OBJECT-ORIENTED ACTIVITY-BASED PROCESS MODELLING

by

DAVID LESLIE HOOD

BISM., St. Francis Xavier University, 2001

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

MANAGEMENT INFORMATION SYSTEMS

THE UNIVERSITY OF BRITISH COLUMBIA

February 2005

Abstract

Many organizations seek to take advantage of the opportunities new technologies offer in the current information age. New technologies can have an impact on the fundamental organizational operations, altering well established policies and procedures. New technologies have also been poorly implemented more often than not thus mitigating the opportunities they offer. The poor implementation tends to be a direct result of a lack of proper documentation of organizational processes. This improper documentation means that any system built to support the processes that is based upon the improper documentation will itself be faulty. This thesis develops the OBPM algorithm into an objected graphical modeling language and process. The Activity-based Process Modeling (ABPM) constructs have specific and well-defined semantics for real world business process representation. Further, the change propagation algorithm which is based upon a set of ontologically derived rules is refined to create a systematic process for modeling a business process. The strength of the algorithm is from its ontological real world foundations rather than programming or data design rules of thumb. This thesis also explores the relationship of ABPM and OOEM. Both languages are designed to model a specific view of organizational activity, irrespective of how a later information system artifact will be built. By relating the two grammars using ontological foundations we can acquire greater understanding of an organization without losing information. Finally, this thesis proposes a set of design principles for an ABPM CASE tool that is implementation independent which means that no matter how one decides to implement ABPM if they follow our requirements they will be able to create a tool to fully support the business process model generation process.

TABLE OF CONTENTS

List of Figures

List of Tables

Acknowledgements

This thesis would not have been possible without the assistance and guidance of Professor Yair Wand, my thesis supervisor. I am also indebted to Professor Carson Woo for his many insightful and helpful discussions.

I also wish to extend thanks to two incredible people Young Eun Lee and Hang (Jasmin) Zeng. Without the emotional and personal support of the two of you I would never have made it this far.

Thank you all.

1. Introduction

Documenting the processes of an organization is not a new concept. Organizations document what is occurring in a process in an attempt to capture what happens in the organization. Process models can be used to show where time and money are being spent and can identify things like inefficiencies and bottlenecks. One of the more common uses of process documentation is for business process reengineering (BPR). Organizations that can successfully reengineer their processes often achieve a large significant competitive advantage over their rivals. BPR typically involves a technological integration that was heretofore unseen. However implementing technology is a poorly performed function. Typically information system projects run over budget, over time, or the output does not perform what is required. Typically, the problems that occur can be traced back to incomplete or unclear requirements concerning what occurs in a process (Standish, 1994, CIO, 1997).

1.1 Motivation

In the field of MIS there exist many process modeling methods, including EPC, IDEF0, IDEF3, UML Activity Diagrams, EDPM, Petri Nets, and PERT/CPM to name a few. Of the methods not one has emerged as the dominant paradigm for organizational process modeling. This raises the rather simple question of "why not?"

Looking at the literature about process modeling in the MIS field, we see that there is a growing trend to suggest using several process modeling languages to capture all the relevant information. An example of this would be the work of Bosilj-Vukšić & Hlupić

(2001), in which they suggest using both Petri Nets and IDEF3 for business process modeling. The combination of multiple methods suggests that the methods themselves are deficient and cannot fully represent a business process fully on their own. This apparently answers our question above about why there is not a dominant process modeling methodology, however this raises a new question of "Is there a way to represent a process completely in one modeling methodology?"

Before we answer this question we also need to consider why it is important to create better business process models. An information system is an artifact that represents another 'real world' system. If we can create a better model of the real world system then we should be able to create better artifacts that represent that system.

Developing an information system has been defined as a three step transformation of Analysis, Design, and Implementation[1] (Wand and Weber, 1990, p.125). The analysis stage is where what an information system artifact will do is formalized based upon some process model, i.e. a representation of what will happen is created. The design stage is where the requirements (i.e. the representation) are translated into a model of the information system. The Implementation stage is where the information system is implemented based on the output of the design stage. Everything is built up from the analysis stage, therefore if there are errors made in the analysis stage they will propagate through to the information system itself (Grause and Weinberg, 1989, Wand and Weber, 1995). Thus, better representations lead to better systems.

[1] We acknowledge there could be others, such as development or testing, but for the purposes of this section the three phases are sufficient.

In terms of cost to fix an error the earlier an error is found in system development the cheaper it will be. If we consider the cost ratio the difference is staggering. This difference is also sensible since it is less expensive to, for example, draw a new diagram than it is to build a new system.

Phase in Which Found	Cost Ratio
Requirements	1
Design	3-6
Coding	10
Development Testing	15-40
Acceptance Testing	30-70
Operation	40-1000

Table 1. Relative Cost to Fix an Error (Grause and Weinberg, 1989)

In our work on conceptual modeling we will be focusing above the requirements stage of system development. In theory this should have an even lower cost ratio if errors are made and found here than 1. Hence if we can improve process models and catch more errors there will also be an impact on the bottom line of an organization.

There is a means of fully representing things that has been successfully used in the MIS field. Ontology, a philosophy concerning what exists, has been applied to the area of enterprise modeling and serves as the basis of object-oriented enterprise modeling (OOEM), however it had not been applied to process modeling until recently (see, for example, Wand, Woo, & Jung, 2000, Wand & Woo, 2002 Zhao, 1995).

In an attempt to assist in developing a robust organizational process modeling methodology to solve the problems mentioned above Wang (2002), proposed Ontology-based process modeling (OBPM). Currently OBPM exists only as an algorithm.

1.2 Thesis Objectives

As mentioned, OBPM exists only as an algorithm. This makes both using OBPM difficult and the output difficult to interpret. To address this deficiency we will present a graphical modeling grammar. This notation will be from the perspective of object orientation. Thus the concepts of encapsulation, composition, classification, and communication will be considered.

A graphical notation unto itself is not necessarily useful if it is not clear how to use the notation. Thus following the notation there must be a process for generating an Object-Oriented OBPM. The algorithm presented in Wang (2002) is textual and was not designed with graphics or object-oriented constructs in mind. Hence we will develop a process for generating an Object-Oriented OBPM.

Another point of interest would be that since OOEM and OBPM are both based on the Bunge-Wand-Weber (BWW) Ontology and are used to model different organizational perspectives then they should be related. We will develop a means to convert from the activity-based Object-Oriented OBPM view to the interacting agent-based OOEM view using the BWWP ontology as the basis of conversion.

Further, a CASE tool for OOEM has been developed. It stands to reason, since there is a relationship between OOEM and OBPM, and there exists a CASE tool for OOEM, that it is possible to develop a CASE tool for OBPM. This also leads to the conclusion that if they can both be represented in a CASE tool, then conversion between OOEM and

OBPM can also be captured in a CASE tool. Therefore a set of design principles for an Objected-Oriented Ontology-Based Process Modeling CASE tool will be developed.

The objectives of this thesis can be summarized as follows:

1. Formalize OBPM into an object-oriented graphical modeling grammar.
2. Present a process for generating object-oriented OBPM diagrams
3. Explore and discover the relationship between OOEM models and Object-oriented OBPM models and present a means of conversion between them using the BWWP ontology as a basis of conversion.
4. To present a set of design principles for an Object-Oriented OBPM CASE tool and link it to the architecture of an OOEM CASE tool.

1.3 Thesis Outline

This thesis consists of five chapters.

Chapter two provides an introduction to process modeling and the related terminology. It also presents the different aspects of process modeling. Examples of process modeling techniques are presented to illustrate the different process modeling aspects, and to illustrate flaws present in ubiquitous methodologies.

Chapter three concerns Ontology-based Process Modeling (OBPM). It formalizes OBPM into an object-oriented modeling grammar called object-oriented activity based process modeling(ABPM). It then introduces graphical modeling grammar for representing object-oriented activity-based process models. It also presents a process for how to create an ABPM. The chapter includes an example to illustrate these points.

Chapter four introduces Object-oriented Enterprise modeling (OOEM). Since the foundations of OOEM and ABPM are similar, this chapter links the two methodologies. It also provides steps relating to how to switch between an ABPM and an OOEM and vice versa. It will include a running example to demonstrate the results.

Chapter five is a look at the design principles for the implementation of ABPM as a CASE tool. It discusses what to add to an existing OOEM CASE tool to both represent ABPMs and for conversion between OOEMs and ABPM. It ends with a discussion of the limitations of the proposed design.

Chapter six concludes the thesis with a summary and discussion of its contributions. It also suggests further research that can be done.

2. Process Modeling Introduction

2.1 Introduction

Information systems offer the potential to supplement the ability to increase revenues while lowering costs in the value creation process of a business by greatly improving both the efficiency and the effectiveness of a process. This is where business process modeling comes in to play in the information systems field. From the business process engineering perspective a process model can serve as: a focus for discussion, a way of communicating a process to others, a starting point for analysis, a starting point for design, a baseline for monitoring process improvement, and control for a real world process (Huckvale and Ould, 1994).

Before we present the object-oriented ontology-based process modeling grammar, we will provide a discussion of the basic concepts behind business process modeling. We will also discuss the basic terminology used in business process modeling. We will then look at what has been proposed for the ideal process model. We then delve into the ontological perspective, what it is, and why it is a better perspective than other views. We will then look at several business process modeling languages. These languages include Petri Nets, Integrated Definition 3 (IDEF3), and Event-controlled Process Chains (EPC). The languages are some of the most ubiquitous process modeling techniques. These languages will be analyzed in terms of their (process-oriented) ontological completeness and (process-oriented) ontological clarity. Our goal is to demonstrate how the languages are deficient and why object-oriented ontology based process modeling will be beneficial.

2.2 *Definition of Business Process modeling*

All businesses strive to achieve their goals via some sort of process. A business process can be defined as the sequence of activities that lead to value creation for the consumer of a business' goods and services. If a good or service is perceived to have a higher value then consumers should be willing to pay more for the good or service. That said it stands to reason that businesses will want to analyze their processes, with the goal of improving them, so that their profits will be increased.

It has been argued (e.g. Hui, 1997, Romney, 1994, VBM, 2004) that redesigning a business process can have many impacts on and benefits for an organization. A brief non-exhaustive list of the impacts and benefits is presented below in table 2-1.

Impact of redesigned process	**Benefits of the impacts**
1. Task elimination 2. Bottleneck and delay elimination 3. Parallel work enabled 4. Eliminate rework and redundancy 5. Decreased defects 6. Eliminate staff 7. Less time spent on non value-added activities	1. Improved productivity 2. Reduced cycle times 3. Reduced costs 4. Improved customer service 5. Improved quality and consistency 6. Increased revenues/charging higher prices 7. Increased competitiveness 8. Improved forecasting 9. Better capacity utilization 10. Quicker delivery of new products and services 11. Greater workload capacity

Table 2-1 Effects of a redesigned process (partially adapted from, Hui, 1997, p.8)

Before we define business process modeling we need to look at and consider modeling itself. A model is an abstraction from the real world of something in the real world. In this abstraction we leave out the facets of whatever we are modeling that are not relevant, and emphasize those that are pertinent.

If we combine this definition of modeling with our above definition of a business process, we can define business process modeling as: the creation of an abstract representation of the real or proposed sequence of value creation activities performed within an organization in order to achieve its business goals. Adapting from Wang (Wang, 2002), the business process model will therefore include: participants in the business process, the events that start activities in the business process, the activities triggered by the events in the business process, the sequence of the activities of the business process that will be performed, the sequence that events in a business process occurred, the resources that are utilized in an activity of a business process, and the inputs and outputs of each activity of the business process.

2.3 The Terminology

In an analysis of process modeling methodologies, Wang (Wang, 2002) found that traditionally, the following constructs have been used in business process modeling: process, agent, non-agent or resource, activity, operation, event, state, data, logical connector, business rule, input, and output.

However, Wang (Wang, 2002), points out that many languages either do not include all of these constructs (construct incompleteness), use many elements of the modeling language to represent the same construct (construct redundancy), or use the same construct to represent many elements (construct overload). This causes process models using these languages to be incomplete and/or ambiguous.

2.4 The Ideal Process Model

Various process modeling methodologies focus on different aspects of what is being modeled (Wang, 2002). The majority of process modeling languages fall into four categories (Huckvale and Ould, 1994, citing Curtis et al., 1992):

1. Functional. A functional process modeling language is concerned with representing what activities are being performed and the dataflows that connect them. Examples in this category include EPC (Event-controlled Process Chains), IDEF3 (Integrated DEFinition 3), and EDPM (Event-Driven Process Modeling).
2. Behavioural. A behavioural process modeling language is concerned with representing when activities occur. They use sequencing, feedback loops, iteration, decision making, triggers, and the like. Examples in this category include PERT/CPM (Project Evaluation Review Technique/Critical Path Method), Petri Nets, and EPC.
3. Organizational. An organizational process modeling language is concerned with representing where and by whom activities are performed. They also include the physical communication mediums and storage media. Examples in this category include EPC, and EDPM.
4. Informational. An informational process modeling language is concerned with representing the data entities that are generated or manipulated by a process, such as documents, data, artifacts, and products. This includes their structure and

interrelationships. Examples of this include DFD (dataflow diagrams), IDEF3 and EDPM.

Due to the fact that these four categories cover different areas, using a methodology from each category should cover all the information necessary to fully represent a process. However (as noted by Wand & Weber, 2002) this leads to potential issues of redundancy and sufficient coverage. Redundancy refers to the same construct being represented more than once. Sufficient coverage refers to all the relevant items of interest being captured.

This leads to the premise if a modeling language included all of the constructs from section 2.3 it would thus cover the functional, behavioural, organizational, and informational perspectives[2]. This is one of the bases for Ontology-based Process Modeling (OBPM). OBPM will be discussed in chapter 3.

