the natural law could be some business rules). From PML perspective, the end of an PML-operation could possibly initiates *all of*, *one of* or *either of* the two or more different PML-events according to the defined logical relations of "AND", "OR" or "XOR". These PML-events restrict the agent's states to a subset rather than all the possible values, which is regarded as lawful states based on the given rules.

Mapping rule 14: The sequence of "operation→logical connector→event" in PML maps to BWWP- state law.

On the other hand, when a PML-event is going to invoke two or more operations according to the logical relations of "AND", "OR" or "XOR", the two or more alternative operations are deemed lawful in the process, and any other operations are not allowed. This restricts the possible alternative operations (which maps to BWWP-transformation) in a process. As a BWWP-transformation law determines all the transformations a thing can undergo, we have the following mapping rule:

Mapping rule 15: The sequence of "event→logical connector→operation" in PML maps to BWWP- transformation law.

4.7.10 PML-Business rule

There are two different mechanisms that PMLs use to capture rules in a business process: They use the concept of PML-*logical connector* for rules between operations, and the concept of PML-*business rule* for rules within an operation.

The construct of "*controls*" in IDEF0 and the construct of "*rules*" in EDPM Method are both PML-business rule concepts. In the "Assembly" process in the WaterSystem example,

the "Product assemble guide" is a business rule for activity "Assemble product" of agent "Manufacturing dept.".

A PML-business rule could be considered as a set of PML-logical connectors when used in executing an operation or activity. The concept could be eventually decomposed into PML-logical connectors when decomposing an operation or activity.

Mapping rule 16: PML-business rule maps to BWWP-law.

4.7.11 PML-input object and PML-output object

As a PML-input object represents the concrete object which is transformed by an activity or operation into output object, and a PML-output object is the concrete object which is produced by the activity or operation, both the PML concepts could be mapped to BWWP-non-actor concept.

Mapping rule 17: PML-input object and PML-output object map to BWWP-non-actor.

4.8 Summaries of the two outcomes

We have two findings in this chapter: one is a set of *PML-BWWP construct mapping rules,* and another is a set of *ontology-based integrity rules*. They are summarized in the following parts for the easy use in the following chapters.

4.8.1 Summary of the PML-BWWP construct mapping rules

The PML-BWWP construct mapping rules are summarized in Figure 4-14. This mapping summery will be used in the next chapter to evaluate the five PMLs.

BWWP concepts	*PML constructs*
Thing	
Actor	Agent
Non-actor	Resource
	Input/output object
State	
State	Event
	State
Stable state	Agent's final event
Unstable state	Agent's all events except the final event
Representation of a thing's state	Data
Event	
Event	A sequence of "event – operation – event"
External event	A sequence triggering the agent in env.
Internal event	All other sequences
Transformation	
A sequence of transformations	Activity
Transformation	Operation
Law	
Law	Business rule
State-law	Operation – logical connector – event
Transformation law	Event – logical connector – operation

Figure 4-14: PML-BWWP construct mapping rules

4.8.2 Summary of the ontology-based integrity rules

The ontology-based integrity rules have been discussed in section 4.6 "Ontological model of a process". These rules form the foundation of an algorithm on how to construct a theory-based process model rather than a process model built on observation and empirical experience. The algorithm will be presented in Chapter six.

The rules are summarized as follows:

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

(2) All things interact by changing the value of mutual properties.

(3) The first change in the system is triggered by the system environment. This change is effected by a thing in the environment to change the mutual property of a thing in the system.

(4) When an actor modifies its intrinsic properties, it does not affect any other things but only changes

the actor itself.

(5) After a thing starts change, it modifies all its properties and eventually reaches stable.

(6) All properties an non-actor are mutual with actors, and its intrinsic properties are of no interest in the model. Its state-changes are always caused by actors, and they are always from a stable state to another stable state.

(7) A composite thing and its components are observed from different standpoints.

(8) The process is completed when the last thing stops changing and all things are in stable state.

(9) In order to change, each thing must be activated by an external event.

(10) The before-state of an external event is always stable. An actor has its external event end with an unstable state. A non-actor has it external event end with a stable state.

(11) All the subsequent events following an external event are internal events for an actor. There is at least one internal event during an actor's change. A non-actor does not have any internal events.

(12) After an actor starts to change, whenever it modifies one property, it undergoes one transformation. When an actor modifies all its properties, it undergoes one sequence of transformations.

(13) An actor undergoes a transformation when its state is changed from one to another over time. A non-actor does not undergo any transformations.

Chapter 5

EVALUATION OF PMLS

The observations on the five PMLs in Chapter two and three show that these PMLs have problems in their expressiveness. As the PML-BWWP construct mapping rules obtained in Chapter four form clear semantics for PMLs, the five PMLs can be evaluated in terms of ontological completeness and ontological clarity based on the construct mapping rules. Identifying possible ontological deficiencies of the current PMLs can help to clarify the basic concepts for the process modeling methodology.

5.1 Evaluation of EPC

Figure 5-1 shows the details in mapping EPC constructs to BWWP concepts and the generic PML concepts. Several observations need to be addressed here:

- **Missing the concept of PML-resource, input/output object**

PML-resource and input/output object concepts represent a non-actor in BWWP concepts. They are used to model the participants whose states are changed only by actors. In the real world, a PML-resource is a thing that is utilized or consumed by an agent during the agent's state-change, a PML-input object is a thing that is transformed by the activity or operation into output object, and a PML-output object is a thing that is produced by an activity or operation. It is possible for an agent to change its state without utilizing a resource or an input object and without producing any output objects as well. For example, without using any tools, a clerk can ask a worker start assembling products. Therefore, a modeling language does not necessarily need to show PML-resource and PML-input/output object in the model. In such case, EPC could be sufficiently complete.

- **Missing the concept of PML-state**

Based on PML-BWWP mapping rule 4, both PML-*event* and PML-*state* map to a single BWWP concept of BWWP-*state*. EPC already includes the PML-event concept, which indicates the "state" of the agent before and after a transformation. Thus, it avoids the problem of "Construct redundancy".

BWWP concepts	*PML constructs*	*EPC constructs*
Thing		
Actor	Agent	Organizational unit
Non-actor	Resource	N/A
	Input/output object	N/A
State		
State	Event	Event
	State	N/A
Stable state	Agent's final event	Agent's final event
Unstable state	Agent's all ev. except final	Agent's all events except final
Repr. of state	Data	Information in/out
Event		
Event	A seq. of "ev. – op. – ev."	A sequence of "ev. – func. – ev."
External event	A seq. triggering the agent	A sequence triggering the agent
Internal event	All other sequences	All other sequences
Transformation		
A seq. of transf.	Activity	**Function**
Transformation	Operation	**Function**
Law		
Law	Business rule	N/A
State-law	Op. –connector – ev.	Function –connector – event
Transformation law	Ev. –connector – op.	Event –connector – function

Figure 5-1: Mapping EPC constructs to BWWP concepts

- **Missing the concept of PML-business rule**

Without PML-business rule construct, EPC is unable to express the rules WITHIN activities or operations. However, it has a PML-logical connector construct, which enables EPC users to express the rules BETWEEN activities or operations. Lacking PML-business rule construct will not cause a big problem because whenever an activity or operation is decomposed, the business rules within it could be eventually expressed by logical connectors

between decomposed activities or operations.

- **Construct overload: Mixing PML-activity and PML-operation**

EPC does not differentiate PML-activity and PML-operation. Both concepts are expressed by the construct of "function". Mixing the two concepts causes the problem of "*Construct overload*". Users do not understand which function of an agent is to be triggered by its environment, which function is an intermediate operation that the agent will perform spontaneously without help from its environment, and which function is the last operation before the agent stops.

Generally speaking, EPC is ontologically complete but has a problem of "Construct overload".

5.2 Evaluation of IDEF0

Figure 5-2 shows the mapping from IDEF0 to BWWP constructs. For each BWWP construct, IDEF0 has at least one concept corresponding to it, so IDEF0 is ontologically complete. However, the language is not ontologically clear enough.

- **Construct overload: Mixing PML-agent and PML-resource**

IDEF0 does not explicitly include the concepts of PML-agent and PML-resource. Both of them are hidden in the construct of *Mechanisms*. Even though the *Mechanisms* construct maps to *BWWP-thing* (which includes both actor and non-actor), there is still the deficiency in that a single concept maps to two different BWWP construct (actor and non-actor). As the method does not differentiate PML-agent and PML-resource, users are hard to clearly identify the process participant who executes an activity or operation and the participant who is only utilized in an activity or operation. This causes the problem of "*Construct overload*" and the modeling language is ontologically unclear even though it is complete.

BWWP concepts	*PML constructs*	*IDEF0 constructs*
Thing		
Actor	Agent	**Mechanisms**
Non-actor	Resource	**Mechanisms**
	Input/output object	**Input, output**
State		
State	Event	**(1) Control (2) Output**
	State	N/A
Stable state	Agent's final event	Agent's final output
Unstable state	Agent's all event except final	Agent's all cont./output except final
Repr. of state	Data	**Input, output**
Event		
Event	A seq. of "ev. – op. – ev."	A seq. of "control – function – output"
External event	A seq. triggering the agent	A sequence triggering the agent
Internal event	All other sequences	All other sequences
Transformation		
A seq. of transf.	Activity	**Function**
Transformation	Operation	**Function**
Law		
Law	Business rule	**Control**
State-law	Op. –connector – ev.	Function –implicit connector – output
Transformation law	Ev. –connector – op.	Output–implicit connector – function

Figure 5-2: Mapping IDEF0 constructs to BWWP concepts

- **Construct overload: Mixing PML-output object, PML-event and PML-data**

The same situation occurs with the concept of *output* in IDEF0. IDEF0-output has three roles: The first one is an output object. The second one is generated by an activity/operation as a new PML-*event* to trigger another activity/operation, and the third one is a PML-*output data* generated by an activity/operation. Therefore, the overloaded construct IDEF0-output maps to three BWWP concepts: BWWP-*non-actor* (PML-output object), BWWP-*state of the actor* (PML-event) and BWWP-*representation of the state of anther thing* (PML-data).

- **Construct overload: Mixing PML-event and PML-business rule**

The third "construct overload" problem occurs with the concept of *Control*. The IDEF0-control construct represents two things: one is a PML-*event* that triggers a new activity/operation, and the other is a PML-*business rule* used in an activity/operation.

Ontologically speaking, IDEF0-Control maps to two BWWP concepts: BWWP-*state* (PML-event) and BWWP-*law* (PML-business rule). This problem potentially reduces the clarity of the model again.

- **Construct overload: Mixing PML-activity and PML-operation**

IDEF0 does not differentiate PML-activity and PML-operation. Like in EPC, both concepts are expressed by the construct of "function".

- **Construct redundancy: mapping two constructs (IDEF0-control & IDEF0-output) to BWWP-state**

When used to trigger activities, PML-*event* concept is sometimes seen in the IDEF0-*control* construct, and sometimes seen in the IDEF0-*output* construct. Both the IDEF0 constructs map to BWWP concept of *state*, causing a problem of "Construct redundancy".

- **Construct redundancy: mapping three constructs (IDEF0-mechanisms, IDEF0-input/output) to BWWP-non-actor**

The concept of BWWP-non-actor is seen in IDEF0 constructs of *Mechanisms*, *Input* and *Output*.

Overall speaking, IDEF0 method is ontologically complete, but it has the problems of "construct overload" and "construct redundancy".

5.3 Evaluation of IDEF3

IDEF3 method introduced in Chapter two includes three diagrams: Process Schematics (Figure 2-4), Transition Schematics (Figure 2-5) and Enhanced Transition Schematics (Figure 2-6). Considering all three diagrams entirely, the mapping to BWWP concepts could be seen in Figure 5-3.

- **Ontological incompleteness: Missing PML-agent in the Process Schematics**

In the Process Schematics, both PML-agent and PML-resource concepts are missing. The model describes a sequence of behaviors but does not mention who just participates in the behaviors and who is undergoing state-change. This causes the problem of "Ontological incompleteness".

At the same time, the missing concepts are included the Transition Schematics and the Enhanced Transition Schematics. In these two diagrams, an agent and a resource are regarded as an object whose PML-*states* are captured in this object-centered view of IDEF3. In this aspect, the Transition Schematics are the compensation to the Process Schematics.

BWWP concepts	*PML constructs*	*IDEF3 constructs*
Thing		
Actor	Agent	**N/A in (1); Object in (2)(3)**
Non-actor	Resource	**N/A in (1); Object in (2)(3)**
	Input/output object	**N/A**
State		
State	Event	**N/A in (1)(2)(3)**
	State	State in (2)(3)
Stable state	Agent's final event	Object's final state in (2)(3)
Unstable state	Agent's all event except final	Object's other states in (2)(3)
Repr. of state	Data	N/A in (1)(2)(3)
Event		
Event	A seq. of "ev. – op. – ev."	State-UOB-state (2); N/A (1)(3)
External event	A seq. triggering the agent	State-UOB-state (2); N/A (1)(3)
Internal event	All other sequences	State-UOB-state (2); N/A (1)(3)
Transformation		
A seq. of transf.	Activity	**UOB**
Transformation	Operation	**UOB**
Law		
Law	Business rule	N/A
State-law	Op. –connector – ev.	State–connector–state in (2)
Transformation law	Ev. –connector – op.	UOB – connector – UOB in (1)

Note: (1) Process Schematics (2) Transition Schematics (3) Enhanced Transition Schematics

Figure 5-3: Mapping IDEF3 constructs to BWWP concepts

- **Construct overload: Mixing PML-agent with resource in Transition Schematics**

However, as the PML-agent and PML-resource are equally treated as an object without differentiating into BWWP-*actor* and BWWP-*non-actor*, the problem of "Construct overload" exists in Transition Schematics.

- **Ontological incompleteness: Missing PML-event in the Process Schematics**

As there is not a PML-*event* concept in Process Schematics, this process-centered view of an agent's behaviors is not able to express the accurate events (or the agent's states) which trigger units of behavior (UOB). It only shows the order of UOBs, but never indicates the agent's states when the UOBs start and end. This is a problem of "Ontological incompleteness".

There is a PML-*state* concept (which is ontologically equivalent to PML-event) in the Transition Schematics, which compensates on the weakness of the Process Schematics.