2.5 *Ontology*

Ontology is a branch of philosophy about what exists in the real world. It is of prime concern when it comes to information systems modeling since the model should accurately reflect what does or will exist in the real world[3]. If the model does not accurately reflect this, then anything created based on the model (in this case an information system) will itself be deficient.

[2] Data itself would be left out of the ideal model since as Huckvale and Ould (1994) note data is typically used to record the state of a process due to people having poor memories or as a means of implementing a transaction. The transaction itself is important not how it is implemented at this level.

According to Wand and Weber (Wand and Weber 1993, 1995) for a grammar to be ontologically expressive, it needs to be both ontologically complete and ontologically clear. Ontological completeness refers to there being at least one construct in the grammar for every ontological construct. If this does not occur then construct incompleteness (or construct deficit) occurs.

Ontological clarity refers to how clearly an ontological construct is represented in a grammar. Ontological clarity can be undermined via construct overload, construct redundancy, and construct excess. Construct overload occurs when an element of the grammar represents more than one ontological construct. Construct redundancy occurs when more than one element of a grammar represents a single ontological construct. Construct excess occurs when there is a construct in the grammar that does not represent any ontological construct.

Typically when a modeling grammar is ontologically deficient predictions can be made as to where the language will suffer in capturing information about the domain of interest. Approaches to solve the deficiencies usually involve using more than one grammar to model what is going on. However there is no formal basis for selection of which grammars to use other than rules of thumb or things seeming to fit and workout.

[3] In this case what will exist is not definite, but rather refers to a proposed information system that may or not be built.

Object-Oriented Enterprise Modeling and Ontology-Based Process Modeling are both based on what has come to be known as the Bunge-Wand-Weber (BWW) Ontology. The main principles of the BWW ontology can be summarized as follows (Wand and Woo, 1999):

- The world is comprised of things
- Things possess properties
- Things interact
- Every thing changes and every change changes things
- Things with similar properties can be grouped into classes

The constructs of BWW Ontology can be organized into four categories as shown in table 2-2 below (Zhao, 1995, Wang 2002). A complete discussion of the BWW constructs can be found in appendix C.

Category	BWW Ontology Construct
Static model of a substantial individual	1. Thing, composite thing 2. Property, intrinsic or mutual property, hereditary or emergent property[4] 3. State, conceivable state space, state law 4. Class, kind, natural kind
Dynamic model of a substantial individual	1. Event, conceivable event space 2. Transformation, transformation law 3. History
Static model of a system	1. Coupling 2. System 3. System Composition 4. system environment 5. system structure 6. system decomposition 7. subsystem 8. level structure
Dynamic model of a system	1. Stable and unstable state 2. Internal event and external event 3. Well defined event and poorly defined event

Table 2-2 BWW Ontology Constructs

[4] Here the use of 'or' does not mean intrinsic is interchangeable with mutual and hereditary is interchangeable with mutual. 'Or' in this sense is a restriction that the property can be one of them but not both.

2.5.1 BWWP Ontology

Some of the BWW ontology concepts are closely related to the concepts of process. The process related BWW ontology concepts have been adapted to create the BWWP (BWW Process) ontology. The BWWP constructs include the 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 constructs added in BWWP are: actor, non-actor, actuator, propagator, and process (Wang 2002).

The new constructs of BWWP are explained as follows. A thing can be either an *actor* or a *non-actor*. A non-actor does not change the state of any thing including the non-actor itself. Only an actor can change the state of a non-actor. An actor is an *actuator* if it changes the state of at least one other thing (actor or non-actor). An actuator is a *propagator* if it changes the state of at least one other actor. A *process* is a set of actors and non-actors that interact with one another. A process is activated one or more actors is changed from a stable state to an unstable said. A process can be triggered by one or more events. When all actors and non-actors are in stable states a process is completed (Wang, 2002).

2.6 Current Business Process Modeling Techniques

The purpose of this section is to demonstrate the differences between modeling methodologies and highlight their ontological deficiencies. The methodologies we have chosen include Colored Petri Nets, Integrated Definition (IDEF3), and Event-controlled Process and Chains (EPC). Petri Nets and IDEF3 were chosen due to the aforementioned suggestion of using them together. EPC was chosen due to the fact it was developed and is used by one of the largest providers of ERP providers SAP. It is beyond the scope of this paper to perform an ontological analysis on all business process modeling techniques. In order to perform this comparison we require a common reference point. The modeling languages will be analyzed using the BWWP ontology. For those unfamiliar with the grammars, an example using each grammar is provided in Appendix D.

2.6.1 Colored Petri Nets

Table 2-3 maps the Colored Petri Net constructs to BWWP Ontological Constructs.

It is readily apparent that Colored Petri Nets suffers from many ontological deficiencies. First of all, construct deficit is present with respect to properties. A token is possessed by a place, making it a property of a place. However there is no concept of an intrinsic token possessed naturally by a place, any token can leave or enter a place based on the firing rules established. There is also no concept of an intrinsic arc or intrinsic firing rule. Other places and transitions need to be established to have arcs and firing rules. Since there are no intrinsic properties, there can be no hereditary or emergent properties.

The second ontological deficiency Petri Nets suffers from is construct redundancy. Consider the BWWP construct of a property. Where there is a mapping, there is overload between arcs and tokens. Is a property a token, or one of the variations on a directed arc?

BWWP Construct	Colored Petri Nets Construct
Thing	Place
Simple Thing	Place
Composite Thing	Place
Actor	Place
Non-actor	Place
Actuator	Place
Propagator	Place
Property	Token, Directed Arc, Inhibitor Arc, Clearing Arc, Priority Transition
Intrinsic property	N/A
Mutual property	Token, Directed Arc, Inhibitor Arc, Clearing Arc, Priority Transition
Hereditary property	N/A
Emergent property	N/A
State	Distribution of Tokens
Stable state	No transitions are enabled
Unstable state	One or more transitions are enabled
Event	Transition
Internal event	Transition
External event	Transition
Transformation	Transition
Law	Directed Arc, Inhibitor Arc, Clearing Arc, Priority Transition
State law	Directed Arc, Inhibitor Arc, Clearing Arc, Priority Transition
Transformation law	Directed Arc, Inhibitor Arc, Clearing Arc, Priority Transition
Process	Colored Petri Net

Table 2-3 BWWP Ontological Analysis of Colored Petri Nets

Finally Petri Nets also suffers from construct overload. Arcs can represent both properties and laws. The construct of a transition is also overloaded. Is a transition a transformation or an event?

Since it suffers from construct deficit, construct redundancy, and construct overload Petri Nets are both ontologically incomplete and ontologically unclear.

2.6.2 Integrated Definition 3

IDEF3 has two main components for modeling: process flow description and object state transition networks. Table 2-4 maps the IDEF3 constructs to BWWP ontological constructs.

We start the analysis of IDEF3 with the process flow description diagram. Process flow description diagrams suffer from construct deficit, construct overload, construct redundancy, and construct excess. Construct deficit is readily apparent from the complete lack of a construct to represent a thing. Also, properties, states, and events are only partially represented. The UOB, links, junctions, and referents are all overloaded. That is, they are used to represent more than one ontological construct. Construct redundancy occurs with property, law, and process. Lastly, the excess constructs occurs with decomposed UOB, UOB numbering, link numbering, junction numbering, partial descriptions, elaborations, and notes. Thus process flow description diagrams are neither ontologically complete nor ontologically clear.

Object state transition networks suffer from construct deficit, construct overload, construct redundancy, and construct excess. Construct deficit exist since there is no representation of a property and only partial representations of thing and event. Construct overload is present since a referent can be used to represent both a transformation and an event. Construct redundancy occurs with state, event, law, and process. Construct excess exists since UOB numbering, link numbering, junction numbering, partial descriptions, elaborations, and notes have no ontological meaning.

BWWP Construct	Process Flow Description	Object State Transition Networks	Enhanced Transition Schematics
Thing		Object	Object, first order objects, second order objects
Simple Thing		Object kind symbols	Object kind symbols
Composite Thing		Hiding Object State information	Hiding Object State information
Actor			Individual symbol
Non-actor			Individual symbol
Actuator			
Propagator			
Property	Links,Junctions,referents	Links,Junctions,referents	Links,Junctions,referents
Intrinsic property			
Mutual property	Links		Relation
Hereditary property			
Emergent property			
State	Activation plots	Object state symbols, interval diagram	Object state symbols, interval diagram
Stable state		State conditions, Exit Conditions	State conditions, Exit Conditions
Unstable state		State Conditions, Exit Conditions	State Conditions, Exit Conditions
Event	UOB	UOB, scenario, Referent	UOB, scenario Referent
Internal event			
External event			
Transformation	UOB	Link, referent	Link, referent
Law	Links, junctions, referents	Link, junction, referents	Link, junction, referents
State law	Simple precedence links, constrained precedence links, dashed links, junctions	Weak transition link, strong transition link, call and continue referent	Weak transition link, strong transition link, call and continue referent
Transformation law	Simple precedence links, constrained precedence links, dashed links, junctions	Entry Conditions, Transition Condition, referent, junctions call and wait referent, referents attached to the same point, temporal indeterminancy marker	Entry Conditions, Transition Condition, referent, junctions call and wait referent, referents attached to the same point, temporal indeterminancy marker
process	Scenario, activation plots	Scenario, Object schematics, complex transition schematic	Scenario, Object schematics, complex transition schematic
	Decomposed UOB, UOB numbering, link numbering, junction numbering, partial descriptions, elaborations, notes	UOB numbering, link numbering, junction numbering, partial descriptions, elaborations, notes	UOB numbering, link numbering, junction numbering, partial descriptions, elaborations, notes

Table 2-4 BWWP Ontological Analysis of IDEF3

Enhanced transition schematics suffer from the same problems as object state transition networks, except the construct deficit is lower. Enhanced transition schematics partially represent properties, and better represent a thing than object state transition networks.

Since the IDEF3 diagrams are meant to be used together, we need to consider whether they achieve minimum ontological overlap (MOO) and maximum ontological coverage (MOC). MOO is when an ontological construct is represented in only one diagram with the goal of reducing representations of a domain that are in conflict. MOC is when in the combination of diagrams covers all phenomena with the goal of having a complete representation of the domain (Wand and Weber, 2002, citing Green 1996). MOO is not achieved in IDEF3. An example of the lack of MOO is process flow descriptions use activation plots to represent states whereas the object transition diagrams use object state symbols and interval diagrams. MOC is not achieved is IDEF3. An example of the lack of MOC is that none of the diagrams have a construct to represent an intrinsic property.

2.6.3 *Event-controlled Process Chains*

Table 2-5 Maps the EPC constructs into ontological constructs

To begin, EPC suffers from construct deficit. The concepts of Property, Thing, and Event are only partially represented. Since things are what perform actions, are affected, etc. and things are not fully represented we cannot fully represent what performs actions, are affected, etc. When we look at event it is not immediately clear about those that are internal or external to the domain of interest. It can be argued that the event that starts the process chain is external, but how is it known which events generated in the domain affect something outside the domain.

BWWP Construct	EPC Construct
Thing	Organizational Unit
Simple Thing	
Composite Thing	
Actor	Organizational Unit
Non-actor	
Actuator	
propagator	
Property	
Intrinsic property	Information/Material flow
Mutual property	
Hereditary property	
Emergent property	
State	
Stable state	No Events occur
Unstable state	Event
Event	Event
Internal event	
External event	
Transformation	Task
Law	Control Flow + Connectors
State law	Control Flow + Connectors
Transformation law	Control Flow + Connectors
Process	Process
	Information Object, Organization assignment

Table 2-5 BWWP Ontological Analysis of EPC

To end EPC also suffers from construct excess since the information object construct and organization assignment constructs have no BWWP ontological meaning.

Since it suffers from construct deficit, and construct excess EPC is both ontologically incomplete and ontologically unclear.

2.7 *Summary*

If we just do a simple count of the number of constructs each grammar has (see table 2-6) our three grammars never achieve MOC even when all three are used together.

		IDEF3				
BWWP Construct	**Colored Petri Nets**	**Process Flow Description**	**Object State Transition Networks**	**Enhanced Transition Schematics**	**Event-Driven Process Chains**	**Total**
Thing	2	0	1	3	1	7
Simple Thing	2	0	1	1	0	4
Composite Thing	2	0	1	1	0	4
Actor	2	0	0	1	1	4
Non-actor	2	0	0	1	0	3
Actuator	2	0	0	0	0	2
propagator	2	0	0	0	0	2
Property	5	3	3	3	0	11
Intrinsic property	0	0	0	0	1	1
Mutual property	5	1	0	1	0	7
Hereditary property	0	0	0	1	0	1
Emergent property	0	0	0	0	0	0
State	1	0	2	2	0	5
Stable state	1	0	2	2	1	6
Unstable state	1	0	2	2	1	6
Event	1	1	3	3	1	9
Internal event	1	0	0	0	0	1
External event	1	0	0	0	0	1
Transformation	1	1	2	2	1	7
Law	1	3	3	3	2	12
State law	5	4	3	3	2	18
Transformation law	5	4	7	7	2	25
Process	5	2	3	3	1	14
No mapping	0	7	6	6	1	20

Table 2-6 Demonstrating MOO and MOC

Also the multiple grammars really begin to make problems with MOO grow. To solve this problem we suggest using one grammar designed using the BWWP ontology namely ontology based process modeling (OBPM). OBPM is discussed in the next chapter.

3. Object-Oriented Activity-Based Process Modeling (ABPM)

3.1 Introduction

The purpose of this chapter is threefold. The first goal is to introduce Ontology-Based Process Modeling (OBPM). The second goal is to present a modeling grammar that can be used to generate object-oriented activity-based process modeling diagrams. The third goal is to introduce a modeling process to guide in the construction of ABPM diagrams in a systematic way. This chapter closes with an example of an ABPM diagram based on the ACME Warehouse Management case.

In order to accomplish the goals of this chapter the following concepts will be introduced: agent, activity, operation, attribute, resource, and law. The relationships between the concepts will also be developed.

3.2 Ontology-Based Process Modeling (OBPM)

After having identified so many deficiencies with other ubiquitous process modeling methodologies the question of is there a better method to capture business processes remains? The answer is yes and no. Ontology-based process modeling (OBPM) has been proposed. OBPM attempts to capture what things are involved in a process, what they do, and what is done to them. It does have the advantages of being real world rather than information systems oriented which allows for representations of a process independent of how it will be implemented, ontologically complete and clear, and formally defined rules to follow for constructing a business model to eliminate ambiguity and confusion, however this methodology is neither graphical nor is the output easy to understand. The results of an OBPM look like computer code. As noted, there are cognitive advantages to visually representing information. “The human ability to extract information from visual scenes is much more fundamental than is our ability to manipulate data verbally or arithmetically” (Zhang 1998, citing Schwartz and Howell, 1985) “Model diagrams enable the analyst to extract process, understand, and respond to much relevant information. The transfer of information is fast, accurate, and the user learning curve is minimized; the analyst can thus build a conceptual model of the problem with fewer perceptual errors” (Zhang 1998, citing Brown 1988) “The diagrams can also serve as the interface between a domain analyst with his/her customers. The visual model is indeed worth thousands of words in terms of communicating with customers.” (Zhang, 1998, p.22)

Another problem with the current incarnation of the OBPM algorithm is that it is based upon a set of modeling integrity rules. The premise behind basing the algorithm on a set of integrity rules is that if the algorithm is followed correctly then any models generated using the algorithm will be correct. The flaw with this approach is that not all of the modeling integrity rules are used in the OBPM algorithm.

The rest of this chapter is organized as follows: to begin there will be a discussion of the OBPM algorithm, this will be followed by a discussion concerning object-orientation and its application to OBPM, a notation for creating object-oriented activity-based process models will be introduced, and then a process for creation and validation of object-oriented activity-based process models will be presented.

3.2.1 OBPM Algorithm

The OBPM algorithm is presented here. It has three parts; the main routine, the affected thing subroutine, and the decompose subroutine. The algorithm serves to identify: Events (what triggers instability in things in a process), Activities (what happens in response to events), Operations (the transformations that occur), and Resources (what is used during transformations)

The main routine begins the modeling process. The events that trigger the process from the environment are identified. For each triggering event the things in the environment that are affected should be identified. Then the subroutine affected thing should be invoked for each thing that was affected.

The affected thing subroutine begins with if a thing is changed from a stable to a stable state it is considered to be a resource and this subroutine exits. If the thing changes from stable to unstable it is considered to be an agent. Agents invoke the decompose subroutine. When the decompose subroutine exits and returns to the affected thing subroutine, the affected thing subroutine will exit and return to the main routine.

The decompose subroutine identifies the sequence of events that occur from when a thing is affected (becomes unstable) until the agent is finished changing (becomes stable). The entire sequence of events is known as an activity. Each event in the activity is an operation. If in the course of the activity other agents are affected then these agents will themselves invoke the affected thing subroutine.

The output of the algorithm is a series of lists. The first list is the agent list, those agents that participate in the process. For each agent there is also a list of the activities the agent performs and their sequence (called the Activity List) and a list of the operations for each activity and their sequence (called the Operations List). The algorithm also produces a resource list of the resources that are used in the process. The last list the algorithm produces is an event list which is a list of the events that happen in the process, and their sequence.

3.3 Object-Oriented Activity-Based Process Modeling Grammar

The concept of object orientation has been applied to MIS and is the basis of object-oriented systems. There also exist graphical modeling languages that are object-oriented such as OOEM. Object-oriented modeling languages can also be used to create a model of what is occurring independent of any implementation. It has been noted that object-orientation allows a process model to be comprehensive, understandable, changeable, adaptable, and reusable (Hui, 1997).As shown in table 3-1, Wand and Woo (2002) previously mapped ontological constructs and premises into object constructs and premises.

Ontologically-Based Concept or Premise	Object Construct
Principles	
The world is made of things possessing properties*	Objects and their properties are the fundamental modeling constructs*
All things change and all changes are tied to things*	Encapsulation: state and behaviour are combined*
Things can combine to form composites*	Objects can form composite objects*
Things can affect each other's state evolution*	Objects interact*
Things can be categorized into classes defined by properties*	A class is a set of objects sharing a group of the same properties*
Concepts	
Thing*	Object*
Actor	Object with services
Non-actor	Object with no services
Actuator	Object with at least one joint state variable that it owns, and one service to modify it.
Propagator	An actuator object with at least one joint state variable that it owns shared with an Actor Object
Properties*	Not Modeled directly. See attributes*
Attribute functions*	Attribute*
Attribute representing inherent property*	Internal state variable*
Attribute representing mutual property*	Joint state variable*
State*	State (attribute values)*
Internal transformations*	Services*
Composite thing*	Composite objects*
Interaction	Communication (via requests)*
x acts-on y*	x and y have a shared state variable modifiable by x only*
Functional schema*	Definition of a class*
Event*	State change*
External event*	Request*
Internal event*	Action (execution of a service)*
Law	Service restriction
Process	System

Table 3-1 Mapping Ontological Constructs And Premises To Object-Oriented Constructs And Premises
***denotes original mapping (Wand and Woo 2002)**

As defined (Wart, Wand, & Woo, 1993) there are three general concepts of object orientation: classes and objects, association, and object communication.

3.3.1 Classes and Objects

A class is a collection of things that share common features. An object is an instance of a class. An object encapsulates its attributes and behaviours, that is, they are included with the definition of the object. For example, my pet trout is an instance of the class trout. We know that my pet trout has fins (property) and will swim (behaviour) from the definition of a trout

When an object is defined only the properties and behaviours relevant for our purposes is defined. Returning to our example, if we are hungry fishermen our definition of a trout (probably) does not include any mention of the light refraction index of the skin of a trout, but rather the fact it is an edible fish. Thus a definition of an object or class is not always perfect. This is the object-oriented concept of abstraction.

As mentioned above, objects have properties. These are referred to as attributes. Attributes only possessed by an object are internal attributes, attributes shared with another object are mutual attributes. When the values of attributes are measured the result is the state of the object. For example if our trout has fins, and the measure of the fins is broken, we can say the state of the trout is injured.

Services are the behaviour of objects. Services are what change attribute values. They have a well defined interface that is used to change attributes. When our trout friend swims the value of the attribute stomach contents will decrease and our trout will become hungry.

Encapsulation refers to storing the attributes and services (i.e. the state and behaviour), of an object together so that other objects do not need to worry about unnecessary information. This means that only the objects behaviour can access or change its state.

The only way one object can find information about another is to send a request to the object. If our trout wants to know if another (bigger) trout is hungry it can swim by its line of sight (sending a message, “Hey, I’m a smaller trout”). If the other trout tries to eat our trout in response, then our trout knows the bigger trout is hungry.

Instantiation is a particular occurrence of an object that can be distinguished from other occurrences of an object. My pet trout is an instance that is different from all the other trout objects that may exist, since its attribute denoting ownership has a value of “me”

Services and attributes are always inherited from the class to which an object is a member of. Hence all instances of trout will have the attributes and services of fish. However, different instances can have different values for their attributes. For example my pet trout may have speckled as its skin color, whereas your trout may have rainbow as its skin color.

3.3.2 Association

Objects can associate, that is, two or more objects can have a relationship. The most important relationships between objects are aggregation and classification.

An aggregation is a collection of component objects. The aggregation can be disaggregated into its component objects. Aggregations typically possess characteristics that are not present in the individual components themselves. A computer is an aggregation of a CPU, monitor, keyboard, etc. Combined they possess a new characteristic, processing power, that is not present in the components. A computer can then be disaggregated (disassembled) into its components.

Classification refers to being able to create generalizations and specializations to represent knowledge about classes. A generalization refers to identifying common properties of things to assist in the creation of abstractions. For example, if we notice my computer has a 17 inch monitor and your computer has a 15 inch monitor we may decide a computer class can be created with the attribute monitor size. A specialization is a more specific class (subclass) that inherits everything from its parent (superclass). Typically a subclass has attributes and services that the superclass does not. A laptop can be a subclass of the class computer. It has all the properties of a desktop, for example processing speed, and properties that a desktop does not have such as battery type.

3.3.3 Object Communication

Objects interact to request other objects perform services that the requesting object cannot perform. The services can be used to enforce constraints on the relationship. A student can request a professor to open a classroom. A professor will not open the classroom if it does not know that the student is a member of the professor's institution. This enforces the constraint that professors only open classrooms to registered students.

Objects can interact with those both inside and outside the system. External objects make requests to internal objects. Internal objects can make requests to both internal and external objects.

3.3.4 The Combination of OBPM and Object-Orientation

Object-orientation, obviously, centers around the object. OBPM centers around the agent. According to the BWWP ontology an agent is a thing that possesses properties and undergoes change via operations. An object possesses attributes and performs services. A service is how an object changes. The combined construct of an OBPM agent and an object will henceforth be referred to as an object-oriented activity-based process modeling agent (or just agent for short). The aforementioned links are still ambiguous and unclear. Figure 3-1 is an illustration of what we now call an agent.

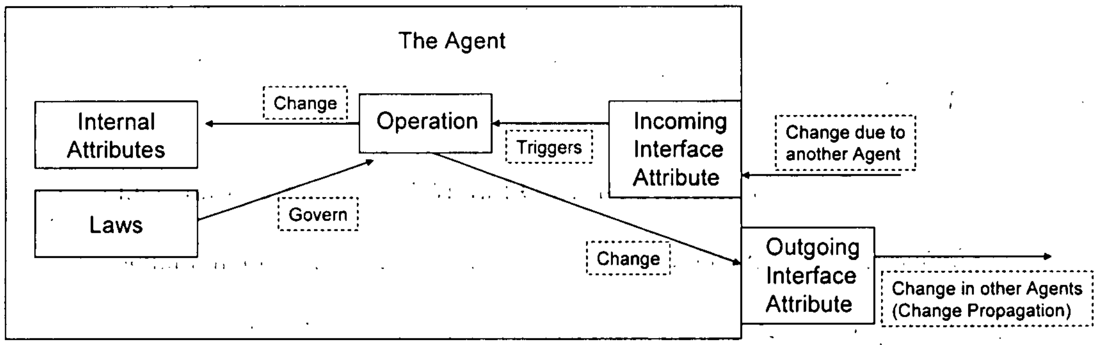

The Agent
Change
Internal Attributes
Operation
Triggers
Incoming Interface Attribute
Change due to another Agent
Laws
Govern
Change
Outgoing Interface Attribute
Change in other Agents (Change Propagation)

Figure 3-1 ABPM Agent

Here is a brief definition of Figure 3-1. An agent "communicates" with another agent via changes. This "communication" can take on two forms. First an agent can have an interface attribute changed by another agent. Second an operation of an agent may change an interface attribute of another agent. A change to an interface attribute may trigger an operation. An operation's triggering or output may be governed by laws. An operation changes either an internal attribute or an outgoing interface attribute. Each of the constructs in figure 3-1 is discussed below.

3.3.4.1 Attributes

Recall that in BWWP attributes are used to represent the properties of a thing, which in turn represent the state of the thing. In ABPM attributes can be used to represent the state of the Agent. In figure 3-1 there are two main kinds of attributes; interface attributes and internal attributes. Internal attributes are not known to other agents and can only be accessed or changed via the services of the agent. This demonstrates object-oriented principles of both encapsulation and object independence. Interface attributes can be accessed (changed) by other objects.

Interface attributes model mutual properties of things. Agents interact with each other via changing the value of mutual properties. Therefore all interface attributes have two agents associated with them; an agent doing the changing (an outgoing interface attribute), and the agent being changed (an incoming interface attribute).

Internal attributes are solely possessed by an Agent and are unknown to other agents. An internal attribute is what is changed by an internal event in an agent.

3.3.4.2 Operations and Change Propagation

Changes result from interactions between agents. There are two types of changes possible; a change in an agent caused by another agent and a change in an agent caused by itself. For our purposes a change can be defined as altering the value of an attribute. An agent causes change in another agent by changing the value of an outgoing interface attribute it owns which is associated with the incoming interface attribute of another agent. An agent causes change in itself by altering the value of an internal attribute.

According to the object oriented literature a service is how an object does anything. Hence for an object to affect itself or another object it needs a service. That said, in OBPM agents affect themselves or each other via changes. Changes are carried out as operations. Hence a change is implemented through an operation in ABPM.

When an agent has an interface attribute changed resulting in instability the agent performs one or more activities to become stable. As part of an activity the agent may change one or more agents causing them to become unstable. These agents may then change others, and so on. This is known as change propagation.

3.3.4.3 Activities

An activity is what happens in an agent from the time when it becomes unstable to the time when it is stable. All agents are initially stable. Instability is caused by an incoming interface attribute being changed. When an activity occurs operations occur until the agent is stable. The operations will change some combination of internal and outgoing interface attributes. Thus an activity is made of operations that use or modify attributes.

3.3.4.4 Laws

According to BWWP laws are properties of a thing. They restrict how a thing can change. In the case of ABPM the restriction is on what operations can occur in an agent.

Recall from chapter 2 in the discussion on the BWW ontology, "The set of values for the attributes of a thing comprise the *state* of the thing. A *conceivable state space* for a thing is the set of all possible states a thing may ever assume. *State laws* serve to restrict the values of the properties of a thing to a subset of the conceivable state space. State laws must enforce a restriction due to either natural or human laws. A law is a property. For example, most bank accounts have the restriction (state law) that the balance must be greater than or equal to zero. This is due to the human law that people are only allowed to spend up to the total amount of money that is in their bank account. The *lawful state space* of a thing is the set of states that exist for a thing that comply with its state laws. A lawful state space is usually a subset of the conceivable state space.