- **Construct Excess in the Enhanced Transition Schematics**

The Enhances Transition Schematics provide a static view of objects participating in a process. The model is similar to "Entity-Relationship" model, and includes BWW Ontological static concepts of *thing*, *composite thing*, *property*, *class*, and *kind*. As these concepts are not related with process models, the problem here could be regarded as "*Construct excess*". Sometimes the model provides two views of a process (dynamic and static) in a single diagram, and the excessive constructs give the diagram reader unrelated information about a process.

- **Construct overload: mixing PML-activity and PML-operation**

In IDEF3 Process Schematics, the concept of "Unit of behavior (UOB)" does not make distinction between a PML-activity and a PML-operation. As mentioned before, it causes the problem of "construct overload" which reduces the language's clearness to represent the real-world business process.

Overall speaking, IDEF3 is ontologically incomplete. In terms of ontological clearness, it has the problems of "construct excess" and "construct overload" in some diagrams.

5.4 Evaluation of UML Activity Diagram

Figure 5-4 clearly shows that most of BWWP concepts do not have counterparts in UML Activity diagram, which indicates the problem of "Ontological incompleteness".

BWWP concepts	*PML constructs*	*Activity Diagram constructs*
Thing		
Actor	Agent	Separated by swim lanes
Non-actor	Resource	N/A
	Input/output object	N/A
State		
State	Event	N/A
	State	N/A
Stable state	Agent's final event	N/A
Unstable state	Agent's all ev. except final	N/A
Repr. of state	Data	N/A
Event		
Event	A seq. of "ev. – op. – ev."	N/A
External event	A seq. triggering the agent	N/A
Internal event	All other sequences	N/A
Transformation		
A seq. of transf.	Activity	**Activity**
Transformation	Operation	**Activity**
Law		
Law	Business rule	N/A
State-law	Op. –connector – ev.	N/A
Transformation law	Ev. –connector – op.	Activity – connector – activity

Figure 5-4: Mapping Activity Diagram constructs to BWWP concepts

An Activity Diagram does not include BWWP concepts of *non-actor*, *state*, *stable state*, *unstable state*, *event*, *external event*, *internal event*, and *state-law*. Without these concepts, the modeling language is week in expressiveness and the constructed model is incomplete in representing the real process. For example, users might not know the states of the agent before and after an activity/operation, which external event triggers the activity/operation,

and which states are lawful after an activity/operation.

Another problem is that Activity Diagram mixes the concepts of PML-activity and PML-operation. As mentioned before, it causes the problem of "Construct overload".

Generally, Activity Diagram is neither ontologically complete nor clear.

5.5 Evaluation of EDPM Method

Figure 5-5 shows that EDPM Method is ontologically complete as every BWWP concept has its counterpart in this modeling language.

BWWP concepts	*PML constructs*	*EDPM Method constructs*
Thing		
Actor	Agent	Agent
Non-actor	Resource	**Resource**
	Input/output object	**Input, output**
State		
State	Event	Event
	State	N/A
Stable state	Agent's final event	Agent's final event
Unstable state	Agent's all ev. except final	All event's except final
Repr. of state	Data	**Input, output**
Event		
Event	A seq. of "ev. – op. – ev."	A seq. of "event – activity – event"
External event	A seq. triggering the agent	A sequence triggering the agent
Internal event	All other sequences	All other sequences
Transformation		
A seq. of transf.	Activity	Activity
Transformation	Operation	Operation
Law		
Law	Business rule	Business rules
State-law	Op. –connector – ev.	Activity –connector – event
Transformation law	Ev. –connector – op.	Event – connector – activity

Figure 5-5: Mapping EDPM Method constructs to BWWP concepts

The following issues need to be addressed here:

- **Construct overload: Mixing PML-input/output object and PML-input/output data concepts**

As EDPM-input/output concept represents both PML-input/output object and PML-input/output data, it has two counterparts in BWWP: non-actor and representation of a thing's state. This causes the problem of "Construct overload".

- **Construct redundancy: mapping two constructs (EDPM-resource and EDPM-input/output) to BWWP-non-actor**

Both EDPM-resource (which is a PML-resource) and EDPM-input/output (which represents PML-input/output object) concepts are mapped to one BWWP concept: non-actor.

- **PML-activity and PML-operation**

EDPM Method is the only language that differentiates these two concepts among the five PMLs. This differentiation enables the model users to decompose an activity when it is necessary to capture the intermediate states of an agent. It also gives the users an idea that an activity involves a sequence of operations, and it is atomic and will eventually reach a stable state once it starts. This helps the users to understand the agent's behaviors more accurately.

- **PML-business rule and PML-logical connector**

Both of these concepts capture the rules of a process from different perspectives. A logical connector indicates the rules between activities/operations, and business rule shows the rules within an activity/operation. Therefore, including both concepts does not cause the any ontological problems.

Generally speaking, EDPM Method is ontologically complete, but it has the problem of "Construct overload" and "Construct redundancy".

Chapter 6

OBPM MODELING ALGORITHM

The evaluation of the five PMLs shows that they have some shortcomings. First, they are information-systems oriented rather than real-world oriented. Second, most of them are either ontologically incomplete or unclear. Finally, they do not have formally defined rules to follow when constructing a business process model.

In this chapter, a theory-based modeling algorithm is developed to provide general procedures for building an ontology-based process model (OBPM). The basic concepts and modeling rules are derived from BWW Ontology, which enables avoiding many problems of IS-oriented PMLs. The theoretical foundation is the set of ontology-based process model integrity rules found in Chapter four. The purpose of the algorithm is to provide a set of formal procedures for constructing theory-based process models.

As well, if the algorithm is used in the context of a specific PML, it can indicate the information that cannot be captured by this PML. Moreover, it can uncover the missing information in specific PML models.

The chapter will present the algorithm and demonstrate it with a business process example to show how the algorithm works.

6.1 Ontology-based Process Model (OBPM) algorithm

This section introduces the proposed conceptual modeling algorithm which is derived from BWW ontology theory for constructing an OBPM model. The purpose of this algorithm is to identify the triggering event(s) for a process, find all participants (agents and resources), recursively identify the activities and operations for each participant, and find out their sequences. For a particular process, the inputs of the algorithm are the process domain

and environment situation. After the algorithm is executed, the users could obtain the following outputs:

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

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

(3) The events that occur in the process, and their sequence.

(4) For each agent, there are:

- A list of activities the agent executes, and their sequences: *activity_list*
- For each activity in *activity_list*, a list of operations involved, and their sequences: *operation_list*

```
Main(environment, process domain)
Identify all triggering events from environment.
FOR EACH triggering event e, DO:
   Identify all things affected by e in process domain.
   IF empty THEN EXIT
   ELSE
      FOR EACH affected thing t, DO:
         Invoke Affected_Thing(t, e).
EXIT
```

Figure 6-1: The main algorithm

```
Affected_Thing(Thing t, Event e)
IF t changes states from stable to stable
   THEN
      Add t to resource_list;
   ELSE
      Add t to agent_list if not already there
      Decompose(t, e)
EXIT
```

Figure 6-2: Subroutine Affected_Thing(t, e)

```
Decompose(Agent a, Event e)
Identify sequence of events (e1, e2, e3, en) until a reaches a stable state. Index e as e0.
Add transformation of <e0, en> as an activity to a's activity_list if it is not there.
FOR EACH event ei in sequence (i=1 to n), DO:
   Add transformation of <ei-1, ei> to a's operation_list of this activity.
   Identify all other things affected by ei.
   IF empty THEN CONTINUE to next event.
      ELSE
         FOR EACH affected thing aff_t, DO:
            Invoke Affected_Thing(aff_t, ei).
EXIT
```

Figure 6-3: Subroutine Decompose(a, e)

6.2 Algorithm description

The proposed algorithm for identifying these lists consists of the following three parts: The main algorithm *Main(environment, process domain)*, subroutine *Affected_Thing(thing, event)* and subroutine *Decompose(agent, event)*. It provides a recursive mechanism to investigate the state-changes of each agent, and generates a process model when it is completed. The main algorithm identifies the triggering events from the environment that occur in order, and identifies all the things affected by the events. For each affected thing, it calls the *Affected_Thing()* subroutine. The *Affected_Thing()* subroutine first looks at the state-change of the affected thing, and identifies the agents and resources. For an agent, the subroutine *Decompose()* is invoked to find out each agent's operations and activities, as well as the events generated during the state-changes.

6.2.1 The main algorithm

Given the process domain knowledge and the environment situation, the algorithm shows how to identify the affected things and how to handle them in later steps.

- Input 1: Environment situation

Environment situation provides the information of the triggering event(s) that is needed to start the process. For example, when *customer payment is made*, OR *assembled products*

are ready, the "*order processing*" process is triggered.

- Input 2: Process domain knowledge

The process domain knowledge gives the information such as which things are affected by the triggering event(s). Here, the word "affected" means that the thing's stable state is changed. It could either change to an unstable state and keep changing till stable, or move to another stable state directly.

The domain expert may be asked the following questions:

What are the participants in the process?

When the triggering event(s) occurs, which participants will start to change states?

If no things are affected by the triggering event(s) e, the algorithm will exit, which means that the environment could not affect the process. Otherwise, for each affected thing t the algorithm will invoke a subroutine Affected_Thing(t, e) to find what happens when thing t is affected by event e.

6.2.2 Subroutine: Affected_Thing(input: Thing t, Event e)

This procedure gives the instructions on how to differentiate resource and agent when thing *t* is affected by event *e*, and how to deal with the state-change of an agent. The two inputs are: Thing t and Event e.

The subroutine starts by looking at the response of affected thing *t* to the event *e*:

We ask the domain expert:

Does this affected thing change its states from stable to stable, or from stable to unstable?

(1) Affected thing *t* is a resource

If its state-change is from stable to stable, that means *t*'s state-change is not able to affect other things in the domain. Therefore, we could consider it as a resource used by the agent *t* during one of its operations or activities. In this way, the resource *t* is added to the process'

resource_list.

(2) Affected thing *t* is an agent

Otherwise, if the state-change of *t* is from a stable to an unstable state, the thing will keep changing and might change the state of other things. This thing should be added to the process's *agent_list* if it has not been added to the list before.

(3) Decompose the change into activities and operations

When an agent's state-change is triggered, a procedure to decompose the agent's state-change into activities and operations is needed. The subroutine *Decompose(Agent t, Event e)* is invoked for this purpose.

6.2.3 Subroutine: Decompose(input: Agent a, Event e)

This subroutine provides instructions on how to decompose the state-change of an agent *a* into activities and operations when it is triggered by event *e*. The inputs are: Agent a and event e.

(1) Identify the sequence of events until agent *a* reaches stable state.

The first question we need to ask is:

*When event **e** occurs and agent **a** undergoes a state-change, what other events in sequence could occur to the agent **a**?*

In order to easily label the activities and operations, each event in the sequence is indexed from *1* to *n* (*n* is the number of events in the sequence), and the triggering event *e* is indexed as e0.

(2) Add an activity to the agent's activity_list

Each pair of any consecutive events represents an operation, and an activity comprises the whole sequence starting from e_0 to e_n. This activity should be added to the agent *a*'s *activity_list*. The last event e_n in the sequence is a stable state of agent *a*, and it might be the

goal of this agent in the process if the agent has no future change. When this goal is achieved, the agent will stay in stable state till another agent triggers it to change state for a new goal.

(3) Add all operations in the current activity to the agent's operation_list of this activity.

The second question to ask the domain expert is:

*What are the operations that agent **a** performs to finish the current activity?*

As each event-pair of any consecutive events in the sequence of $\{e_0, e_1, e_2, \ldots, e_n\}$ is an operation, we need to add them to the agent's *operation_list*.

(4) For each event in sequence, find the affected things.

The third question to ask the domain expert is:

For each event in the sequence, which things are affected by it?

If there are not any things affected by the event, it means that the agent is changing continuously without affecting other things. Therefore, we will not include the agent itself from the affected things list. The algorithm will continue and check the next event in the sequence till the end.

(5) For each affected thing t by each event e, invoke Affected_Thing(t, e).

For each event e_i in the sequence, if it affects other things, the algorithm will recursively call the subroutine *Affected_Thing* (t, e_i) to each affected thing t, until the state-change of all things t affected by e_i are decomposed into activities and operations. As previously described, this subroutine will recursively do the following: Find out whether the affected thing t is a resource or an agent. If it is a resource, add it to the *resouce_list* of the activated operation of t. And if it is an agent, the subroutine *Decompose*(t, e_i) will be invoked recursively.

6.3 Theoretical foundation of the algorithm

The algorithm is generated based on the ontological-based integrity rules, which are

summarized in Chapter four. Table 6-1 shows the relationships between the steps in the OBPM algorithm and the ontology-based integrity rules.

Algorithm	Step	Rule number
Main()	Identify all triggering events from environment.	(1), (3), (9)
	Identify all things affected by each triggering event.	(2), (3), (10)
Affected_Thing(t, e)	If t changes state from stable to stable, then add it to resource_list.	(2), (6), (10), (13)
	Otherwise, add t to agent_list.	(2), (10), (12), (13)
Decompose(a, e)	Identify sequence of event until a reaches stable.	(5), (8), (12), (13)
	Add the whole sequence as an activity to activity_list.	(12), (13)
	Add each pair of consecutive events as an operation to operation_list.	(12), (13)

Note: The ontology-based integrity rules are listed as follows for easy reference.

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

(2) All things interact by changing the value of mutual properties.

(3) The first change in the system is triggered by the system environment. This change is effected by a thing in the environment to change the mutual property of a thing in the system.

(4) When an actor modifies its intrinsic properties, it does not affect any other things but only changes the actor itself.

(5) After a thing starts change, it modifies all its properties and eventually reaches stable.

(6) All properties an non-actor are mutual with actors, and its intrinsic properties are of no interest in the model. Its state-changes are always caused by actors, and they are always from a stable state to another stable state.

(7) A composite thing and its components are observed from different standpoints.

(8) The process is completed when the last thing stops changing and all things are in stable state.

(9) In order to change, each thing must be activated by an external event.

(10) The before-state of an external event is always stable. An actor has its external event end with an unstable state. A non-actor has it external event end with a stable state.

(11) All the subsequent events following an external event are internal events for an actor. There is at least one internal event during an actor's change. A non-actor does not have any internal events.

(12) After an actor starts to change, whenever it modifies one property, it undergoes one transformation. When an actor modifies all its properties, it undergoes one sequence of transformations.

(13) An actor undergoes a transformation when its state is changed from one to another over time. A non-actor does not undergo any transformations.

Table 6-1: OBPM algorithm and the ontology-based integrity rules

6.4 WaterSystem Example

For illustration purpose, this section will use the "Order processing" process in the *WaterSystem* example introduced in Chapter two to present how to use the algorithm to generate an OBPM model.