A transformation is a state (attribute) change from one state to another state. A *lawful transformation* defines the events that are lawful for a thing. The *lawful event space* is usually a subset of the event space, and defines those events in a thing that are lawful."

For ABPM, a state law still enforces constraints. They dictate the constraints on what the output from an operation is, the output of an operation is an alteration of the value of a property. Thus they restrict the values a property can be altered to and by extension they restrict the lawful state space.

For ABPM transformation laws enforce constraints on which changes could occur. They restrict the possible set of transformations (i.e. operations, since operations are how the transformations are carried out) to a set that are deemed lawful in an agent.

In short, attributes and operations represent the properties and behaviour of agents, while the state and transformation laws are the constraints on properties (attribute values) and behaviour respectively.

3.3.5 Mapping Summary

Table 3-2 summarizes how the OBPM constructs will be mapped to Object constructs to create Object-Oriented OBPM Constructs.

3.4 Meta-model of ABPM

Figure 3-2 presents the ABPM meta-model. The metamodel can be used to show the relationships between the constructs, as discussed above, in a condensed manner. The meta-model shows the ABPM constructs as rectangles. The relationships between constructs are shown using arrows. The cardinality numbers indicate the requirements on the relationship. The inverted Ψ show that a construct exists in both the generalized and specialized role.

OBPM Construct	Object Construct	ABPM Construct
Agent	Object	Agent
Actor	Specialization of an Object	Agent
Resource	Specialization of an Object	Resource (An Agent with no services)
Properties	Attribute	Attribute
Intrinsic Property	Internal Attribute	Internal Attribute
Mutual Property	Shared Attribute	Incoming Interface Attribute + Outgoing Interface Attribute
Composite thing	Composite object (aggregation)	Composite Agent(aggregation)
State	Attribute values	Attribute values
Agent in the domain of interest	Internal object	Internal Agent
Agent outside the domain of interest	External object	External Agent
Operation	Service	Operation
Activity	The events that occur when an Object becomes unstable	The events that occur when an agent becomes unstable
Transformation law	Service restriction	Transformation law
State law	Service restriction	State law
Event	A trigger of a service	A trigger of an operation
Internal event	An event inside the object causing a transformation in the object	Change of an internal attribute, performed by an operation
External event	An event outside the object causing a transformation in the object .	Change of an incoming interface attribute
X affects Y	X changes the shared attribute of X and Y	X changes its outgoing interface attribute which is tied to the incoming interface attribute of Y
Stability	No services required by an object	No operations required by an Agent
Inherited Attributes	Inheritance	Super and Sub agents
Process	Propagation	Process

Table 3-2 Mapping Summary

The meta-model is explained as follows: an agent is either internal or external. Internal agents perform activities. An internal agent performs one or more activities, but the activity is performed by only one agent.

Activities consist of operations and attributes. An activity must consist of one or more operations; however the operations occur in only one activity. An activity involves changes to two or more attributes; however the attributes are affected in only one activity.

An operation changes one or more attributes, the attributes can only be changed by one operation.

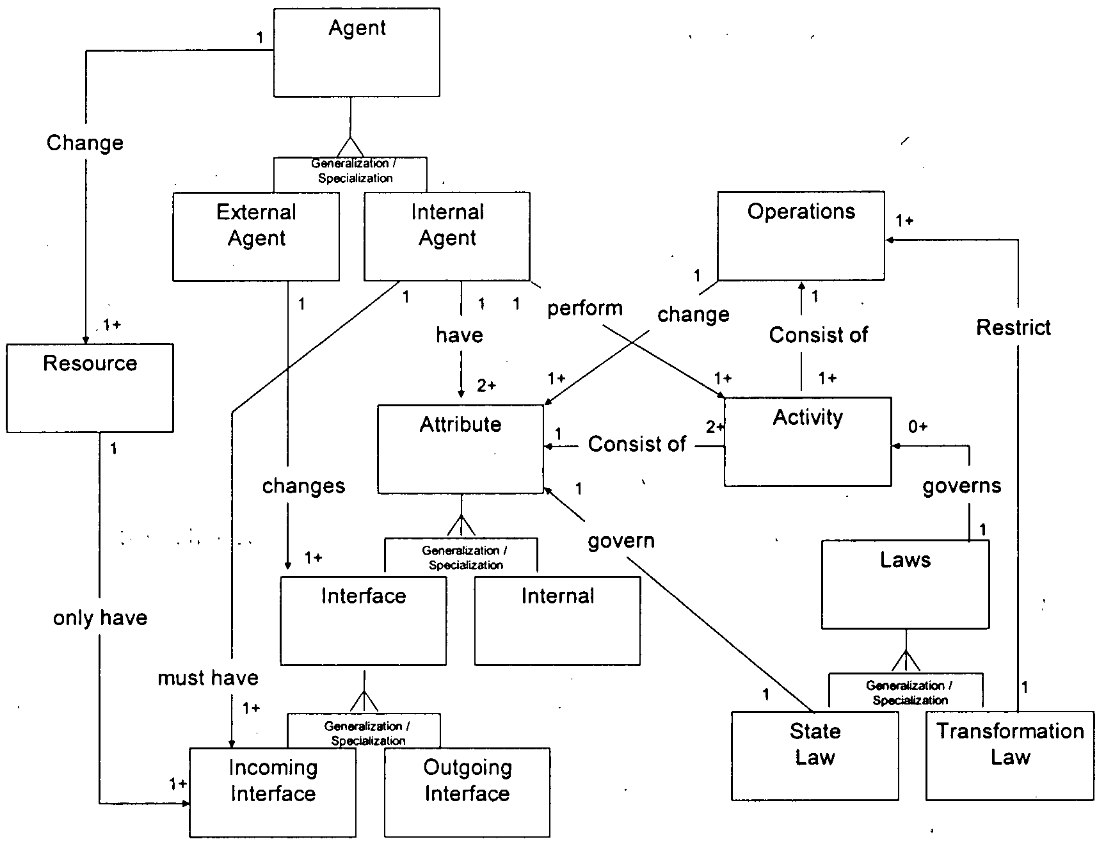


Figure 3-2 ABPM Meta-Model

An activity may be governed by zero or more laws, the laws govern only one activity. A law is either a state law or a transformation law. State laws govern attributes, one state law governs one attribute. A transformation law governs one or more operations, the operations are governed by only one transformation law.

Attributes are either internal attributes or interface attributes. Interface attributes can further be broken up into incoming interface attributes and outgoing interface attributes.

Internal agents have two or more attributes, the attributes are only possessed by one agent. Internal agents have one or more incoming interface attributes, the incoming interface attribute(s) is(are) only possessed by one agent. Internal agents are required to have at least one incoming interface attribute since something external to the agent is what initiates change in the agent. The requirement of two or more attributes refers to an internal agent needing to both be changed (i.e. have one incoming interface attribute), and then change something else (in itself or another agent). Hence at least two attributes, one incoming interface + at least one other attribute it changes in response to being changed. If an internal agent were to only have incoming interface attributes then it would be a resource.

An external agent has one or more interface attributes that are changed, the interface attributes are possessed by only one agent. Resource are changed by agents. An agent can change one or more resources, but the changes are performed by only one agent. A resource only has incoming interface attributes. A resource can have one or more incoming interface attributes, but the incoming interface attributes are possessed by only one resource.

3.5 Graphical Representation

We are now ready to introduce our graphical constructs for ABPM. The following subsections will first show the construct then follow with an explanation of the construct. The graphical constructs can be used to assist the user in following the ABPM modeling process (as presented in section 3.6 below).

3.5.1 Domain Representation

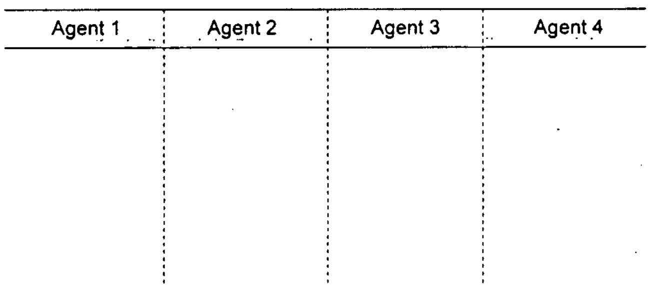


Figure 3-3 Domain Representation

If we are trying to show the agents that interact in a process, we need to consider the domain they will interacting in, i.e. the scope of the process. The domain in which they interact will be represented using swimlanes, with the swimlanes themselves being the boundary of an agent.

3.5.2 External Agent

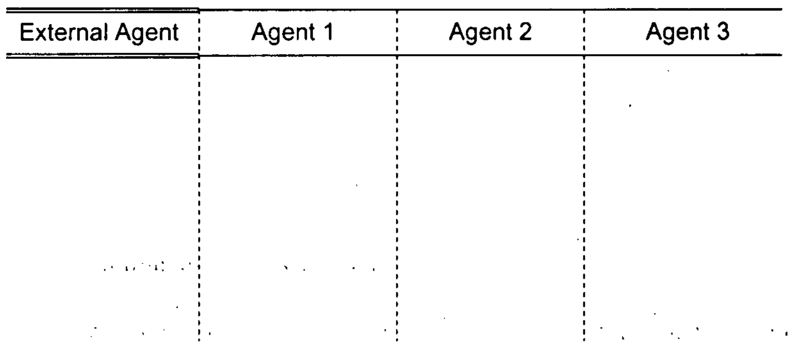


Figure 3-4 External Agent Representation

The name of an external agent is doublelined. External agents are what cause the initial event(s) that occur outside the domain of a process. They perform activities that change (affect) a property of a thing in the domain. An external agent may also have one or more attribute changed by an agent in the system. The only information we need to represent in the swimlane of an external agent are the incoming interface attributes from the system and the outgoing interface attributes to the system.

Incoming interface attributes follow the notation:

agent that changed the attribute::attribute changed

Incoming interface attributes are how an agent receives a change from another agent.

Outgoing interface attributes follow the notation:

attribute changed::agent changed

Outgoing interface attributes are how an agent initiates change in another agent.

3.5.3 Internal Agent

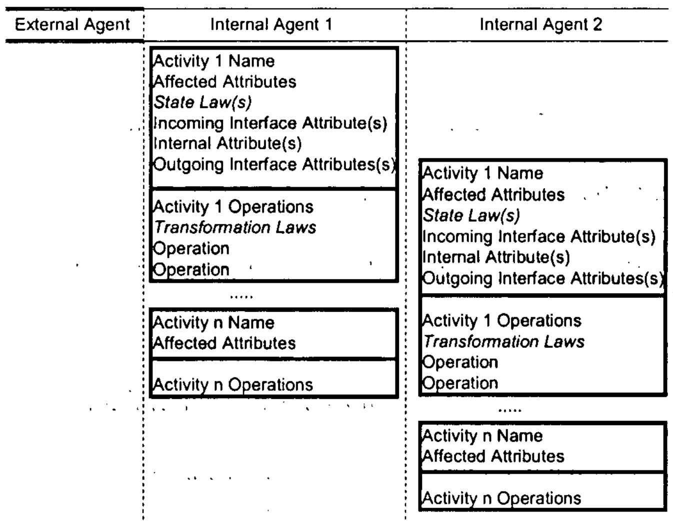


Figure 3-5 Internal Agent Representation

Since external agents outside the domain affect agents in the domain we need internal agents that represent the agents within the domain. The name of an internal agent is singly-lined.

An activity consists of attributes, operations and laws. In particular, an activity is the attributes that are changed and the operations that change them in response to the agent receiving a change from another agent with the laws that dictate contraints on how they are changed. An agent can receive an infinite number of changes from other agents hence there could be an infinite number of activities.

In keeping with the concept of encapsulation in the representation of an activity we keep the attributes that are changed and the operations that change or use them together. Hence we outline each activity to illustrate the encapsulation.

The first information included in the affected attributes of an activity is the state laws on the attributes. A state law dictates if there are any restrictions on the values an attribute can take. This is a freeform box of text before the incoming interface attributes of an activity.

Incoming interface attributes follow the notation:

agent that changed the attribute::attribute changed

Incoming interface attributes are how an agent receives a change from another agent.

Internal attributes are changed solely by the internal agent. They are changed via operations.

Outgoing interface attributes follow the notation:

attribute changed::agent changed

Outgoing interface attributes are how an agent initiates change in another agent. They are changed via operations, which will also change the associated incoming interface attribute of another agent.

When we look at the operations for an activity the first thing that is represented is the transformation laws. Transformation laws dictate which behaviours (operations) occur under what conditions. A transformation law is represented by a freeform box of text before the operations of an activity.

An operation is what changes an attribute. An operation may change several attributes but must at least change one attribute. Hence for every attribute that changes there must be an operation that changes it. The attributes that an agent changes are its internal attributes and its outgoing interface attributes. Thus for every internal attribute and every outgoing interface attribute there is an operation that changes it.

The temporal sequence of attributes and operations should be kept the same. If the laws and incoming interface attributes are removed from an activity, an attribute in position X of the affected attributes list, should have the operation that changes it in position X of the operations list. This is in keeping with the general notion of a process.

We also need to consider how do we number activities? Our solution to this is that we number them according to how many activities there are in an agent. So if an agent performs nine activities, they are numbered one through nine. Thus different agents will have their own activity one. Typically process models will give each activity its own number thus there will only be one activity number one, etc.