Figure 6-4 shows the EPC model of the process. There are two external events

generated from the process' environment. One is e01 (e01 = Payment made by customer) generated by the "Order taking" process, and the other is e02 (e01 = Assembled products ready) generated by the "Assembly" process. Taking their sequence of occurrence into consideration, and also considering the value of event e2 which is the *state of warehouse dept. after checking the order in warehouse*, we need to analyze the final model for the following three cases:

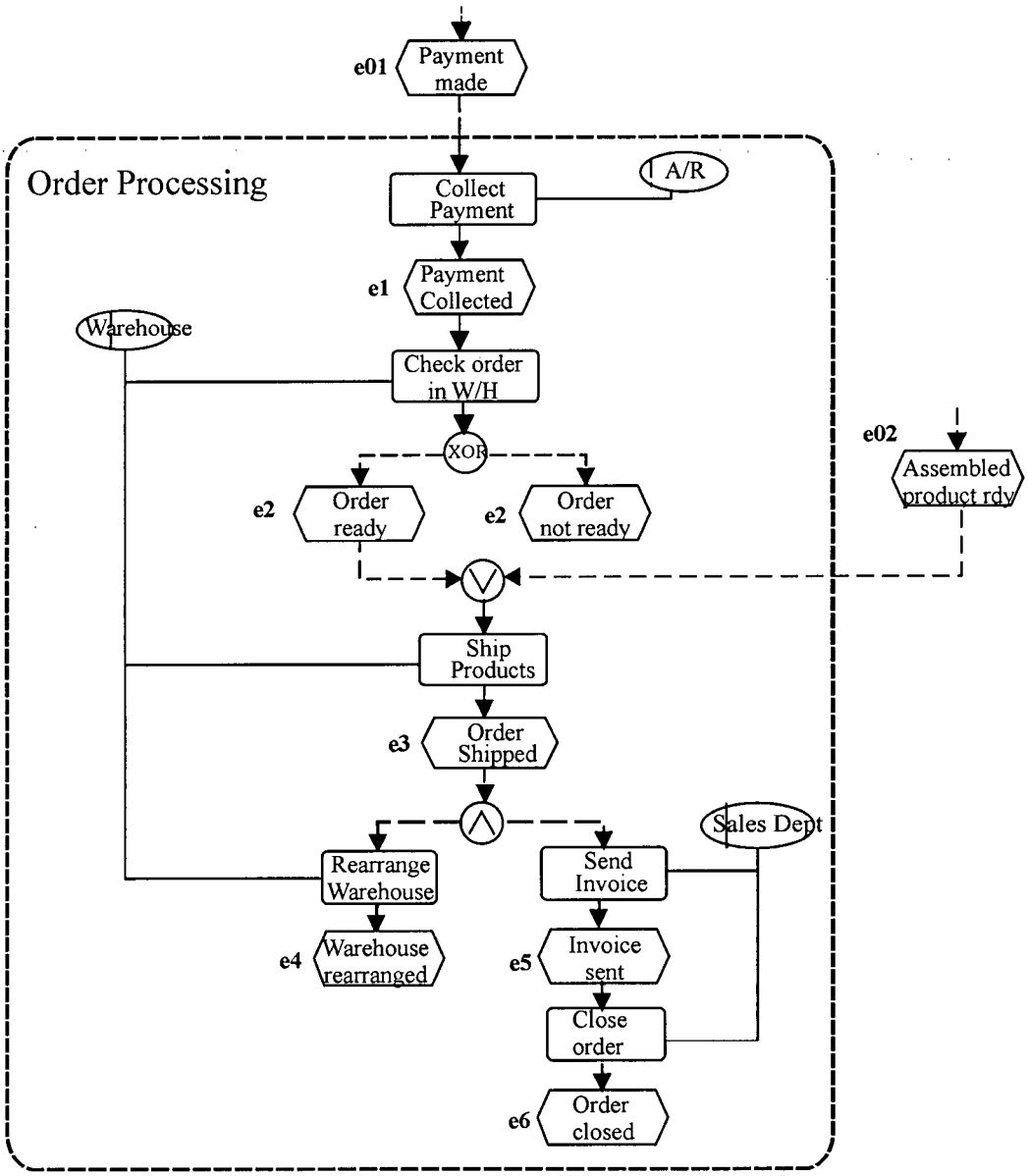


Figure 6-4: EPC model of "Order processing" process

(1) Case A: e01 occurs first, e2 = order not ready, and then e02 occurs.

(2) Case B: e01 occurs first, e2 = order ready, and then e02 occurs.

(3) Case C: e02 occurs before e01. [4]

6.4.1 Running the algorithm on case A

Figure 6-5 presents the steps of running the algorithm on Case A.

Step 1 and 2 identifies the triggering events: e01 and e02.

e01 = Payment made

e02 = Assembled products ready

Steps 3 to 6 handle the state-changes of the affected things of e01: A/R dept., Bank machine and Bank teller. Steps 7 to 10 handle the state-changes of the affected things of e02: Warehouse dept., Forklift and Elevator. Among these steps, 5, 6, 9 and 10 simply add the resources into resource_list because the affected things only changes states from stable to stable. Steps 4 and 8 deal with the situations that the affected things change states from stable to unstable. The most important steps are explained below.

6.4.1.1 Analyzing step 4: Affected_Thing(A/R, e01)

This step deals with the procedures needed for A/R dept. when it is affected by e01 (e01 = Payment made).

First (in step 41) it is noticed that the A/R dept. changes states from stable to unstable, which means it is an agent rather than a resource. Thus, the A/R dept. is added to the agent_list, which is empty at the moment.

Then in step 42 the algorithm attempts to decompose the state-change of A/R dept. into activities and operations through invoking *Decompose(A/R., e01)*.

6.4.1.2 Analyzing step 42: Decompose(A/R, e01)

First, identify a sequence of events until the A/R dept. reaches a stable state. Only one event is found:

[4] Note: The three cases in this example indicate three different workflows of the same process.

```
1. e01 = Payment made
2. e02 = Assembled products ready
3. Affected things of e01: {A/R, Bank machine, Bank teller}
4. Affected_Thing(A/R, e01)
        41. Agent_list: {A/R}
        42. Decompose(A/R, e01)
                421. e1 = Payment collected
                422. A/R activity_list: {<e01, e1>}
                423. A/R first activity'soperation_list: {<e01, e1>}
                424. Affected things of e1: {W/H}
                425. Affected_Thing(W/H, e1)
                        4251. Agent_list: {A/R, W/H}
                        4252. Decompose(W/H, e1)
                                42521. e2 = Order not ready
                                42522. W/H activity_list: {<e1, e2>}
                                42523. W/H first actvity'soperation_list : {<e1, e2>}
                                42524. Affected things of e2: {}
                                Return
                        Return
                Return
        Return
5. Affected_Thing(Bank machine, e01)
        41. Resource_list: {Bank machine}
        Return
6. Affected_Thing(Bank teller, e01)
        51. Resource_list: {Bank machine, Bank teller}
        Return
7. Affected things of e02: {W/H, Forklift, Elevator}
8. Affected_Thing(W/H, e02)
        81. Agent_list: {A/R, W/H}
        82. Decompose(W/H, e02)
                821. e3 = Order shipped
                822. e4 = Warehouse rearranged
                823. W/H activity_list: {<e1, e2>, <e02, e4>}
                824. W/H second activity's operation_list: {<e02, e3>}
                825. Affected things of e3: {Sales, Mail}
                826. Affected_Thing(Sales, e3)
                        8261. Agent_list: {A/R, W/H, Sales}
                        8262. Decompose(Sales, e3)
                                82621. e5 = Invoice sent
                                82622. e6 = Order closed
                                82623. Sales activity_list: {<e3, e6>}
                                82624. Sales first activity's operation_list: {<e3, e5>}
                                82625. Affected things of e5: {Sales, Computer}
                                82626. Affected_Thing(Computer, e5}
                                        826261. Resource_list: {Bk mchn, Bk tlr, Cmpt.}
                                        Return
                                82627. Sales first activity's operation_list: {<e3, e5>, <e5, e6>}
                                82628. Affected things of e6: {}
                                Return
                        Return
                827. Affected_Thing(Mail, e3)
                        8271. Resource_list: {Bank machine, Bank teller, Computer, Mail}
                        Return
                828. W/H second activity's operation_list: {<e02, e3>, <e3, e4>}
                829. Affected things of e4: {}
                Return
        Return
9. Affected_Thing(Forklift, e02)
        91. Resource_list: {Bank machine, Bank teller, Computer, Mail, Forklift}
        Return
10. Affected_Thing(Elevator, e02)
        101. Resource_list: {Bank machine, Bank teller, Computer, Mail, Forklift, Elevator}
        Return
```

Figure 6-5: Running the algorithm on case A

e1 = Payment collected

Next, an activity of <e01, e1> is added to A/R's activity_list. This is the first activity of the A/R dept. in the list.

A/R's activity_list: {<e01, e1>} (Namely, Collect payment)

Next, operations are identified for this first activity. Only one operation is found, and it is added to the operation_list of this activity:

A/R's first activity's operation_list: {<e01, e1>} (Namely, Collect payment)

Next, the affected things of event e1 are found: Warehouse dept. only. The algorithm will deal with the situation again that an agent is affected by an event. This is executed in step 425.

6.4.1.3 Analyzing step 425: Affected_Thing(Warehouse dept., e1)

When the Warehouse dept. is affected by e1 (e1 = Payment collected), the algorithm adds the Warehouse dept. into the agent_list as it changes states from stable to unstable. Then it decomposes the state-change of the Warehouse dept. into activities and operations.

Similar to step 42, it first identifies the sequence of events, which has only one event e2 (e2 = Order not ready). Then it adds the first activity <e1, e2> (Namely, Check order in Warehouse) into the Warehouse dept.'s activity_list, adds the only operation <e1, e2> (Namely, Check order in Warehouse) into the operation_list of this activity, and check if anything is affected by e2. Fortunately, nothing is affected and the subroutine returns.

6.4.1.4 Analyzing step 8: Affected_Thing(Warehouse dept., e02)

Similar to step 4, step 8 also deals with the situation that the Warehouse dept. is affected by event e02. It first identifies two new events:

e3 = Order shipped

e4 = Warehouse rearranged

The activity <e02, e4> (Namely, ship products and rearrange warehouse) is the second

activity of the Warehouse dept. It is identified in step 823 and added to the activity_list of the Warehouse dept. And, two operations are added to the second activity's operation_list. The first operation <e02, e3> (Namely, ship products) is added to the list in step 824, and the second operation <e3, e4> (Namely, rearrange warehouse) is added in step 828. Some resources (Computer and Mail) are added to the resource_list in step 82625 and 8271. Step 826 deals with the situation that the Sales dept. is affected by the event e3.

6.4.1.5 Analyzing step 826: Affected_Thing(Sales dept., e3)

As the Sales dept. is a new agent, it is added to the agent_list.

When decomposing the Sales dept.'s state-change, two new events are identified:

e5 = Invoice sent

e6 = Order closed

An activity is found for the Sales dept. in step 82623, and two operations are added to the operation_list of this activity in steps 82624 and 82627: one is <e3, e5> (Namely, Send invoice) and <e5, e6> (Namely, Close order).

6.4.1.6 Final output of case A

The final output of the algorithm is the OBPM model the user obtained. Case A has the following outputs:

Triggering event: e01 = Payment made by customer

e1 = Payment collected

e2 = Order not ready

e3 = Order shipped

e4 = Warehouse rearranged

e5 = Invoice sent

e6 = Order closed

Agent_list: {A/R department, Warehouse department, Sales department}

Resource_list: { Bank machine, Bank teller, Computer, Mail, Forklift, Elevator}

A/R dept. activity_list: {<e01, e1>}

A/R dept. operation_list of first activity: {<e01, e1>}

Warehouse dept. activity_list: {<e1, e2>, <e2, e4>}

Warehouse dept. operation_list of first activity: {<e1, e2>}

Warehouse dept. operation_list of second activity: {<e02, e3>, <e3, e4>}

Sales dept. activity_list: {<e3, e6>}

Sales dept. operation_list of first activity: {<e3, e5>, <e5, e6>}

The final model of case A is presented as Figure 6-6.

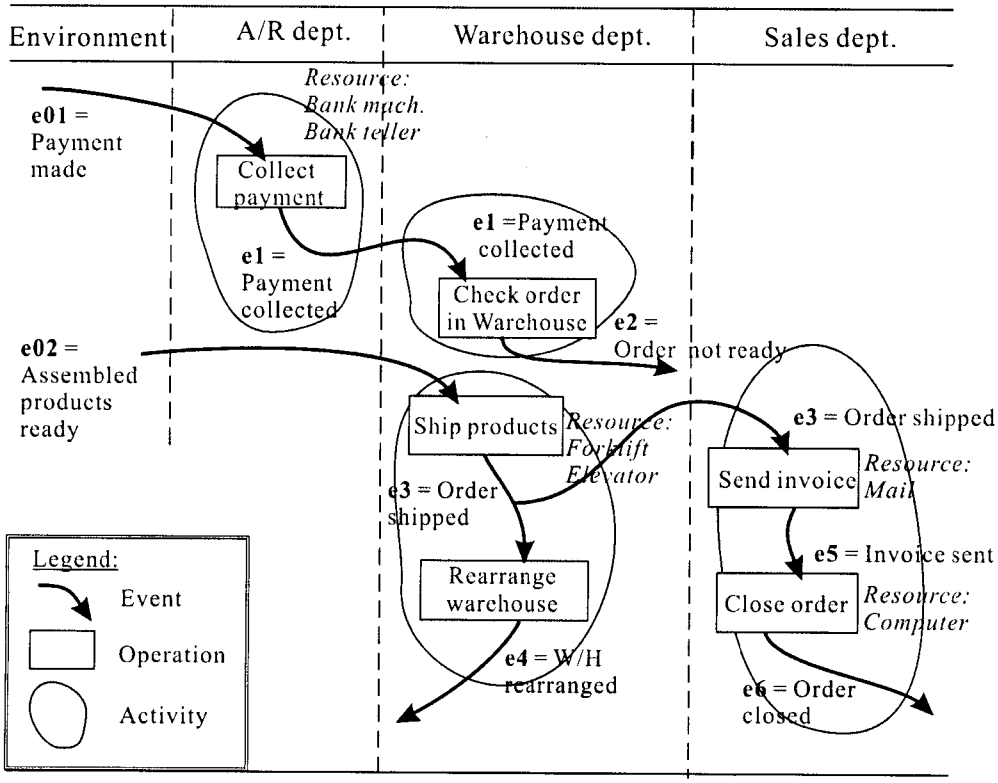


Figure 6-6: Final model of case A: triggered by e01 & e02, and e2 = order not ready

6.4.2 Running the algorithm on case B

In case B, e01 occurs first, e2 = order ready, and then e02 occurs. As e2 = order ready, the agent Warehouse dept. is at an unstable state and it will continue its state-change till it reaches a stable state. Thus, a sequence of events <e1, e4> is an activity and it has three operations: <e1, e2> (Namely, check order in warehouse), <e2, e3> (Namely, ship products) and <e3, e4> (Namely, rearrange warehouse). Therefore, when the Warehouse dept. is at

unstable states and keeps changing, the occurrence of e02 (Namely, Assembled products ready) will have no effect on it. The algorithm determines that the set of affected thing by e02 is empty, and returns.