3.5.4 Resources

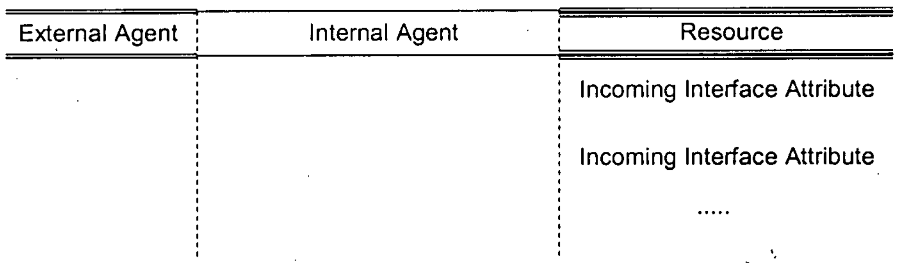


Figure 3-6 Resource Representation

Resources are those things that only are changed by other things. A resource is indicated by a treble line around the name. When they are changed they go from one stable state to another stable state. The have no internal transformations. Hence they have no operations. Thus a resource will only have attributes of the notation:

agent that changed the attribute::attribute changed

3.5.5 Agents Sharing A Mutual Attribute

The arrow in figure 3-7 designates that the agents share an interface attribute. The agent at the tail of the arrow changes the outgoing interface attribute it has while the agent at the head of the arrow is the receiver of the change via its incoming interface attribute. A block arrow designates that an external agent changes some thing in the system.

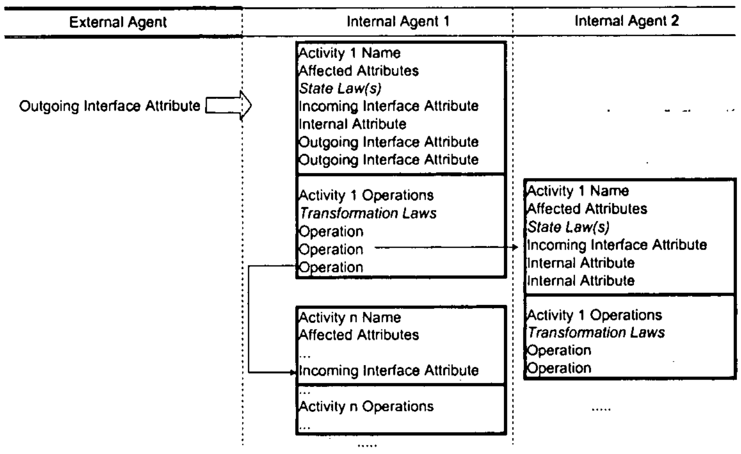


Figure 3-7 Agents Sharing A Mutual Attribute Representation

Recall, we are creating models at the class level. When an arrow exists between two activities in the same agent it shows one instance of agent can change another instance of the same agent.

In the event an agent can change the exact same incoming interface attribute of another agent from different activities, there are be multiple outgoing interface attributes but only one incoming interface attribute that is changed. In this case the arrows are merged into one doubled arrow as illustrated in figure 3-8.

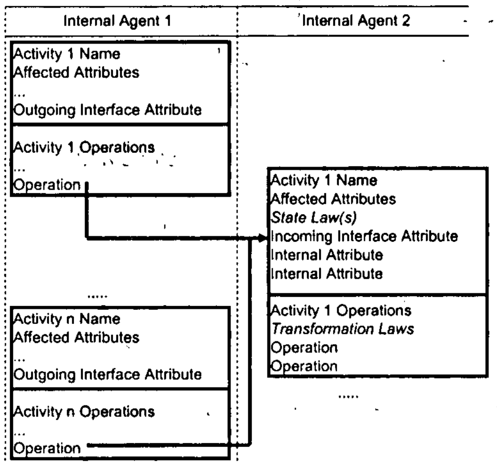


Figure 3-8 Different Operations changing one incoming interface attribute

3.5.6 Composite agents

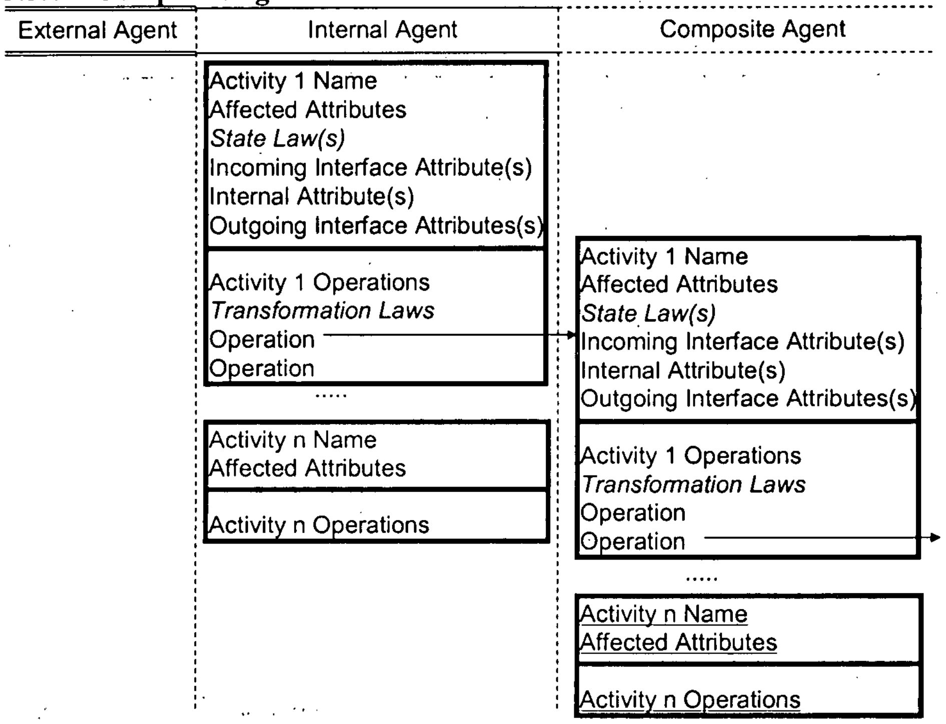


Figure 3-9 Composite Agent Representation

Composite agents in their composite view appear the same as any other agent except the lines around their name are dashed. If an activity in a composite agent is underlined the activity is an emergent activity that would not exist without the aggregation of components. Activities that are not underlined are those from component agents.

The composite agent view is used since we may not be interested in all the details of a composition. In those situations where we need to know the details of a composite agent a composite agent can be decomposed down into its component agents.

When a composite agent is created from component agents it must possess emergent attributes, i.e.its own attributes that are not part of any component agent, thus they need their own operations as well. The emergent attributes and operations would not exist unless the agents are aggregated. Figure 3-10, illustrates how a composite agent can be added to a diagram without having to rearrange the entire diagram. The composite agent lists its component agents, while the components list that they are part of a composition.

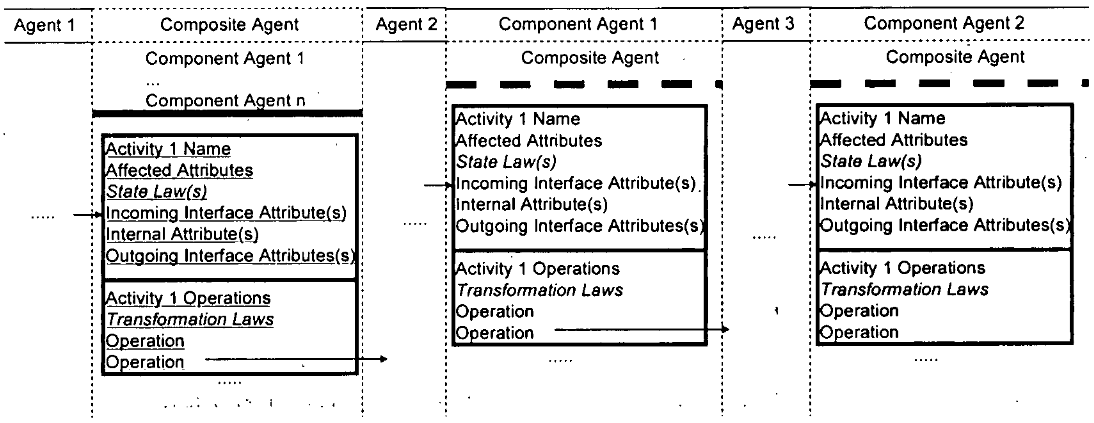


Figure 3-10 Adding A Composite Agent

The decomposition of a composite agent allows us to see how the agents that make up the composite agent interact, and what attributes and operations emerge from the composition in one figure. Figure 3-11 is an example of the decomposition for a composite agent. Ideally we would include a diagram like figure 3-11 and not figure 3-10 when interested in the composition of a composite agent. However when it is not feasible to rearrange the diagram to have all the component agents side by side the arrangement in figure 3-10 can be used.

The decomposition as shown in figure 3-11 can also be used to verify the integrity of a composite agent. By only including the composite and its components we can verify that all incoming changes and outgoing changes are received and generated (respectively) by the composition or its components.

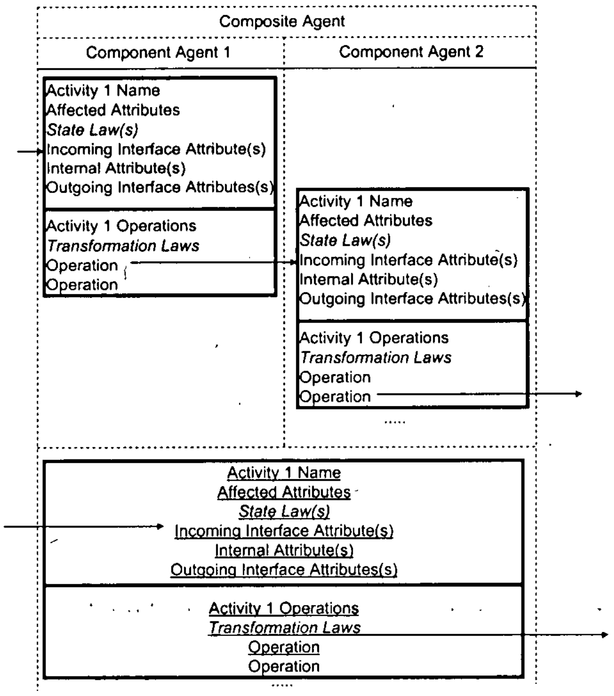


Figure 3-11 Decomposition Of A Composite Agent

When decomposing a composite agent into its components the following must specified:

- Which incoming changes go to which component agent?
- Which incoming changes go to the composite agent (are emergent)?
- Which outgoing changes are generated by which component agent?
- Which outgoing changes are generated by the composite agent (are emergent)?
- Which changes do the components use to interact with each other? These changes are not present in the composite view of the agent.
- Which internal attributes belong to the composite and the components?
- What laws exist on the incoming and outgoing changes of the components?

When the attributes are reassigned to either the composite or component the operation that is associated with that attribute is reassigned as well. Component agents may require new attributes and services not present in the composite agent to model the interaction of the component agents.

3.5.7 Superagents And Subagents

As shown in figure 3-9, the agent on top is the superagent(i.e. the generalization), the agents on bottom are subagents(i.e. the specialization). Attributes and operations in the superagent are inherited by the subagent. Inherited attributes and operations are not shown in the subagent. Subagents have attributes and operations possessed only by that subagent. A subagent's inherited attributes and operations can never be different from the attributes and operations of the superagent (from the definition of a subclass), hence we do not need to show them. Figure 3-13 shows how to represent superagents and subagents when it is not feasible to rearrange the diagram to have them side by side.

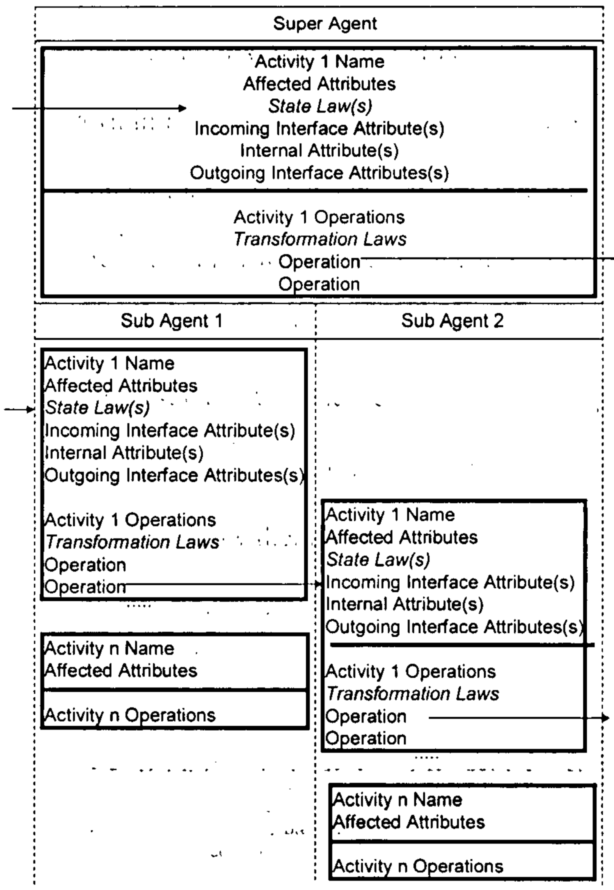


Figure 3-12 Superagent and Subagent Representation

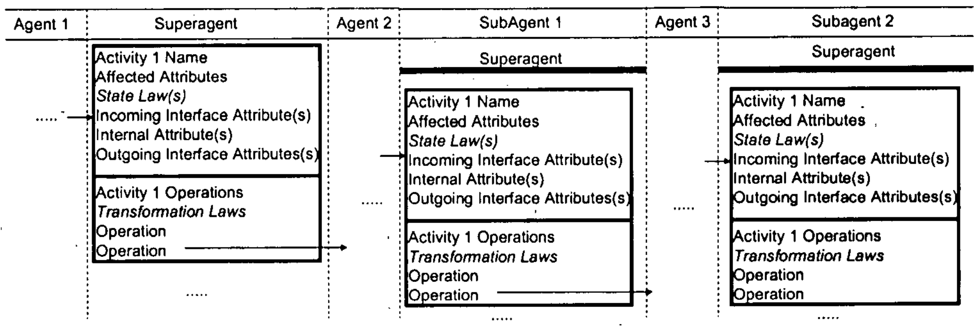


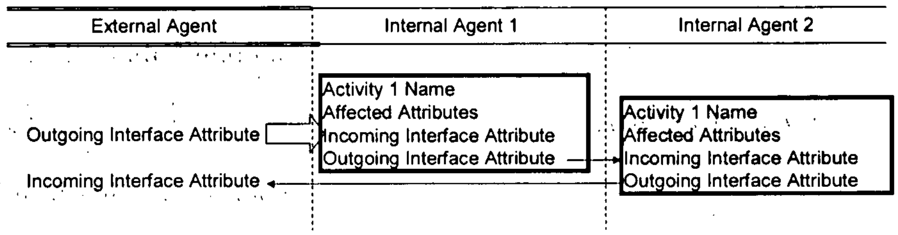
Figure3-13 Creating A Superagent

3.5.8 An Alternate Notation

There are two situations where it is beneficial to have an ABPM with fewer details.

1. The details are not known
2. The diagrams are large and unwieldy

We can handle both situations using Agent Templates (AT). An AT table can be used to store some or even all the pertinent details about the agent and an agent with less or even no details appears in the ABPM. Since we are focusing on change propagation the two suggested compressions are either retaining only the interface attributes that are changed in an activity (as per figure 3-14), or compressing an activity totally down to its name(as per figure 3-15).



External Agent
Internal Agent 1
Internal Agent 2
Outgoing Interface Attribute
Incoming Interface Attribute
Activity 1 Name
Affected Attributes
Incoming Interface Attribute
Outgoing Interface Attribute
Activity 1 Name
Affected Attributes
Incoming Interface Attribute
Outgoing Interface Attribute

Figure 3-14 Example Compressing Down To Agent Interactions

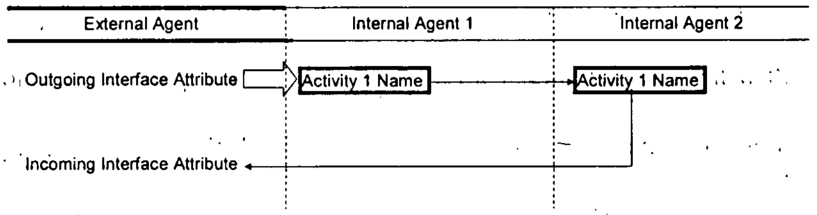


Figure 3-15 Example Compressing Down To Activity Name

The internal agent template is presented in Figure 3-16. It will be filled in using the notation developed above.

Agent					
	Attributes			Operations	
Activity	State Law	Interface Attributes	Internal Attributes	Transformation Law	Operation
1					
2					
…					
n					

Figure 3-16 Internal Agent Template

The Resource template is presented in figure 3-17. Since it only has incoming interface attributes all entries will be in the form:

agent that changed the attribute::attribute changed

Resource
Incoming Changes

Figure 3-17 Resource Template

With external agents less detail is needed than with internal agents. The only information of interest is the incoming and outgoing interface attributes that are altered. The external agent template is presented in figure 3-18. Incoming interface attributes will follow the form:

agent that changed the attribute::attribute changed

Outgoing interface attributes will follow the form:

attribute changed::agent being changed

Agent	
Incoming Changes	Outgoing Changes

Figure 3-18 External Agent Template

In a composite agent the agent template has extra rows. The extra rows indicate which agents are components of the composite agent. The component agents are the same as any internal agent and thus have the same agent template as an internal agent. *The composite agent template has two views (using the same template). The first view is just of the activities of the composite agent (i.e. the emergent attributes and operations). The second view is the 'composited' view*, that is, all the activities of the components that are included in the composite and its emergent activities. The emergent attributes and operations would be italicized in the composite view to distinguish them from component attributes and services (if the agent has been decomposed). Figure 3-19 shows the composite agent template.

Agent					
Component Agent 1:					
Component Agent 2:					
...					
Component Agent n					
	Attributes			Operations	
Activity	State Law	Interface Attributes	Internal Attributes	Transformation Law	Operation
1					
2					
...					
n					

Figure 3-19 Composite Agent Template

The agent template for superagents is not different from the internal agent template. The agent template for subagents has an extra row to indicate the superagent from which it is derived. The attributes and operations that are shown are those that are unique to the subagent. The subagent template is shown in figure 3-20.

Agent					
Superagent:					
	Attributes			Operations	
Activity	State Law	Interface Attributes	Internal Attributes	Transformation Law	Operation
1					
2					
...					
n					

Figure 3-20 Subagent Agent Activity Template

3.5.9 Allowed Interactions

This section summarizes how the constructs are allowed to interact .

External agents: Initiate change in an agent in the system. Receive a change from an agent in the system. Refer to figure 3-21

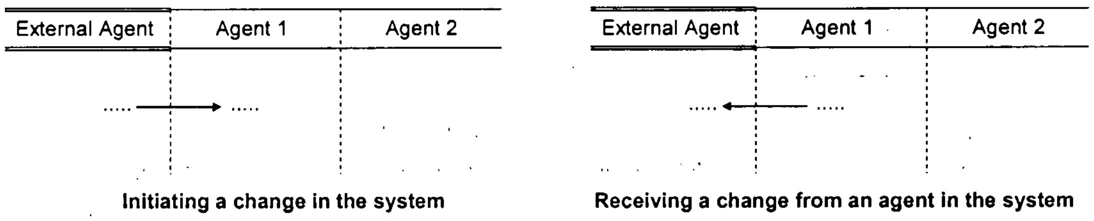

External Agent
Agent 1
Agent 2
Initiating a change in the system
External Agent
Agent 1
Agent 2
Receiving a change from an agent in the system

Figure 3-21 Allowed Interactions For An External Agent

Resources: Receive a change from an agent in the system. Refer to figure 3-22

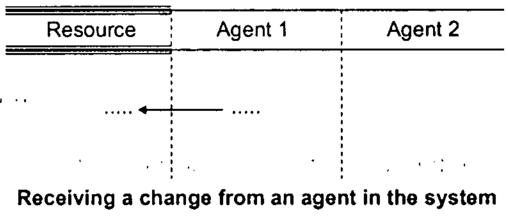
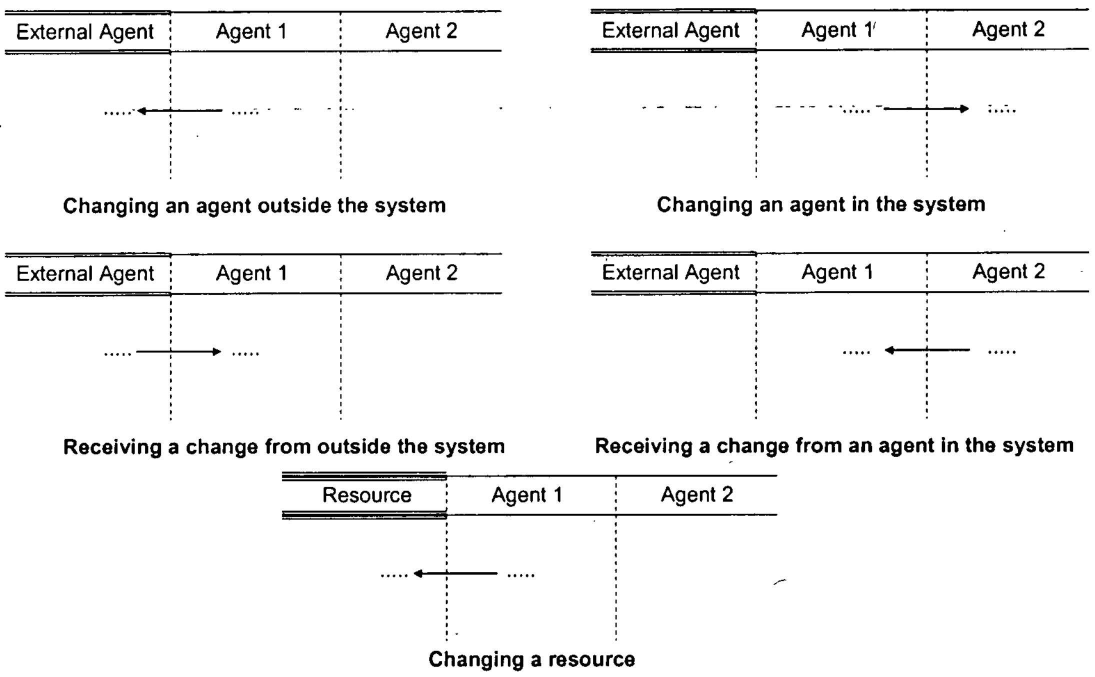


Figure 3-22 Allowed Interactions For A Resource

Internal agent: Initiate change in both internal and external agents. Receive a change from internal and external agents. Change a resource. Refer to figure 3-23


External Agent
Agent 1
Agent 2
Changing an agent outside the system
External Agent
Agent 1
Agent 2
Changing an agent in the system
External Agent
Agent 1
Agent 2
Receiving a change from outside the system
External Agent
Agent 1
Agent 2
Receiving a change from an agent in the system
Resource
Agent 1
Agent 2
Changing a resource
Figure 3-22 Allowed Interactions For An Internal Agent

3.6 ABPM Modeling Process

The main concept to keep in mind when generating an ABPM is change propagation. That is, when one agent is changed, this change will cause the agent to change other agents, which will lead to other agents being changed, and so on. Eventually, there are no more agents that are changed and the process ends when all the agents that are changed become stable (stop changing).

Frequently process-modeling languages lack a well-defined method for generating a process model (Huckvale and Ould, 1994, Wand and Woo, 1999, Wang 2002). One of the goals behind the development of OBPM was to eliminate problems with existing process modeling languages. Hence we must provide a clear and unambiguous method to create an ABPM. Thus we will present an algorithm for the graphic representation of ABPMs. This algorithm is based upon a set of modeling rules.

3.6.1 Modeling rules

At this point we need to introduce a set of rules (Wand and Woo, 1999) and assumptions to address the following points:

1. What is the scope of the model?
2. What agents should be included in the model?
3. What resources should be included in the model?
4. What agent properties and operations should be included in the model?
5. When to include composites agents?
6. When to subclassify agents?
7. When to begin a new activity?

Due to the fact that they are developed specifically for object-oriented models that are ontologically based (albeit for enterprise modeling) we will use and adapt the rules developed elsewhere (Wand and Woo 1999) to answer the questions. Only when the rule is changed from its original intent will we define its development.

Rule #1: The scope identification rule

This rule is used to define what should be included in the process model of the system. It is based upon the belief that anything that happens in the system is in response to something happening outside the system. That is, something outside the system affects something in the system causing the system to become unstable. This is the only way a system can become unstable. Once a modeler decides what events happen which are external to the system, all direct and indirect actions due to the external event are in the scope of the system. Thus the rule reads as follows:

Rule 1 (scope): The aspects of the system to be modeled are all and only those needed to represent the effects of the relevant external events[5].

Rule #2: The affected thing identification rule

An event is a change. From our ontological foundation only things change. To be included a thing has to either be changing another thing or changed by another thing. Thus the rule reads as follows:

Rule 2a (agent identification): The agents included should be those that are either generating changes in the system, or are responding directly and indirectly to the external changes to the system.

Rule 2b (Resource identification): The resources included should be those that are changed by agents in the system.

The agent identification rule highlights two kinds of agent. External agents are outside the system that either change something in the system or are changed by something in the system. Internal agents are the agents that make up the system. An internal agent must be changed by at least one other agent. The resource identification rule indicates that only the resources used by internal agents should be included

Rule #3: The operation inclusion rule

Based on our mapping operations represent transformations that happen to an agent. Operations will be invoked by an unstable agent to change an attribute in an attempt to become stable during an activity. An agent becomes unstable when the prerequisite interface attributes (i.e. incoming interface attributes) of an activity have been changed.

Rule 3a (operation inclusion): An operation will be included in an activity if it is invoked as a result of an agent attempting to become stable.

Rule 3b (transformation law inclusion): A transformation law will be included if it affects what operations occur.

Rule #4: The attribute inclusion rule

Since all activity in a system is initiated due to an external change, only those attributes that are part of the activities due to the external change should be included. The attributes that are part of an activity are those that either initiate the activity or are changed by operations during an activity. The attributes must be used or modified by an operation.

Rule 4a (interface attribute inclusion): an interface attribute will be included if it is known by and shared between two things. An outgoing interface attribute can only be possessed by an agent, whereas an incoming interface attribute can be possessed by an agent or a resource.

Rule 4b (Internal attribute inclusion): an internal attribute will be included only for those operations that do not act upon an interface attribute as defined from 4a. An internal attribute is internal to an agent.

Rule 4c (state law inclusion): a state law will only be included if it restricts the values an attribute can be changed to.

[5] This is the original Wand and Woo (1999) definition of the rule.

Rule #5: The attribute ownership rule

Since properties belong to things, every attribute is owned by only one agent.

> Rule 5 (attribute ownership): For every attribute in the model there is exactly one agent that can modify it. For an internal attribute, the agent is the only one that can access the value of the attribute. For interface attributes the agent that possesses the outgoing interface attribute is the agent that modifies both the outgoing interface attribute and the incoming interface attribute (even though the incoming interface attribute is owned by another agent).

Rule #6: The composite agent rule

In some cases agents may need to be functioning together to respond to the changes resulting from the external stimulus since neither may be able to respond on its own. This creates emergent behaviour not present in either agent. Recall from ontology, that composite things have emergent behaviour.

> Rule 6 (Composite agents): A composite agent may be created only if it possesses emergent attributes not present in any of its components. A composite agent possesses all the attributes and operations of its components.

Rule #7: The sub-classification rule

In some cases agents may have properties that are very similar. It may be beneficial to create a superclass to simplify a model.