If e02 occurs when the Warehouse dept. finishes the activity of "ship product and rearrange warehouse" and stays in a stable state, it will check the affected thing by e02 and proceed.

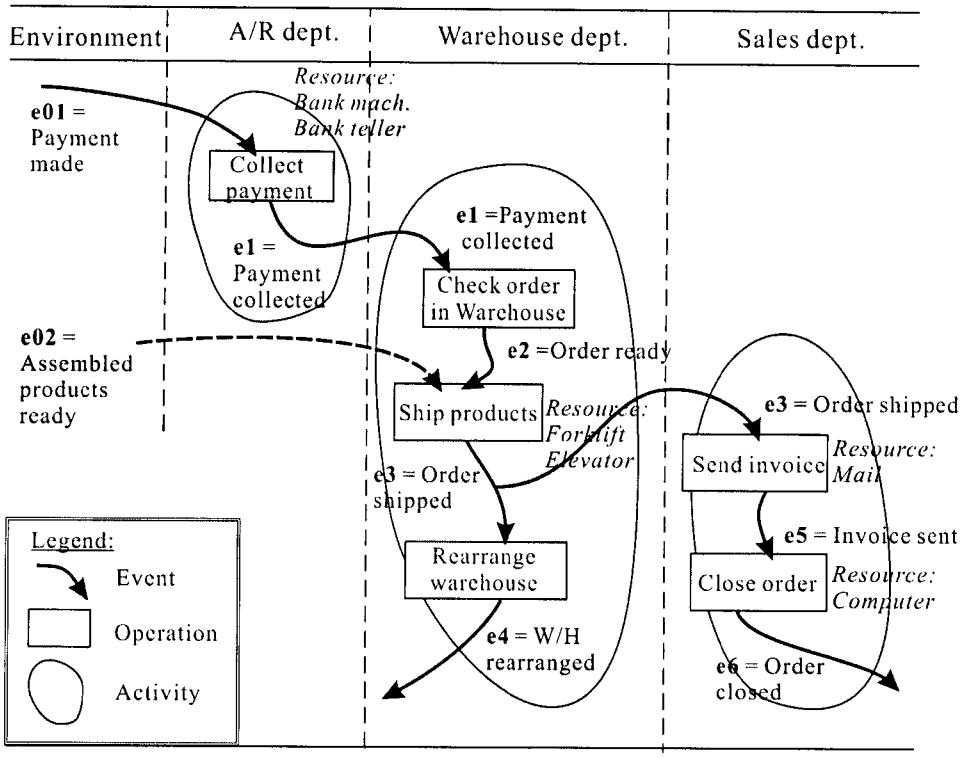


Figure 6-7: Final model of case B: triggered by e01, and e2 = order ready

The result of running the algorithm on case B is omitted. The final model of case B is presented in Figure 6-7.

6.4.3 Running the algorithm on case C

In case C, e02 occurs before e01, and e2 could be whichever values.

Before e01 (e01 = Payment made by customer) and e02 (e02 = Assembled products ready) occur, the Warehouse dept. is in a stable state. Event e02 triggers the Warehouse dept. to change until it reaches a stable state. The algorithm would add the first activity <e02, e4>

to the Warehouse dept.'s activity_list, and find its operations <e02, e3> (Namely, Ship products) and <e3, e4> (Namely, Rearrange warehouse). It also identifies the activities and operations for the Sales dept.

If e01 (e01 = Payment made by customer) occurs in the meantime and e1 (e1 = Payment collected) occurs when the Warehouse dept. does not finish its current activity, it will not affect the Warehouse dept. as it is in unstable states. If e1 occurs when the Warehouse dept. finishes its first activity and stays in a stable state, e1 will cause the agent to change again and a second activity is added, which could be <e1, e2> if e2 = order not ready, or <e1, e4> if e2 = order ready.

The final model of case C is displayed in Figure 6-8.

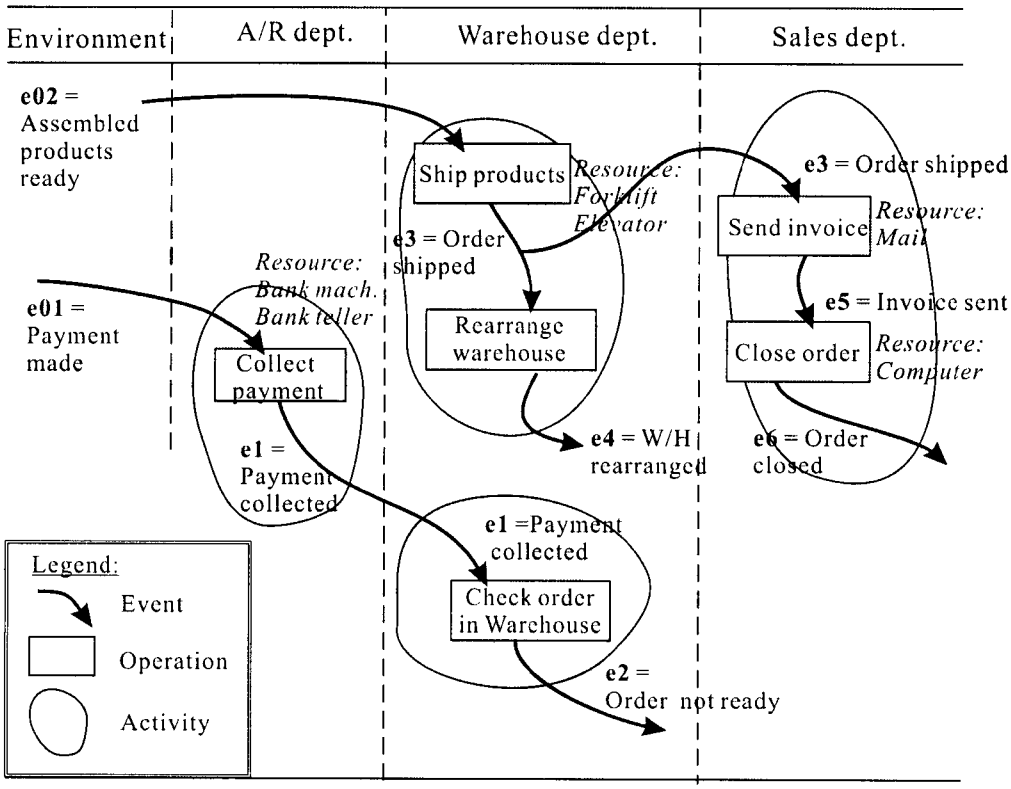


Figure 6-8: Final model of case C: e02 occurs before e01

6.5 Evaluation of the OBPM algorithm

The OBPM algorithm provides a set of necessary concepts and formal instructions on

how to build a process model. It is necessary to evaluate its usefulness in terms of its basic concepts and procedures.

6.5.1 Basic concepts

The OBPM algorithm generates the following information in the final model: *agent*, *resource*, *event*, *activity*, *operation*, as well as *business rule*. It is noted that the concept of *business rule* is not explicitly expressed in the algorithm; however, an analyst could generate different outputs according to different situations of the process domain. This implicitly expresses the business rules in the process. For example, in the previous discussion of case A, B and C, the business rules are implied in the order of the event's occurrence.

The basic concepts of OBPM algorithm are derived from BWWP ontological constructs. And this enables the algorithm to capture the real world knowledge of a business process as complete and clear as possible. On the other hand, as we discussed previously, most of the five PMLs have ontological problems in terms of constructs expressiveness. Therefore, it is necessary to use the algorithm with the PMLs together so that a theory-based process model could be obtained.

6.5.2 Basic procedures

The OBPM algorithm sets up its modeling procedures according to the ontology-based integrity rules. The rules represent the basic truth and the general law of cause and effect of the real world. For example, one of the rules mentioned: "*All things interact by changing the value of mutual properties.*" It is clear for a model builder that if two things do not have mutual property, they have no way to interact.

While in most PMLs, the procedures on how to build a process model are not well defined. They are based on traditional process-modeling heuristics, and do not have formal rules to follow. An analyst could only discover the domain knowledge from particular facts

and examples, and move to a solution by trial and error. It is convinced that using PMLs which are guided by the OBPM algorithm is necessary to build a reasonable model for a real-world business process.

Chapter 7

CONCLUSION

7.1 Conclusion

The thesis developed a theory-based process modeling methodology. The methodology covers all the aspects (such as functional, behavioral, organizational and informational aspects) of process modeling languages. It includes the constructs and rules that are derived from BWWP ontological concepts, and the OBPM algorithm that suggests a procedure for constructing a process model in a given situation. Additionally, it presents integrity rules that show the way to check that a process model is semantically correct.

The thesis starts by comparing five currently used process modeling languages including EPC, IDEF0, IDEF3, UML Activity Diagram, and EDPM Method. It finds the generic constructs needed for a PML by observing the difference between five PMLs as well as what they have in common. In order to gain a clear understanding of these modeling languages' expressiveness, each of them is used to model a whole set of five business processes of a manufacturing enterprise.

The next step is to examine the process related ontological concepts (BWWP concepts) to define an ontology-based process model.

This ontological model of a process is investigated by doing ontological analysis to a sample process. The process model includes four things, three of them are actors and one is a non-actor. The things interact with each other. The model also incorporates composite things. By looking at each thing's state curve, the relationships between PML constructs and BWWP concepts are clarified. A set of PML-BWWP construct mapping rules is finally generated, and leads to a set of ontology-based process-model integrity rules. These two

outcomes have the following benefits:

(1) The PML-BWWP mapping rules are used as a guideline to evaluate a general process modeling language. The thesis specifically evaluates the five PMLs based on these mapping rules, and their ontological deficiencies are found accordingly.

(2) Ontology-based integrity rules are used to set up an "ontology-based process-modeling" algorithm.

7.2 Contributions

A significant contribution of the thesis is the PML-BWWP concept mapping rules. The mapping rules could be used in future research to evaluate other process-modeling languages, and could also guide the applications of PMLs in the business practices.

In addition, the OBPM algorithm on how to establish an ontology-based process model can be used for automatically generating a process model. The strength of this conceptual modeling algorithm is that it is derived from the formal ontological foundation rather than from traditional process-modeling heuristics.

7.3 Limitations and future research

One limitation of the OBPM algorithm is that it does not include the "*Data*" concept. As "*Data*" represents the stable state of a thing in ontology and the represented thing behind data is not of interest to most PMLs, we have not addressed it. A deeper meaning of data could provide new insights in future research.

Even though the algorithm generates a *resource_list* in the model, it does not tell *who* utilized the resource in the list during *which* operation. The algorithm need to be refined to find the operation that the resource is used/modified, and the agent who uses/modifies the resource in future research. And, it also need to be refined to clearly present the logical connectors and business rules instead of implying them by a number of cases. Moreover, it

should be able to identify the input/output concrete objects in the future.

By adding a standard set of symbols and giving them meanings, OBPM algorithm could be further developed into a process-modeling language in the future.

Also, as the OBPM algorithm is new to the process-modeling field, and it was only tested in a limited number of examples. In future research, a CASE tool could be developed to automate the model generation procedures, so that the OBPM algorithm could be applied and tested in more complicated cases.

Finally, the empirical studies could be conducted to examine the usefulness and the ease-of-use of the OBPM algorithm as well.

BIBLIOGRAPHY

Ambler, Scott W. (2000). How to dram UML Activity Diagrams, IBM DeveloperWorks, September 2000

Ambrose, Paul J. and Ramaprasad, Arkalgud. (1997). Developing Sustained Competitive Advantage: Business Process Reengineering versus Management of Information as a Resource, Association for Information Systems, 1997 America Conference, Indianapolis, Indiana, August 15-17, 1997

Bider, Ilia. (2000). IbisSoft Inc. Stockholm Sweden, Business Process Modeling –Concepts, Workshop on Practical Business Process Modeling (PBPM*00), Stockholm, 5-6 June 2000

Bunge, M.A. (1977). Ontology I: The Furniture of the World, Vol. 3 of Treatise on Basic Philosophy. D. Reidel Publishing Company, Dordrecht, Holland.

Bunge, M.A. (1979). Ontology II: A World of Systems, Vol. 4 of Treatise on Basic Philosophy. D. Reidel Publishing Company, Dordrecht, Holland.

Davenport, T.H. & Short, J.E. (1990). "The New Indusgtrial Engineering: Information Technology and Business Process Redesign", Sloan Management Review, pp. 11-27.

Evermann, Jeorg and Wand, Yair. (2001a). Towards Ontologically Based Semantics for UML Constructs, Conceptual Modeling – ER 2001, 20th International Conference on Conceptual Modeling Proceedings, P354 – 367

Evermann, Jeorg and Wand, Yair. (2001b). An Ontological Examination of Object Interaction in Conceptual Modeling, Proceedings of the Eleventh Annual Workshop on Information Technologies & Systems, WITS '01, New Orleans, Louisiana, USA, P91-96

Gemino, A. (1999). Empirical Comparisons of Systems Analysis Modeling Techniques. Ph. D. Thesis, University of British Columbia, Canada.

Green, P. (1997). Use of Information Systems Analysis and Design (ISAD) Grammars in Combination in Upper CASE Tools – An Ontological Evaluation. In Proceedings of the 2nd CAiSE/IFIP8.1 International Workshop on the Evaluation of Modeling Methods in Systems Analysis and Design (EMMSAD'97). Barcelona, Spain, 1-12.

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

Grover, Varun and Kettinger, William J. (1995). Business Process Change: Reengineering, Concepts, Methods and Technologies, IDEA Group Publishing.

Hanrahan, Robert P. (1995). The IDEF Process Modeling Methodology, CrossTalk Journal of Defense Software Engineering, U.S.Air Force's Software Technology Support Center, June 1995

Huckvale, Tim and Ould, Martyn (1995). Process Modeling – Who, What and How: Role Activity Diagramming. IDEA Group Publishing.

Jablonski, Stefan and Bussler, Christoph (1996). Workflow Management: Modeling Concepts, Architecture and Implementation. International Thomson Publishing Company.

Jung, Darrell. (1997). Object-oriented Modeling: From Analysis to logical Design, Master's thesis, Faculty of Commerce and Business Administration, University of British Columbia, August 1997

Knowledge Based Systems Inc., (2000) IDEF3 Process Flow and Object State Description Capture Method Overview, 2000

Mayer, Richard J., et al., (1992). IDEF Family of Methods for Concurrent Engineering and Business Re-engineering Applications, Knowledge-Based Systems, Inc., 1992.

Mayer, Richard J., et al. (1995). Information Integration for Concurrent Engineering (IICE) IDEF3 Process Description Capture Method Report, September 1995.

Martin, J. & Odell, J. (1992). "Object Oriented Analysis & Design". PTR Prentice Hall 1992, ISBN 0-13-630245-9.

OMG Unified Modeling Language Specification Version 1.3. (1999).

Paulson, Dan and Wand, Yair. (1992). An Automated Approach to Information Systems Decomposition, IEEE Transactions on Software Engineering, Vol. 18, No. 3, March 1992.

Scheer, August-Wilhelm. (1989). Business Process Engineering: Reference Models for Industrial Enterprises, Springer-Verlag Berlin, Heidelberg 1989.

US Department of Commerce. (1993). Federal Information Processing Standard Publication 183 (FIPSPUB 183), Announcing the Standard for Integration Definition for Function Modeling (IDEF0), Dec 21, 1993.

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

Wand, Yair and Wang Richard Y. (1996): Anchoring data quality dimensions in ontological foundations. Communications of the ACM, vol. 39, no. 11, pp. 86-95.

Wand, Y. and Weber, R. (1989). An ontological evaluation of systems analysis and design methods. Information System Concepts: An In-depth Analysis, E.D. Falkenberg, P. Lindgreen, ediors, pp. 79-107, North-Holland.

Wand, Y. and Weber, R. (1990a). An ontological model of an information system, IEEE Transactions on Software Engineering, 16(11), 1281-1291.

Wand, Y. and Weber, R. (1990b). Mario Bunge's Ontology as a Formal Foundation for Information Systems Concepts. Studies on Mario Bunge's Treatise. Rodopi B.V., Amsterdam – Atlanta, GA, (p123-149)

Wand, Y. and Weber, R. (1993) On the ontological expressiveness of information systems analysis and design grammars, Journal of Information Systems, 3(4), 217-237.

Wand, Yair and Weber, R. (1995). On the deep structure of information systems, Info Systems Journal 1995. 5, 203 – 223.

Wand, Yair and Woo, Carson. (2000). Object-Oriented Modeling: From Enterprise Model to Logical Design, Proceedings WIT's 2000

Zhao, Hao. (1995). Object-oriented Enterprise Modeling, Master's thesis, Faculty of Commerce and Business Administration, University of British Columbia, July 1995

Appendix I

WATERSYSTEM EXAMPLE

1. Process Description

The five business processes are presented here. The concepts used to depict the processes in the tables are based on EPC constructs.

■ Process 1: Order taking

The process starts when a customer has placed an order. The Sales department will check if the ordered products are available. The department's clerk will use the information system in computer to check products availability. If the ordered products are not available in warehouse, the clerk will make an assembly application so that the ordered products could be assembled when the required components are available.

Starting event	Triggered operation	Ending event	Resource used	Agent
Order placed	Check product availability	Product available XOR product unavailable.	Computer	Sales dept.
Product unavailable	Apply for assembly	Assembly application made	-	
Product available OR Assembly application made	Approve order	Order approved	Computer	
Order approved	Arrange payment	Payment made	Telephone, fax	

Table I -1: Definition of "Order taking" process

If the products are available in warehouse, or the unavailable products' assembly application is made, the order will be approved by the department manager, and the customer file will be updated. Then, the department will call or fax the customer according to the purchase order information and arrange them to make payment.

■ Process 2: Order processing

The triggering events of this process are "*customer payment made*" generated from

"Order taking" process and "*assembled products ready*" coming from "Assembly" process. When the first triggering event occurs, the Account Receivable section of Accounting department will collect customer's payment into the company's bank account. The section clerk might use bank machines or ask bank teller for help.

Starting event	Triggered operation	Ending event	Resource used	Agent
Payment made	Collect payment	Payment collected	Bank machine, bank teller	A/R
Payment collected	Check order in warehouse	Order ready XOR order not ready	-	Warehouse
Order not ready	-	-	-	Warehouse
Order ready OR assembled product ready	Ship products	Order shipped	Forklift, elevator	Warehouse
Order shipped	Rearrange warehouse	Warehouse rearranged	Forklift	Warehouse
Warehouse rearranged	-	-	-	Warehouse
Order shipped	Send invoice	Invoice sent	Mail	Sales dept
Invoice sent	Close order	Order closed	Computer	Sales dept
Order closed	-	-	-	Sales dept.

Table I -2: Definition of "Order processing" process

After payment is collected, the Warehouse department will check if the order is ready for shipping. If not ready, she might wait for notification from the manufacturing department. If the order is ready in warehouse OR the assembled products are ready for shipment (which is the 2nd triggering event), the Warehouse department will ship the order with help of a forklift and an elevator. When order is shipped, two things need to be done: Sales department should send invoice to the customer by mail and close the order, and Warehouse department should rearrange the warehouse space for new products and components to enter in (both assembled products and purchased components are stored in the same warehouse).

- **Process 3: Assembly**

The process is triggered by one or both of the two events: (1) Assembly application was made AND customer payment is made in the process of "*Order taking*"; OR (2) Warehouse department found an assembly application in the waitlist when new components arrived and added to warehouse in the process of "*Arrival*".

Starting event	Triggered operation	Ending event	Resource used	Agent
Assembly application made OR Assembly application waiting	Check components availability	Enough stock XOR short stock	Computer, product specification	Manuf. dept.
Enough stock	Assemble product	Assembly done	Product spec., assembly machines	
Short stock	Put assembly application to waitlist	Assembly application on waitlist	-	
Short stock	Make purchase application	Purchase needed	-	

Table I -3: Definition of "Assembly" process

The Manufacturing department will then check the components availability based on the ordered products' specification and the information system. If the required components are out of stock, two operations need to be done: (1) Manufacturing department will put the assembly application to the waitlist; AND (2) Manufacturing department will make purchase application. The latter case will trigger the process of "*purchasing*" which will be described later.

When the components are all available, the Manufacturing department will start assembling the ordered products, and add them to the Warehouse when assembly is done. Till then, the assembled product is ready for shipment.

- **Process 4: Purchasing**

This process is triggered by the event of "*purchase needed*" sent from the process of "*Assembly*". The Purchasing department will prepare a purchase plan, and then will place order with vendors. When orders are placed, the Account Payable section of Accounting

department will make payment to the vendors and the process is completed.

Starting event	Triggered operation	Ending event	Resource used	Agent
Purchase needed	Make purchase plan	Plan made	-	Purchasing dept
Plan made	Place order with vendors	Order placed with vendors	Telephone, fax	
Order placed with vendors	Pay vendors	Paid vendors	Bank account	A/P
Paid vendors	-	-	-	A/P

Table I -4: Definition of "Purchasing" process

- **Process 5: Arrival**

The triggering event "components arrived" comes from the environment. When a truckload of purchased components arrives, the Warehouse department will add them to the warehouse with help of forklift and elevator. And then, the Warehouse department will check if there are any waiting assembly applications. If there is no waiting application, the process is completed. Otherwise, it will trigger the "*Assembly*" process to manufacture the ordered products with the new arrived components.

Starting event	Triggered operation	Ending event	Resource used	Agent
Components arrived	Add to warehouse	In warehouse	Forklift, elevator	Warehouse
In warehouse	Check waiting assembly applications	Assembly application waiting XOR no assembly application waiting	Computer	Warehouse
No assembly application waiting	-		-	

Table I -5: Definition of "Arrival" process

2. Model the processes with five PMLs

For simplicity, IDEF3 Enhanced Transition Schematic model is omitted.