> Rule 7 (Sub-classification): A sub-class should be created only when it has properties not present in the superclass. A sub-class inherits all properties of the

superclass. In order to form a superclass two or more agents need to have some properties in common.

Rule# 8: The new activity rule

The first change in an agent is always the beginning of a new activity. An activity is a sequence of events and changes. Typically an actor becomes unstable, undergoes transformations, then becomes stable. Later on, other changes may cause the agent to yet again become unstable and the process repeats itself. Resources never become unstable, they are stable, changed by another agent, then are stable.

> Rule 8 (New activity) When an agent becomes unstable after being stable it is the beginning of a new activity. When an activity ends an agent is in a stable state.

3.6.2 Modeling Process

If we follow the rules in a systematic manner we can produce an object-oriented activity-based process model. Further, there will be no need to check the integrity of the model since by following the algorithm correctly we ensure the model is correct. The guidance in applying the rules comes from the original OBPM algorithm. That is, we identify the changes generated external to the system, identify what is affected by the changes, and then analyze what has been affected. We can use this to follow changes as they propagate through the system. The following algorithm for creating ABPMs is designed to center around change propagation.

```
1. Identify the external agents.
2. For each external agent identify the changes generated.
3. For each change:
   3.1. Identify the agent or resource that was changed
   3.2. If a resource was changed identify the incoming interface attribute
   3.3. If the agent is an internal agent:
      3.3.1 If this is the first change to an agent, or the last activity of an agent has
            gone through a sequence of instability-change-stability create a new
            activity
      3.3.2 Identify the incoming interface attributes that were modified
      3.3.3 Identify any state laws that may restrict change
      3.3.4 Identify any transformation laws that may exist for the incoming interface
            attributes
      3.3.5 If an agent becomes unstable:
         3.3.5.1 Identify the operations that may occur
         3.3.5.2 Identify any transformation laws that may affect what operations occur
         3.3.5.3 Identify the internal attributes that will be affected
         3.3.5.4 Identify the outgoing interface attributes that were modified
         3.3.5.5 Repeat steps 3.3.5.1 to 3.3.5.4 until the agent becomes stable
      3.3.6 Repeat step 3 for each outgoing interface attribute of an agent that was
            changed in step 3.3.5.4
4  If needed identify super and subagents using the internal agents.
5  If needed identify composite and component agents using the internal agents.
```

Table 3-3 shows the relationship between the rules, the ABPM algorithm, and the original OBPM algorithm. From this table we can see how the proposed ABPM algorithm encompasses and expands upon the original OBPM algorithm. We can also see that by following the ABPM algorithm we force the modeling rules developed above in section 3.6.1 to be followed

Step	Rule	Purpose	OBPM algorithm
1. Identify the external agents	2a	Agent Identification	
2. For each external agent identify the changes generated.	1	Scope Identification	Main
3. For each change: 3.1 Identify the agent or resource that was changed	2a and 2b	Agent and Resource Identification	Main
3.2 If a resource was changed identify the incoming interface attribute	4a	Interface Attribute Identification	
3.3 If the agent is an internal agent: 3.3.1 If this is the first change to an agent, or the last activity of an agent has gone through a sequence of instability-change-stability create a new activity	8	Activity Identification	Affected thing
3.3.2 Identify the incoming interface attributes that were modified	4a	Interface Attribute Inclusion	
3.3.3 Identify any state laws that may restrict change	4c	State Law Inclusion	
3.3.4 Identify any transformation laws that may exist for the incoming interface attributes	3b	Transformation Law Inclusion	
3.3.5 If an agent becomes unstable: 3.3.5.1 Identify the operations may occur	3a	Operation Inclusion	Decompose
3.3.5.2 Identify any transformation laws that may affect what operations occur	3b	Transformation Law Inclusion	
3.3.5.3 Identify the internal attributes that will be affected	4b	Internal Attribute Inclusion	
3.3.5.4 Identify the outgoing interface attributes that were modified	4a	Interface Attribute Inclusion	Affected thing
3.3.5.5 Repeat steps 3.3.5.1 to 3.3.5.4 until the agent becomes stable	3	Operation Inclusion	Decompose
3.3.6 Repeat step 3 for each outgoing interface attribute of an agent that was changed in step 3.3.5.4	5	Attribute Ownership	Decompose
4. If needed identify super and subagents using the internal agents.	7	Superagents and Subagents	
5. If needed identify composite and component agents using the internal agents.	6	Composite and Component Agents	

Table 3-3 Relating The Rules To The ABPM And OBPM Algorithms

3.6.3 Model Integrity

Once a model has been constructed it may be necessary to check the model if the model is semantically correct. Although the above modeling algorithm is supposed to ensure semantic correctness there may be other ways to generate an ABPM (such as from an OOEM, see the next chapter) that do not necessarily guarantee the model will be semantically correct. We can check using a set of model integrity rules. The model integrity rules reflect the modeling rules (Wand and Woo, 1999).

1. Every change in the system should be able to be traced back to an initial change from an external agent.
2. Every agent must have at least one activity.
3 Every Activity must have at least one operation
4. Every resource must only have incoming interface attributes.
5. Every attribute is changed by only one operation.
6. Every activity can only have one incoming interface attribute unless governed by a transformation law
7. Every outgoing interface attribute must have a corresponding interface attribute and vice versa.
8. Every composite agent must possess emergent attributes and operations not present in the component agents
9. Every subagent must possess attributes and operations that are unique to the subagent and are not inherited from the superagent.

3.7 An Example

We recognize that this chapter has presented a fair bit of new ideas and concepts. To illustrate them, we will use the following example

The ACME Warehouse Management Inc. Case[6]

ACME Warehouse Management Inc. offers storage facilities and redistribution services (between their different warehouses) across the nation. A customer can request space in a particular warehouse, request items to be transferred to another warehouse, or request withdrawal of items from a particular warehouse (even for items not stored there).

For the purpose of this case, we only look at the activities involved in processing a withdrawal request. A customer contacts ACME headquarters to request a withdrawal. An office clerk checks whether the customer has the authority to withdraw the items. The clerk then passes the withdrawal request to the warehouse where the customer wants to pick up the items.

If the warehouse does not have the items or does not have enough quantity of the items, the warehouse manager will contact other warehouses for the requested items. If the items are located the warehouse manager will ask the planner to arrange for transportation for the requested items.

[6] Based on a case in I. Jacobson, Object-Oriented Software Engineering, Addison-Wesley, 1992

The planner's responsibility is to schedule the company's truck fleet to accommodate requests for transportation, taking into account the existing schedule of each truck and its capacity. The warehouse manager will be notified whether the transportation request can or cannot be satisfied.

The warehouse manager will notify the office clerk if the request can be fulfilled or not, and the reason. The office clerk will notify the customer as to the status of the request (approved, or declined due to lack of authority, no inventory, or no transportation).

The planner issues transport orders to truck drivers. After receiving a transport order, the truck driver informs the warehouse about the pickup of the items. The warehouse manager will make arrangements to have the items ready when the truck arrives. When the truck arrives at the warehouse the items are loaded. The truck driver then informs the next warehouse about the delivery. When the truck has arrived at the next warehouse, the items are unloaded. A warehouse worker finds space for the items and arranges to have them moved to the allocated space. The worker updates the warehouse's inventory information. Truck drivers are required to report the status of the truck and the delivery to the planner after each step.

The customer will come to the warehouse on the required date to pick up the items. A warehouse employee will check all the necessary documents and will deliver the items with an accompanying documentation to the customer.

Supplemental description

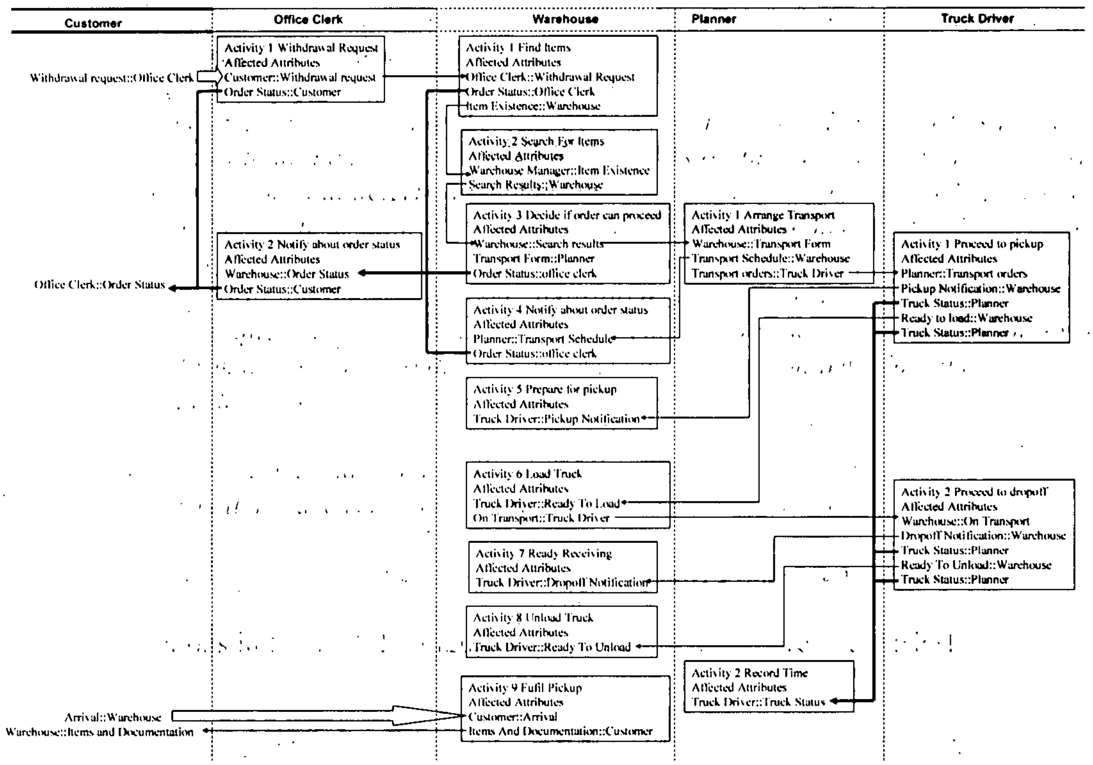
Once the office clerk has recorded the items to be withdrawn, he or she forwards the request to the manager (foreman) of the warehouse. The warehouse manager is responsible for directing the redistribution of items between warehouses. If the items are not all available in the warehouse, transport requests are issued. The warehouse manager fills out a redistribution form with the following information: items to be moved, place from which to take the items, warehouse to transport the items to, quantity to be moved, and the date by when the redistribution must be done. The warehouse manager forwards the form to the planner to organize the interwarehouse transportation of the items. The items to be moved are marked as move-pending, and the planner initiates a plan to have the items at the appropriate warehouse at the given date. Once interwarehouse transport plans are finalized, transport requests are issued to the truck drivers.

The truck driver alerts the warehouse manager of the time he or she will be at the warehouse to pick up the items. The warehouse manager gives appropriate requests to the warehouse worker on the date of delivery to have the items ready for when the truck is expected. When the warehouse worker gets a request to fetch items, he or she, at the appropriate time, orders forklift operators to move the items to the loading platform. The forklift operators execute the internal warehouse operation. When the truck driver arrives, the driver notifies the warehouse worker to have the items loaded into the truck. The truck driver notifies the next warehouse manager when it is expected to arrive at the *next warehouse. The number of items in the current warehouse decreases, and the transport request is marked as on transport.*

When the truck has arrived at the next warehouse, the truck driver notifies the warehouse worker to unload the items. The truck driver signs off the job. The warehouse workers receive the items and determine a place for them in the warehouse. Forklift operators are told to move the items to the new place in the warehouse. When the truck driver confirms the delivery of the items, the records are updated to reflect the new place for the items. The transportation time is recorded and stored. The redistribution and interwarehouse transport request are marked as performed. The warehouse worker fills in an inventory update form and sends it to the warehouse manager for confirmation and update of the inventory database.

When the customer has fetched the items the warehouse workers mark the withdrawal as ready. The items are removed (decreased) from the information system.

Based on the case we develop the Compressed ABPM diagram showing only the agent interactions in figure 3-24. Figure 3-25 is the full uncompressed ABPM.