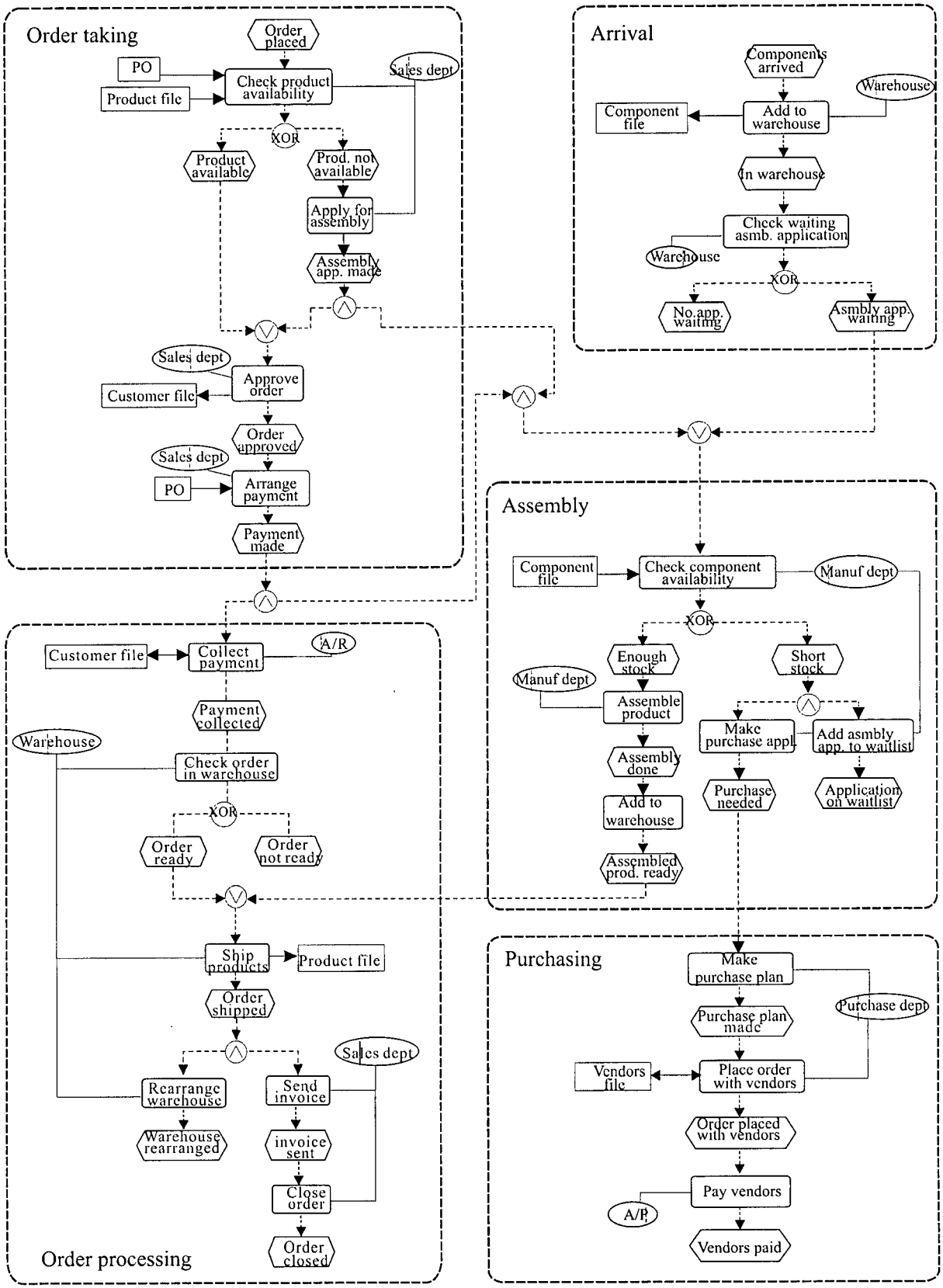


Figure I-1: EPC Model of WaterSytem Example

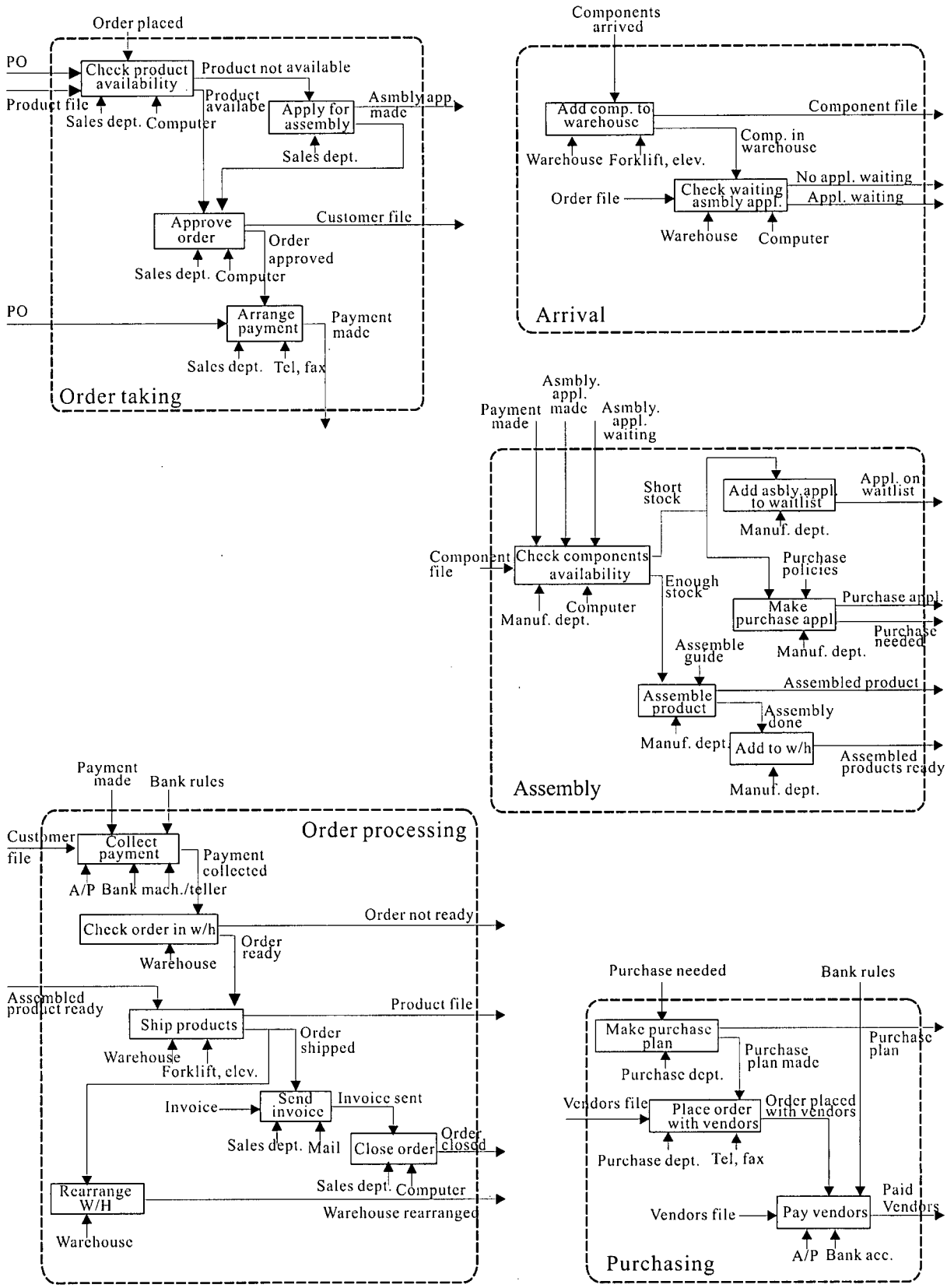


Figure I-2: IDEF0 Model of WaterSytem Example

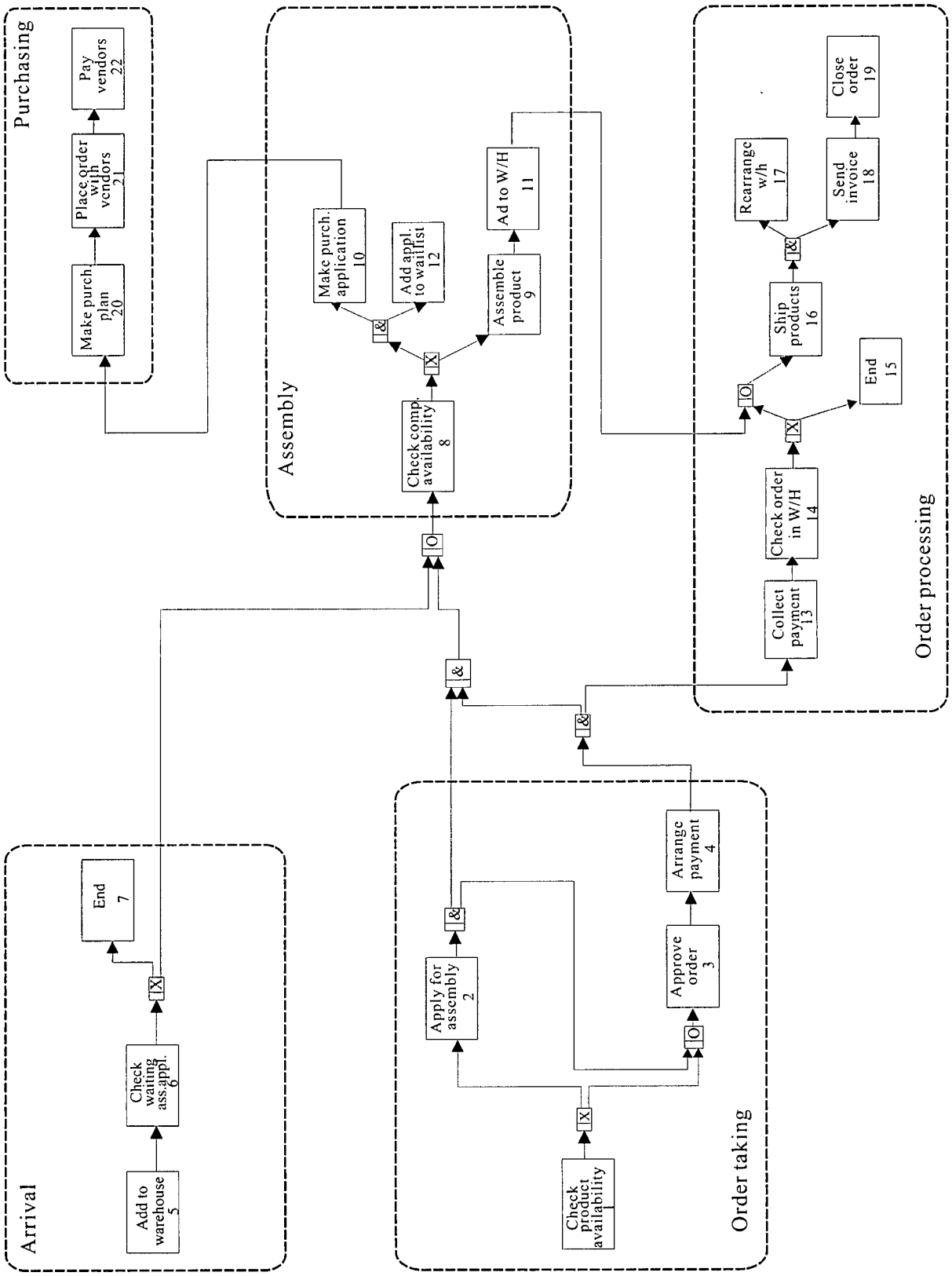


Figure I-3: IDEF3 Process Schematic Model of WaterSytem Example

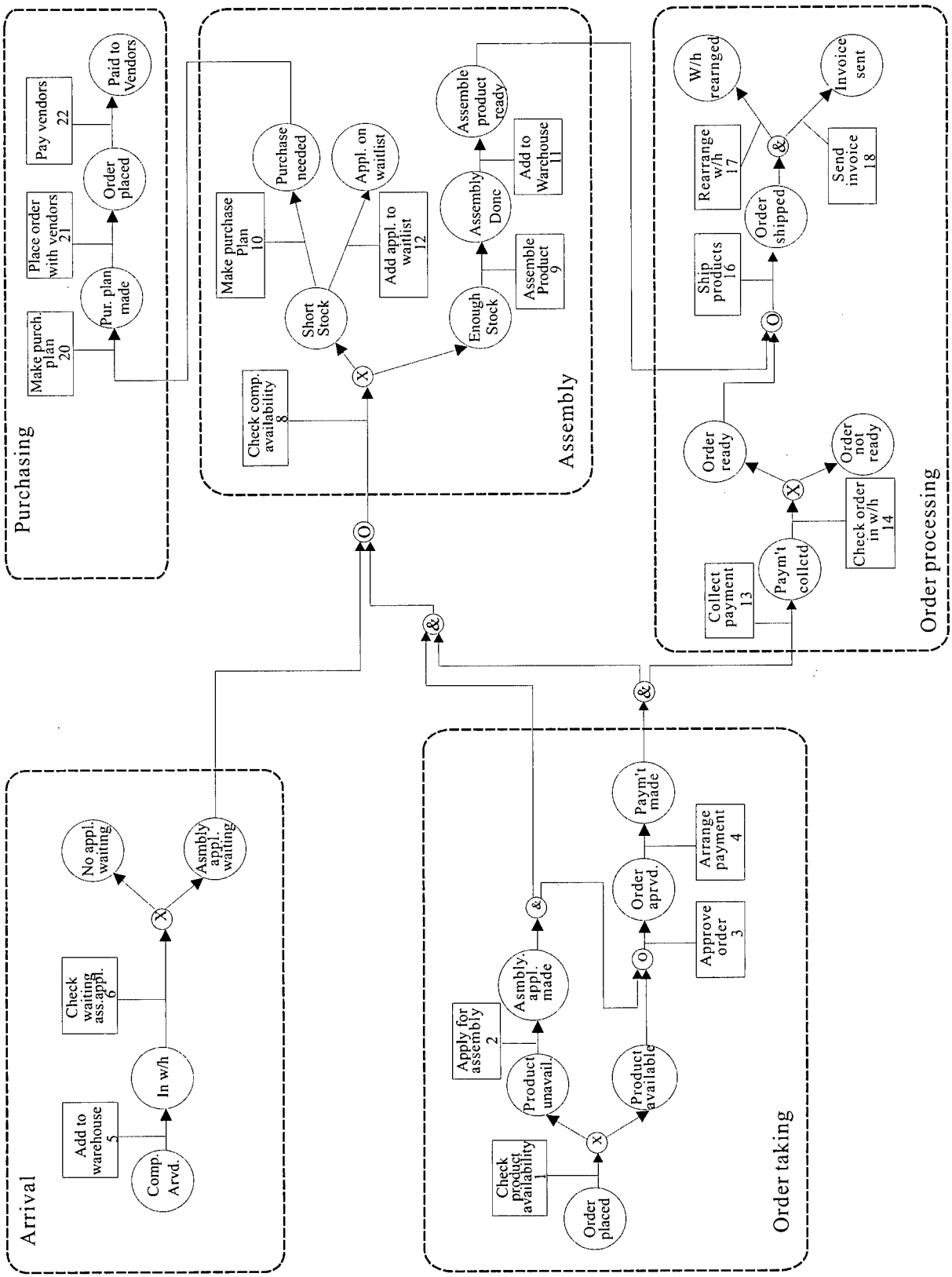


Figure I-4: IDEF3 Transition Schematic Model of WaterSytem Example

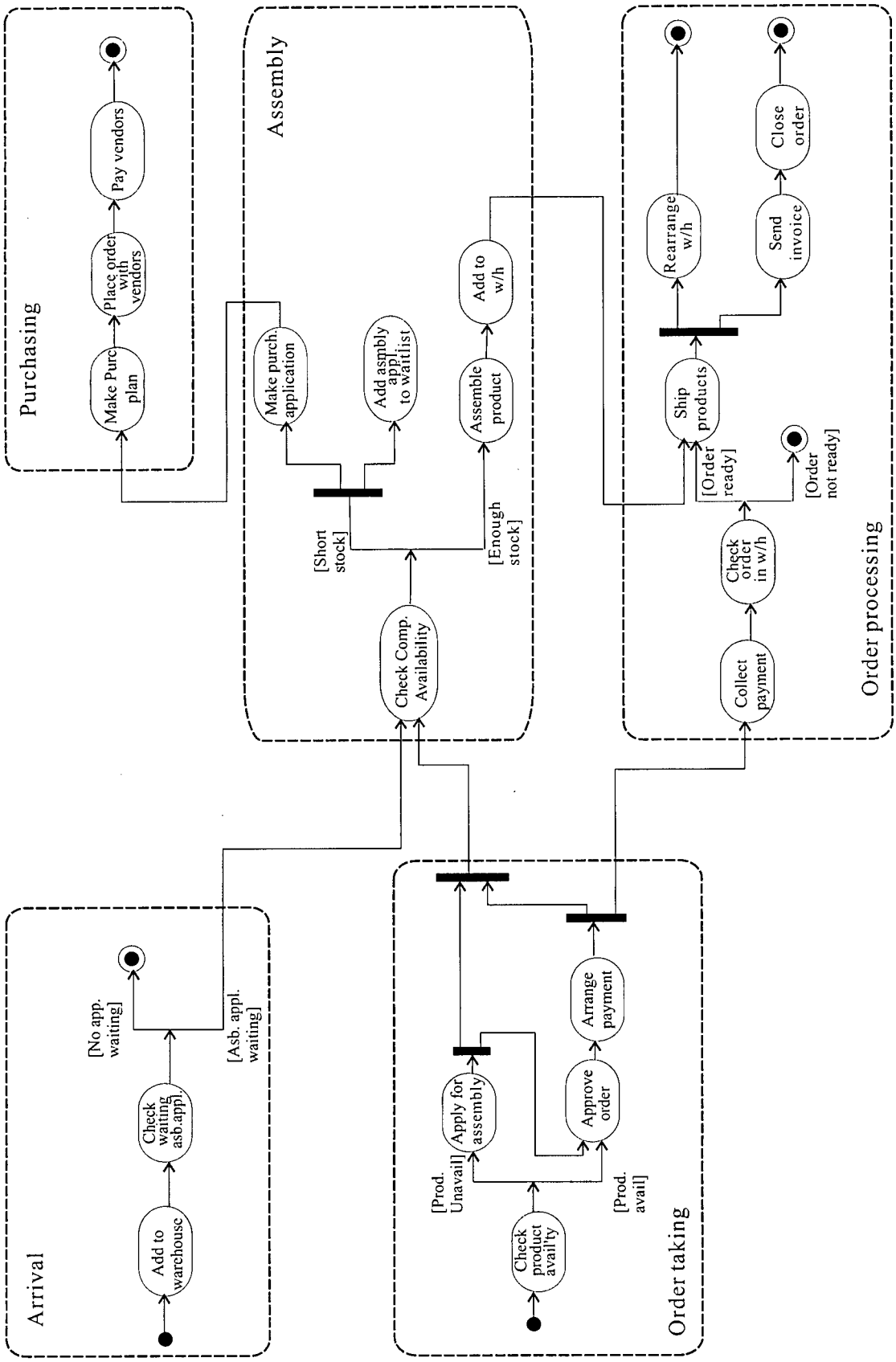


Figure I-5: UML Activity Diagram Model of WaterSytem Example

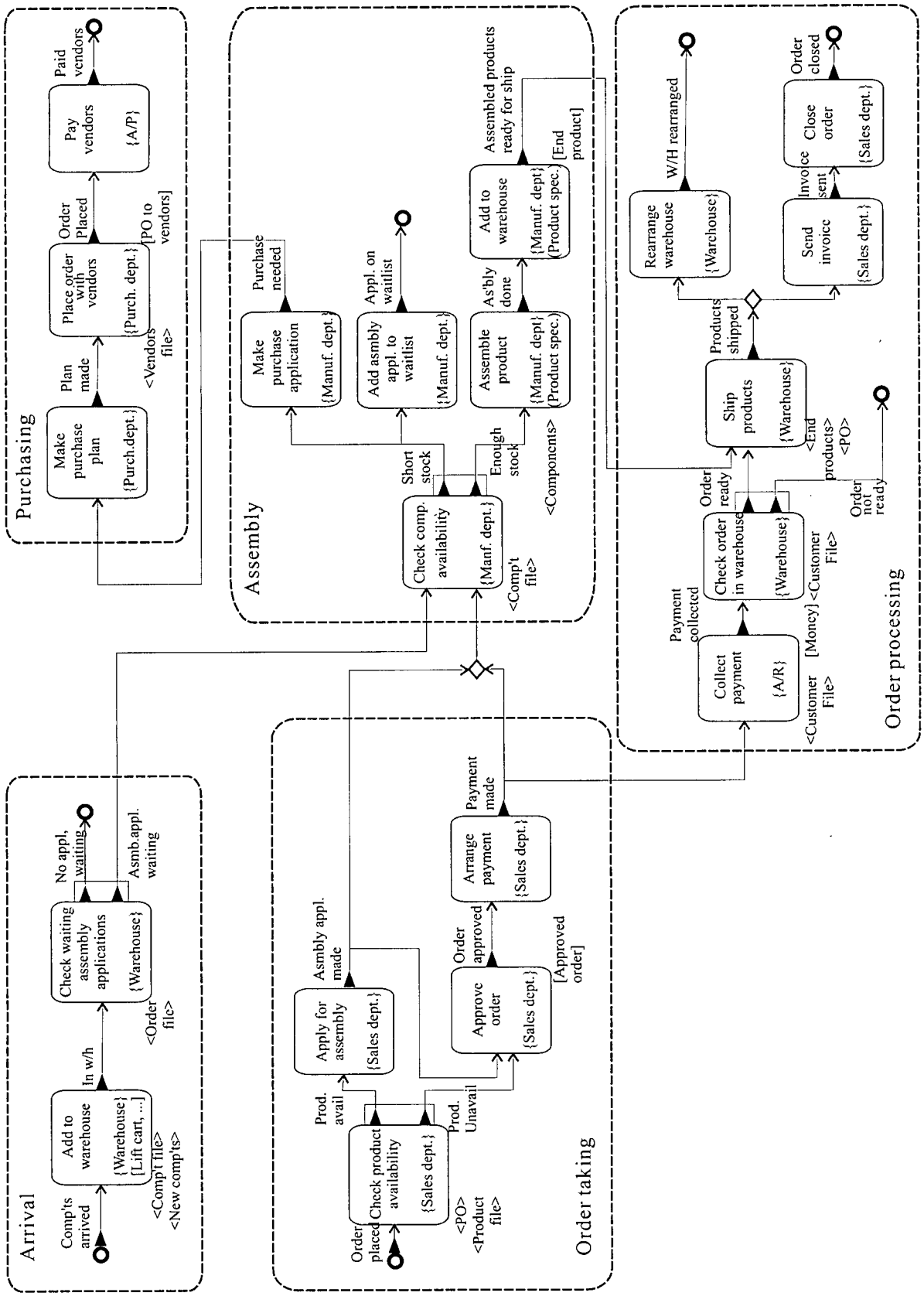


Figure I-6: EDPM Model of WaterSytem Example

Appendix II

ONTOLOGICAL ANALYSIS OF PML-LOGICAL CONNECTORS

The ontological analysis to PML-logical connectors is part of the analysis work in Chapter four. Altogether three situations are investigated here:

1. One event → logical connector → two and more operations

One PML-event of thing A triggers two PML-operations, which could be executed by A, B or C. There are following two cases:

(1) A → A, B (one thing changes the state of another thing and itself)

(2) A → B, C (one thing changes the state of two other things)

(3) A → A, A (one thing changes the state of itself)

(4) A → B, B (one thing changes the state of another thing)

1.1 Logical connector "AND"

For logical connector "AND", case (3) and (4) are not possible because the event of a thing is not able to cause two simultaneous operations of either the thing itself or a different thing at the same time.

For case (1), when thing A ends its state-change at PML-event e, both thing A and thing B start their state-change immediately at the same time. For case (2), when thing A ends its state-change at PML-event e, both thing B and thing C start their state-change immediately at the same time.

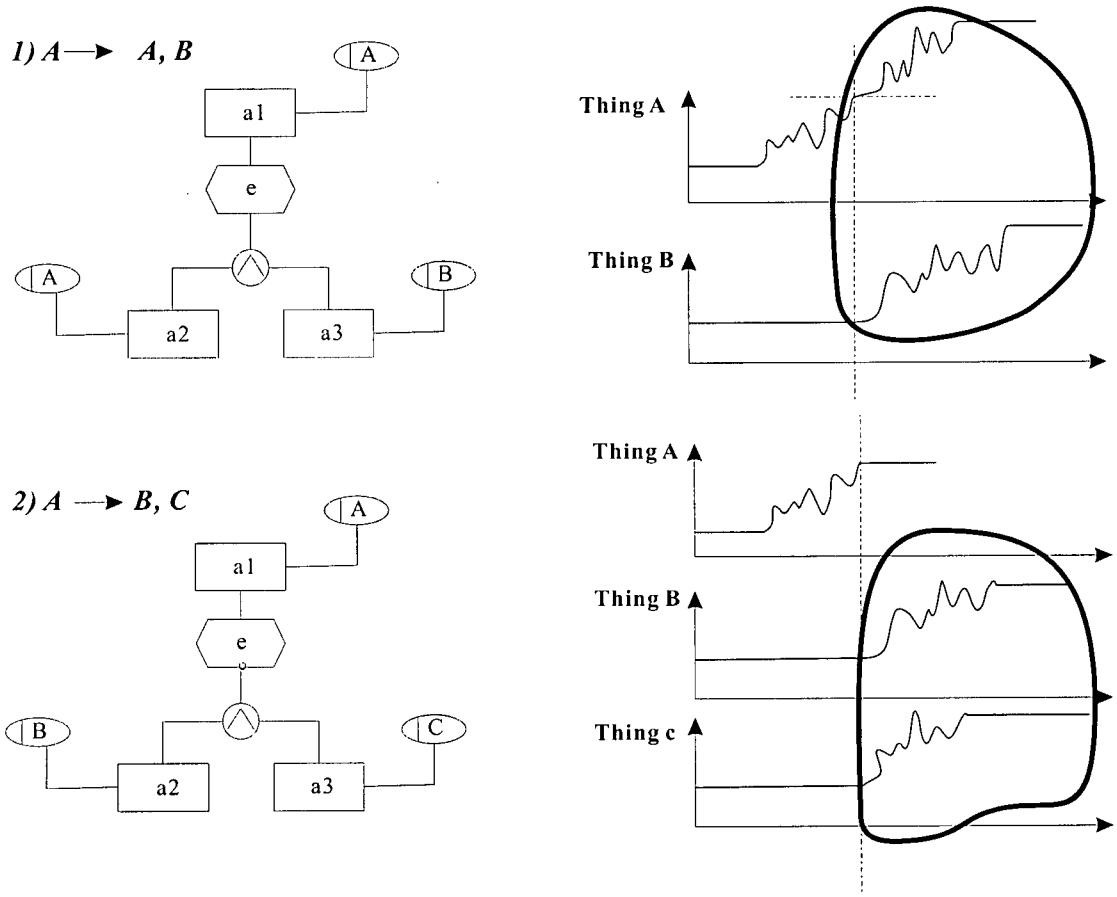


Figure II-1: Ontological analysis to "one event → AND → two operations"

1.2 Logical connector "OR"

For logical connector "OR", case (3) and (4) are not possible to define, because the event of a thing is not able to cause either of the two operations of the same thing.

For case (1), when thing A ends its state change at PML-event e, at least one of thing A and thing B starts its state-change immediately. For case (2), when thing A ends its state-change at PML-event e, at least one of thing B and thing C starts its state-change immediately.

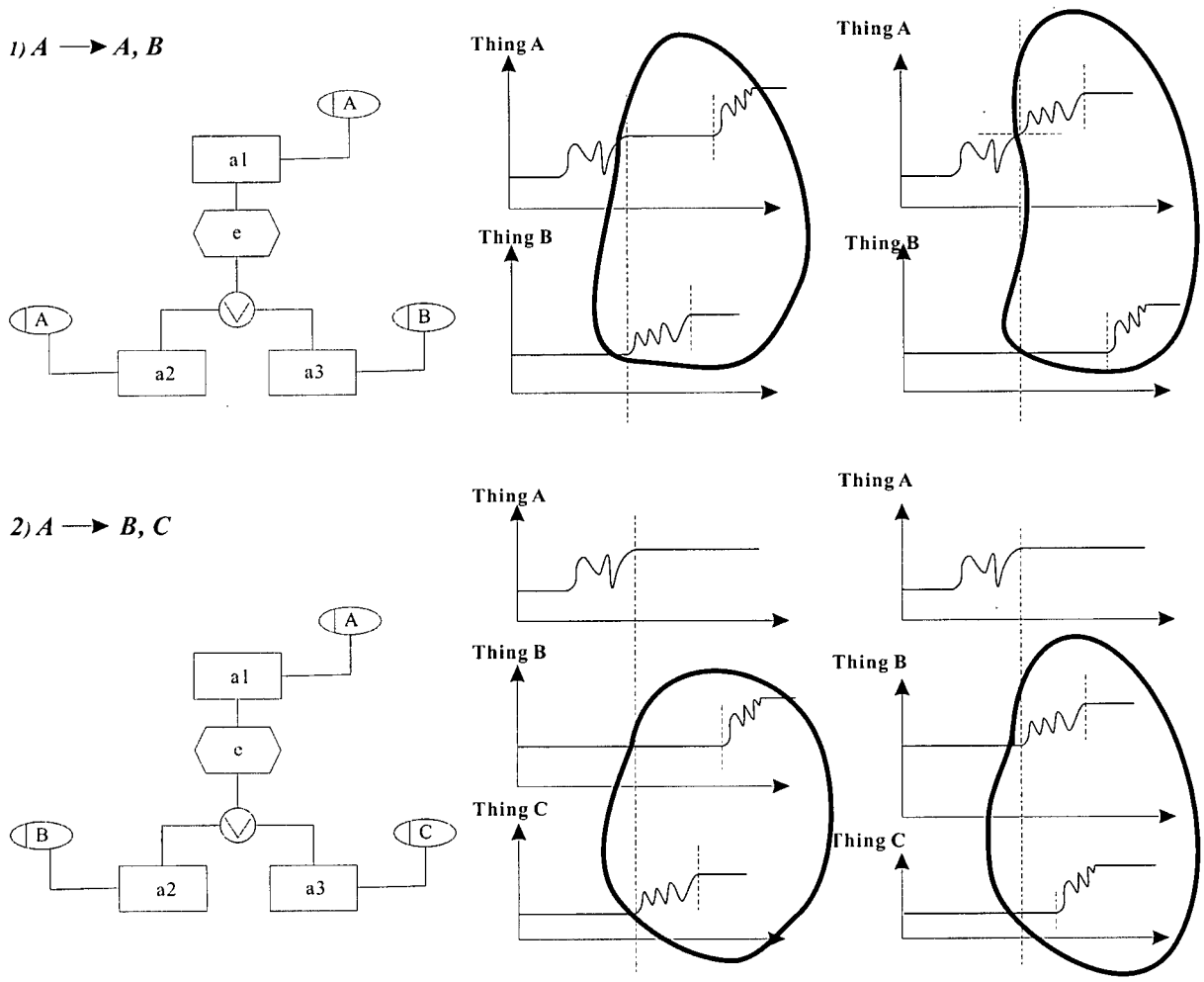


Figure II-2: Ontological analysis to "one event → OR → two operations"

1.3 Logical connector "XOR"

For logical connector "XOR", cases (1), (2), (3), and (4) are all possible.

For case (1), when thing A ends its state-change at PML-event e, either of thing A or thing B starts its state-change immediately. For case (2), when thing A ends its state-change at PML-event e, either of thing B or thing C starts its state-change immediately. For case (3), when thing A ends its state-change at PML-event e, A starts its another state-change immediately, either in one way or in another. For case (4), when thing A ends its state-change at PML-event e, B starts its state-change immediately, either in one way or in another.

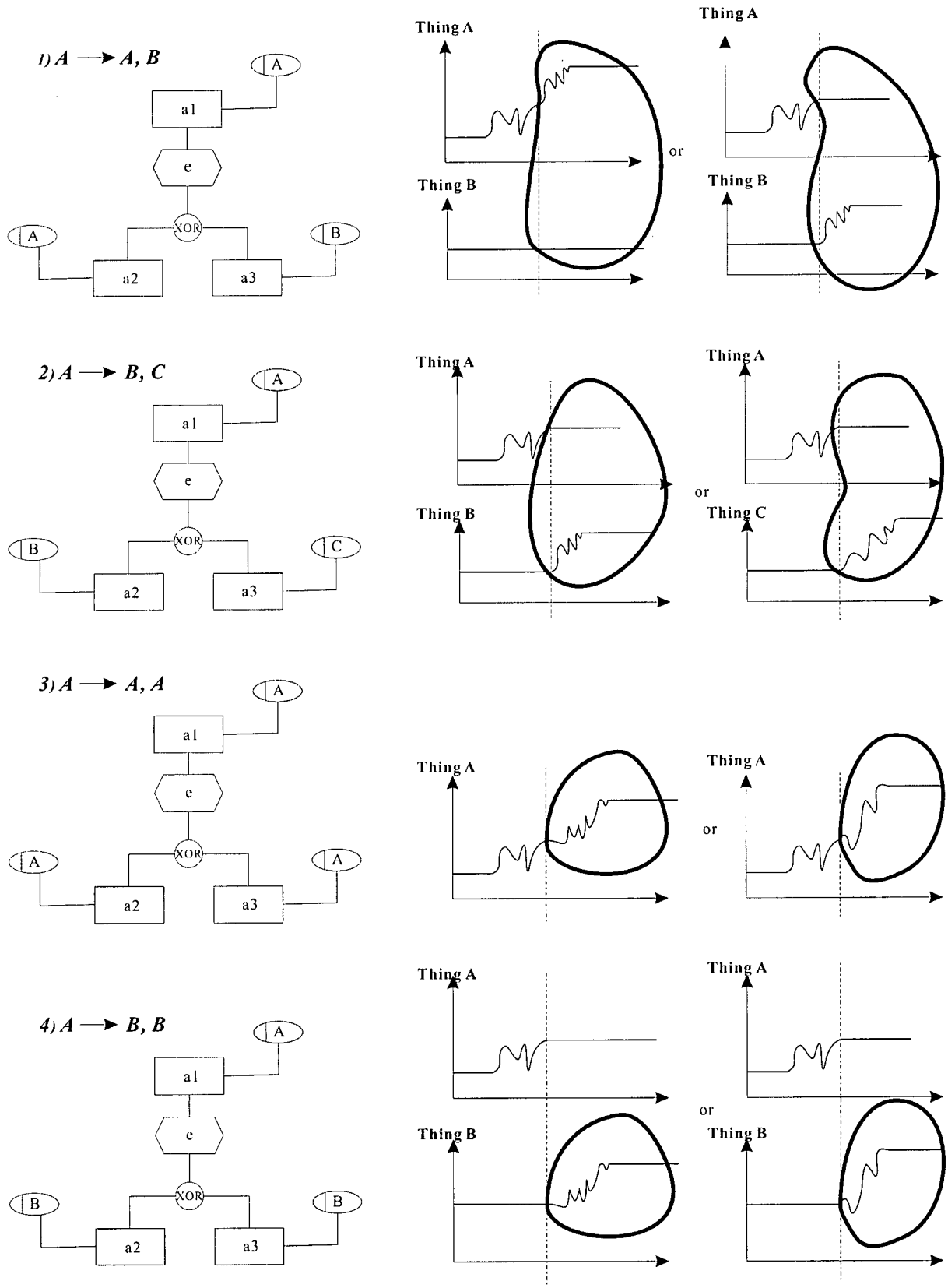


Figure II-3: Ontological analysis to "one event → XOR → two operations"

2. Two and more events → logical connector → one operation

Two or more PML-events of thing A triggers one PML-operations, which could be executed by A, B or C. There are following two cases:

(5) A, B → A (two actors change the state of one of them)

(6) A, B → C (two actors change the state of a third thing)

2.1 Logical connector "AND"

For case (5) A "AND" B -> A, only when both A and B complete their state-changes will A start the next state-change immediately. For case (6) A "AND" B -> C, only when both A and B complete their state-changes will C start its state-change immediately.

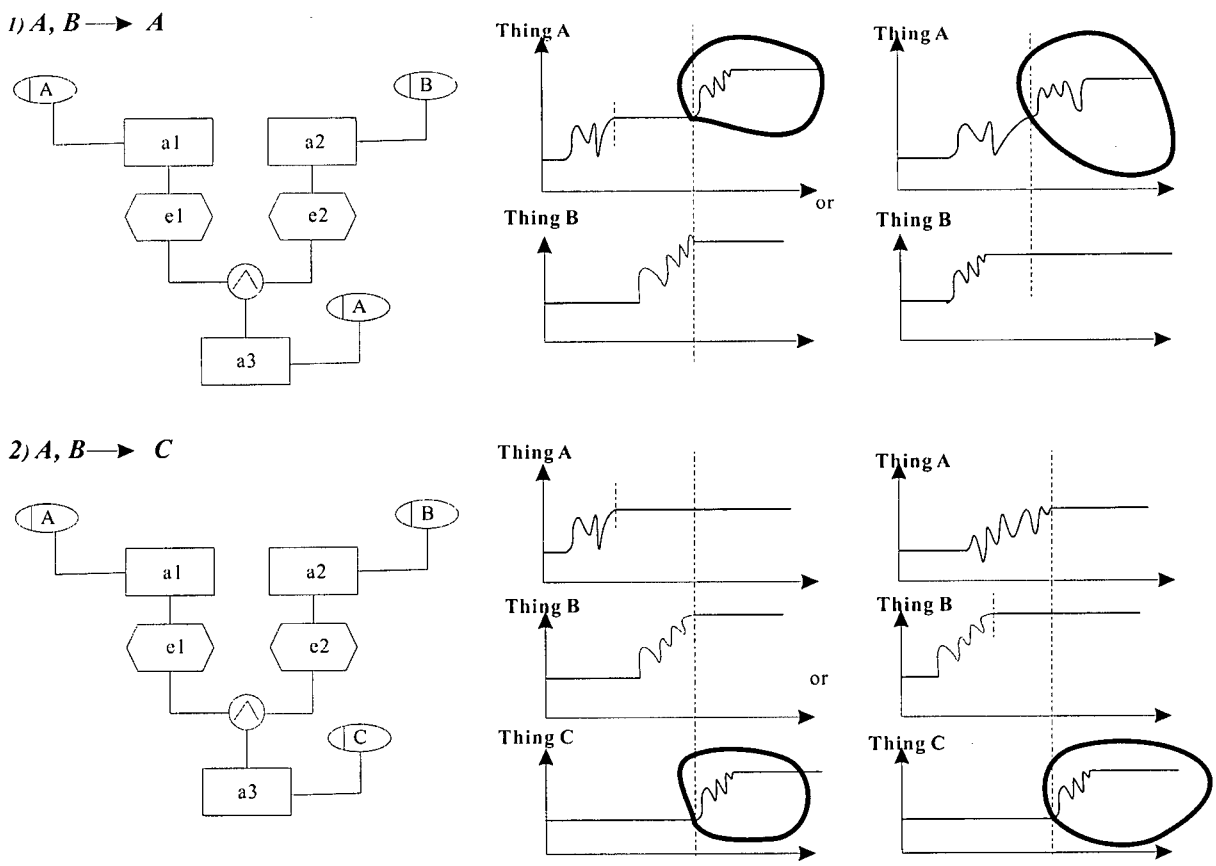


Figure II-4: Ontological analysis to "two events → AND → one operation"

2.2 Logical connector "OR"

For case (5) A "OR" B ->A, whenever at least one of thing A and thing B has completed its state-change, A will start its next state-change immediately. For case (6) A "OR" B -> C, whenever at least one of thing A and thing B has completed its state-change, thing C will start its state-change immediately.

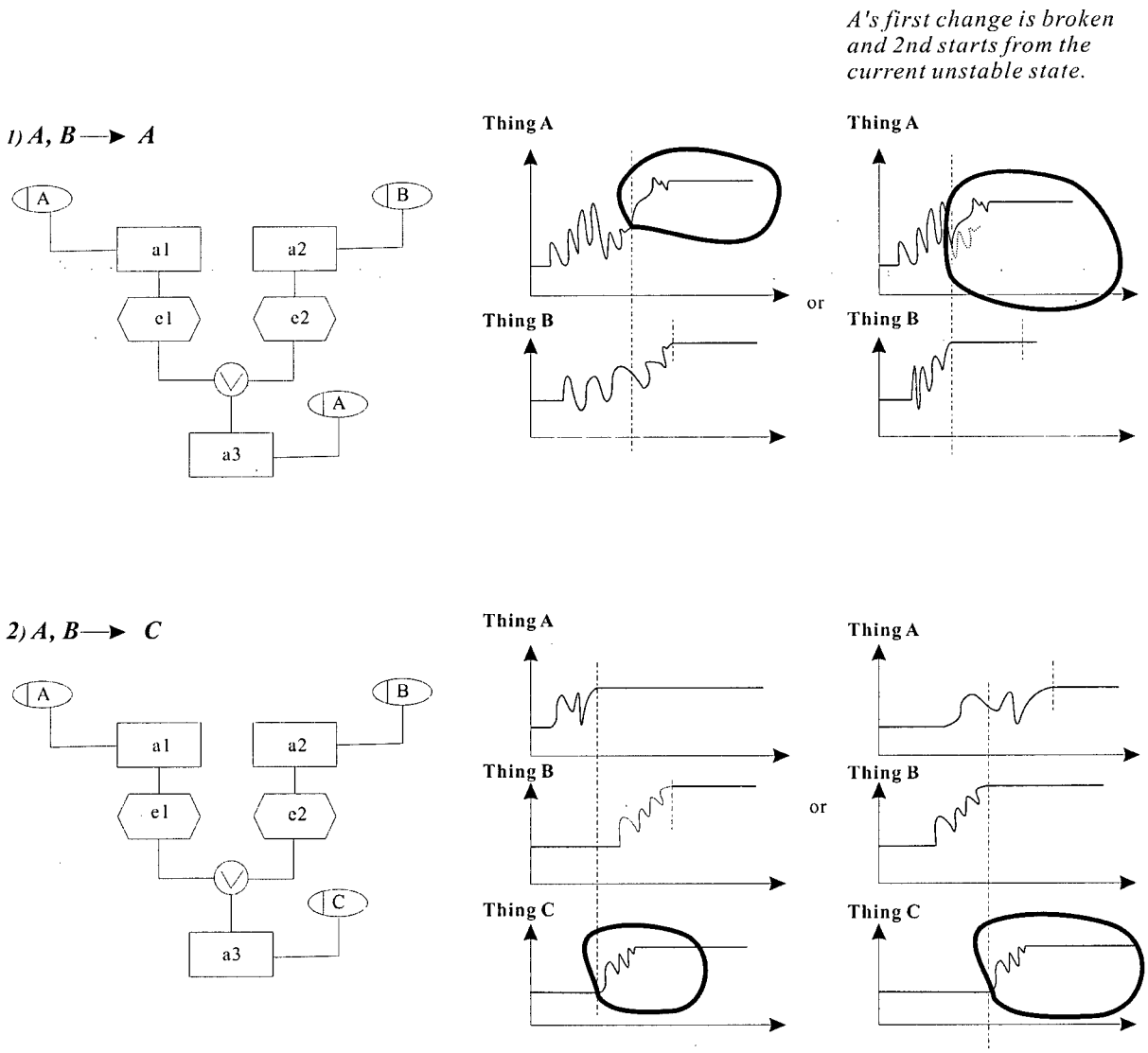


Figure II-5: Ontological analysis to "two events → OR → one operation"

2.3 Logical connector XOR

Logical connector "XOR" in this case is just the variance of "OR".

3. One operation → logical connector → two and more events

This is the situation that one operation could lead to two different stable states. Therefore, the logical connector "AND" and "OR" are not possible, and there are four cases for the logical connector "XOR":

(7) A → A, B (one thing changes the state of either another thing or itself)

(8) A → B, C (one thing changes the state of only one of the two other things)

(9) A → A, A (one thing changes the state of itself either like this or like that)

(10) A → B, B (one thing changes the state of another thing either like this or like that)

3.1 Logical connector "XOR":

For case (7) A -> A "XOR" B, the operation of thing A will lead to two different states: high and low. If A reaches high, A will continue its state-change from the state point of high. On the other hand, if A reaches low, B will start its state-change right after. For case (8) A -> B "XOR" C, if A reaches high, B will start its state-change immediately after. If A reaches low, C will start its state-change right after. For case (9) A -> A "XOR" A, if A reaches high, A will continue its state-change starting from state point of high. If A reaches low, A will continue its state-change starting from state point of low. For case (10) A -> B "XOR" B, if A reaches high, B will start its state-change in one way, and if A reaches low, B will start its state-change in another way.

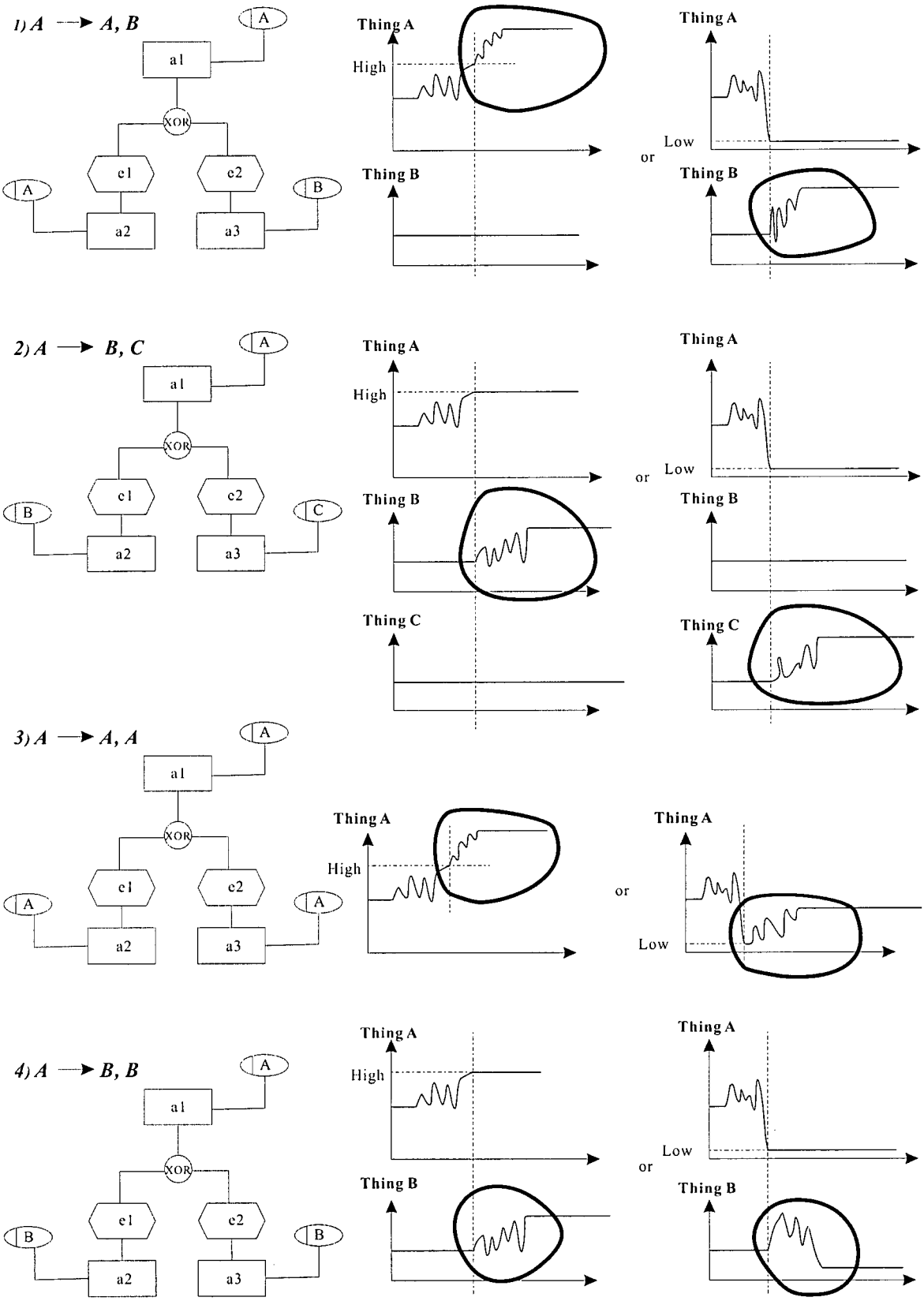


Figure II-6: Ontological analysis to "one operation → XOR → two events"