Customer
Office Clerk
Warehouse
Planner
Truck Driver
Withdrawal request::Office Clerk
Activity 1 Withdrawal Request
Affected Attributes
Customer::Withdrawal request
Order Status::Customer
Activity 1 Find Items
Affected Attributes
Office Clerk::Withdrawal Request
Order Status::Office Clerk
Item Existence::Warehouse
Activity 2 Search For Items
Affected Attributes
Warehouse Manager::Item Existence
Search Results::Warehouse
Activity 3 Decide if order can proceed
Affected Attributes
Warehouse::Search results
Transport Form::Planner
Order Status::office clerk
Activity 1 Arrange Transport
Affected Attributes
Warehouse::Transport Form
Transport Schedule::Warehouse
Transport orders::Truck Driver
Activity 1 Proceed to pickup
Affected Attributes
Planner::Transport orders
Pickup Notification::Warehouse
Truck Status::Planner
Ready to load::Warehouse
Truck Status::Planner
Activity 2 Notify about order status
Affected Attributes
Warehouse::Order Status
Order Status::Customer
Office Clerk::Order Status
Activity 4 Notify about order status
Affected Attributes
Planner::Transport Schedule
Order Status::office clerk
Activity 5 Prepare for pickup
Affected Attributes
Truck Driver::Pickup Notification
Activity 6 Load Truck
Affected Attributes
Truck Driver::Ready To Load
On Transport::Truck Driver
Activity 2 Proceed to dropoff
Affected Attributes
Warehouse::On Transport
Dropoff Notification::Warehouse
Truck Status::Planner
Ready To Unload::Warehouse
Truck Status::Planner
Activity 7 Ready Receiving
Affected Attributes
Truck Driver::Dropoff Notification
Activity 8 Unload Truck
Affected Attributes
Truck Driver::Ready To Unload
Activity 2 Record Time
Affected Attributes
Truck Driver::Truck Status
Activity 9 Fufil Pickup
Affected Attributes
Customer::Arrival
Items And Documentation::Customer
Arrival::Warehouse
Warehouse::Items and Documentation

Figure 3-24 Compressed ABPM showing Agent Interactions

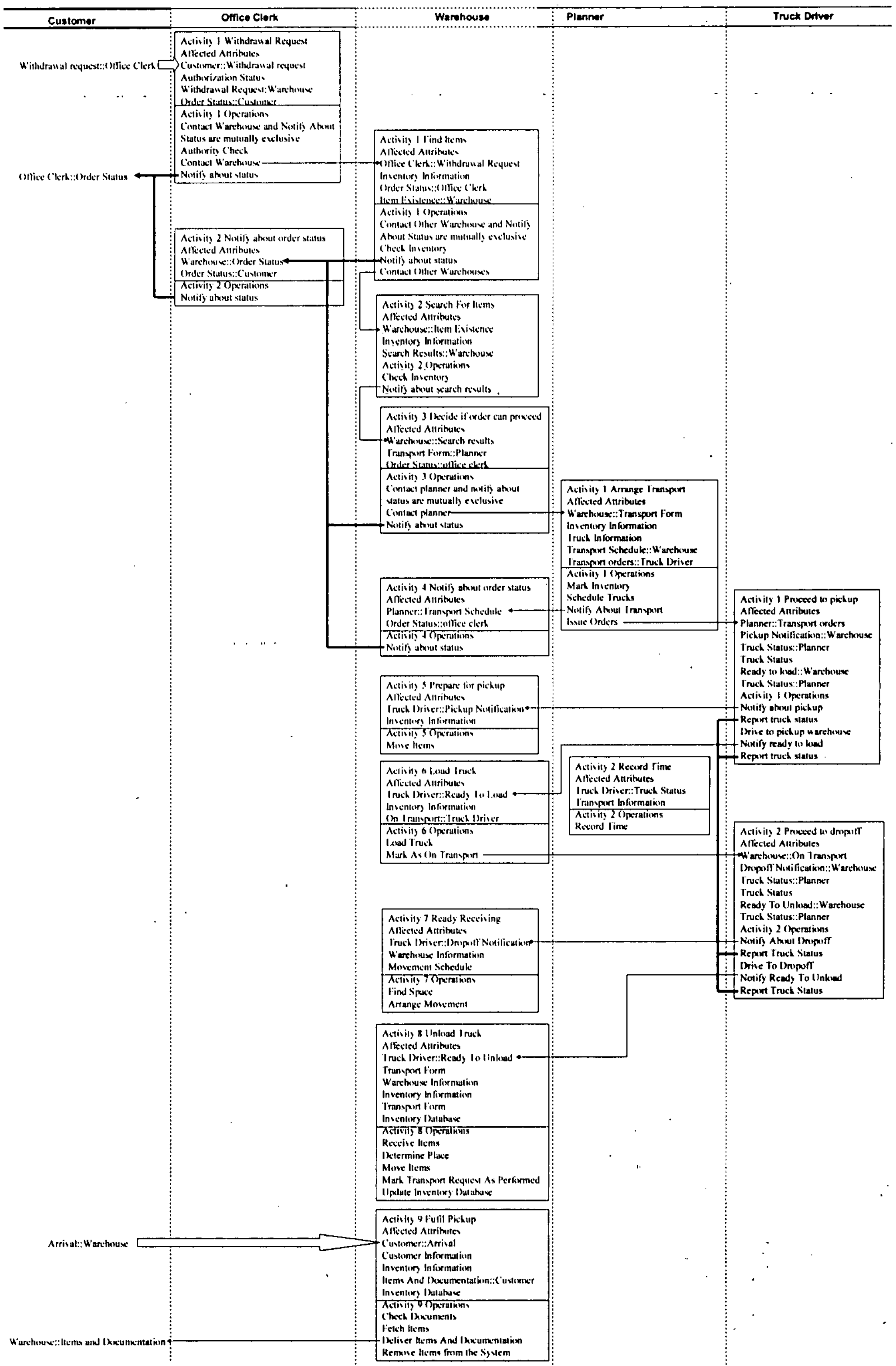


Figure 3-25 ABPM For The ACME Warehouse Management Case

A full step by step derivation of the diagram can be found in appendix A (including the associated decomposition for Warehouse), while the associated agent templates can be found in appendix B.

3.8 Summary

This chapter introduced the constructs of Ontology-Based Process Modeling. The constructs were then mapped to object-oriented concepts. This led to the introduction of an objected-oriented graphical notation for ABPM diagrams, along with a procedure for their creation and validation. To illustrate the concepts an example was presented.

4. Linking OOEM and ABPM

4.1 Introduction

This chapter has the goal of relating Object-Oriented Enterprise Modeling (OOEM) and Activity-Based Process Modeling (OBPM). OOEM is used to describe what things an organization does. ABPM is used to describe how an organization does things. Thus, it is logical that these two methodologies should be related. Also they have similar foundations, which should simplify linking the methodologies. The following diagram illustrates the proposed relationship.

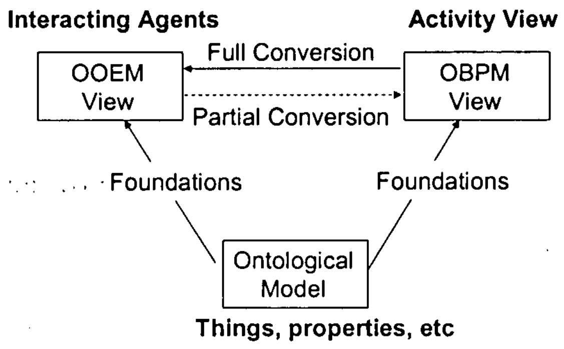


Figure 4-1. Proposed Relationship And Its Foundations.

In particular we consider the following as to how we will relate the grammars:

- A business process is everything that occurs within the system from the initial request to the system to the final response from the system.
- A service is everything that occurs within an object from the initial request to the final response to that request. That is, a process confined to one object (i.e. what is carried out).
- An activity is part of a service. An activity is everything within a service confined by (at most) 2 interactions.
- An operation is part of an activity. The operations of an activity define how the activity is carried out (and by extension a service is therefore carried out by operations).

4.2 Object-Oriented Enterprise Modeling (OOEM)

The very first question that comes to mind in systems analysis is why would Enterprise Modeling be needed? The relationship between systems analysis and enterprise modeling can be clarified as: "Systems analysis is the process of understanding the organizational environment and specifying the requirements of it. In order to specify the requirements of a system correctly, a system analyst must first understand the related business areas by developing a model of the enterprise" (Zhao, 1995, citing Gorman 1994). This suggests that the very first activity in systems analysis should be to develop an enterprise model to describe what an organization does, which will then lead to a process model being developed later to illustrate how an organization does things

4.2.1. OOEM constructs[7]

This subsection will explain the graphical constructs used in OOEM.

To begin there exist two kinds of objects. Those that are external to the system and those that are internal system. Both kinds of objects possess attributes and perform services. External objects make request to objects in the system or have requests made to them from the system. Their services and attributes are not modeled since they are external to the system, and the only thing that matters is the requests they generate or receive. Internal objects receive requests via interface attributes. The request to an object triggers a service in the object. In the course of a service occurring one or more internal attributes may be accessed, as well as one or more requests may be generated. A request may or

[7] This discussion of OOEM constructs is based on Wand and Woo 2002

may not have a response generated. And said response may be generated directly or indirectly due to the service it triggered.

4.2.2 Request Propagation[8]

External objects affect internal objects by making requests to them. In order to satisfy the request the object may invoke a service. A service is a series of actions that the object performs. During the performance of said actions requests may be generated that affect other objects (or the object itself). The affected objects may then generate requests in fulfilling their responses to the requests made by the first object, and so forth. When all requests to internal objects have been fulfilled and no further requests to internal objects are generated the OOEM is considered complete. Hence an initial request propagates through the system.

4.2.3 How To Generate An OOEM[9]

The algorithm for generating an OOEM is as follows:

1. Identify external objects
2. For each external object:
   2.1. Identify all requests generated
   2.2. For each request, identify:
      2.2.1. The object receiving it
      2.2.2. The service invoked in the object
      2.2.3. The interface attribute
      2.2.4. The response returned by the service (if any)
      2.2.5. The internal attributes (if necessary)
      2.2.6. All requests spawned by the service (if any) and repeat 2.2 for each request
3. If necessary, represent composite/component and/or super/sub-classes using those found in 2.2.1

[8] This discussion of OOEM request propagation is based on Wand and Woo 2002

4.2.4 Graphical Representation Of OOEM

OOEM has two commonly used representations. The first representation is using an object communication diagram with internal and external object templates. The object communication diagram "...employs simple notation to represent objects (Zhao, 1995)", while the object templates are where the services, attributes, and request information are stored. The second common method is to have all information on the graphic model. For our discussion we will use the second method as illustrated in Figure 4-2 showing an OOEM diagram for the aforementioned ACME Warehouse case (adapted from Wand and Woo, 1999).

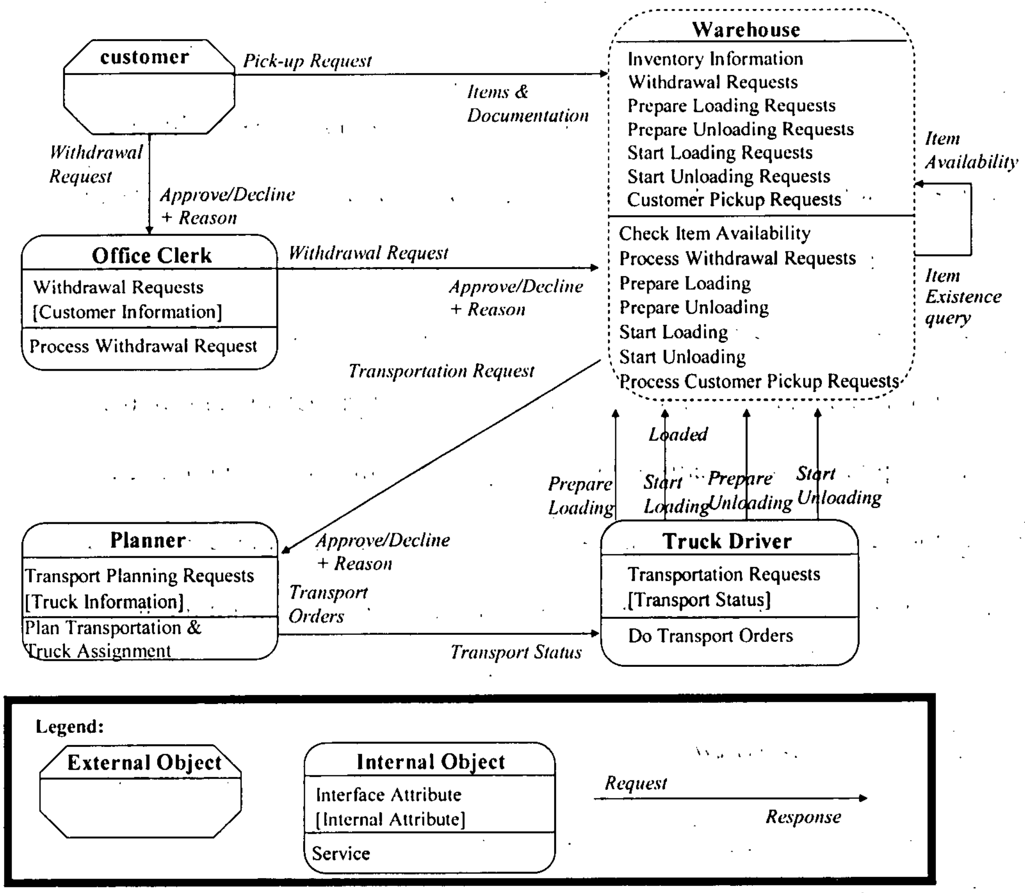


Figure 4-2 OOEM For The ACME Warehouse Management Case
note: Warehouse is a composite object hence the dotted outline

[9] This discussion of the OOEM algorithm is directly from Wand and Woo 2002

4.2.5. A Shortcoming of OOEM

In OOEM there is no indication of how the services are performed. A service is "...a well-defined series of actions taken by the object with the goal of satisfying the request. This series of actions models the state law of the thing" (Zhao, 1995). This sounds good, but it raises the question of where is the 'well-defined series of actions' represented in OOEM? Consider an OOEM service defined simply as sell goods. An executive may be more interested in an expanded view such as stock shelves, price good, display appropriate advertising, and markdown outdated inventory, in order to streamline the selling goods aspect. Also Hui (1997) notes "OOEM describes workflow participants, their responsibilities, and their interactions in a process; it does not capture the execution order of work...." Conveniently, ABPM is concerned with representing series of actions and their order (processes). Hence, if we can find the exact relationship between OOEM and ABPM we can relate what an organization does and how an organization does it in a well-defined manner.

4.3 Basis Of Conversion

The consideration to keep in mind during the conversion process is that despite different nomenclatures for the modeling grammars, ontologically a thing is still a thing, a property is still a property, etc. In order to establish a meaningful conversion process between OOEM and ABPM we need to consider how things in each grammar respond to an external event. An external event is a state change of some thing in the system.

As illustrated below in figure 4-3 in the enterprise modeling approach an external object generates a request to an object in the system causing it to go from s_0^1 to s_1^1. The notation s_x^y denotes the object y in state x. Each time there is a state change it will be noted in the diagram. In servicing the request, Object 1 generates a request to Object 2, and so forth. We can then reinterpret the OOEM approach into a state change view as illustrated in figure 4-4 below.

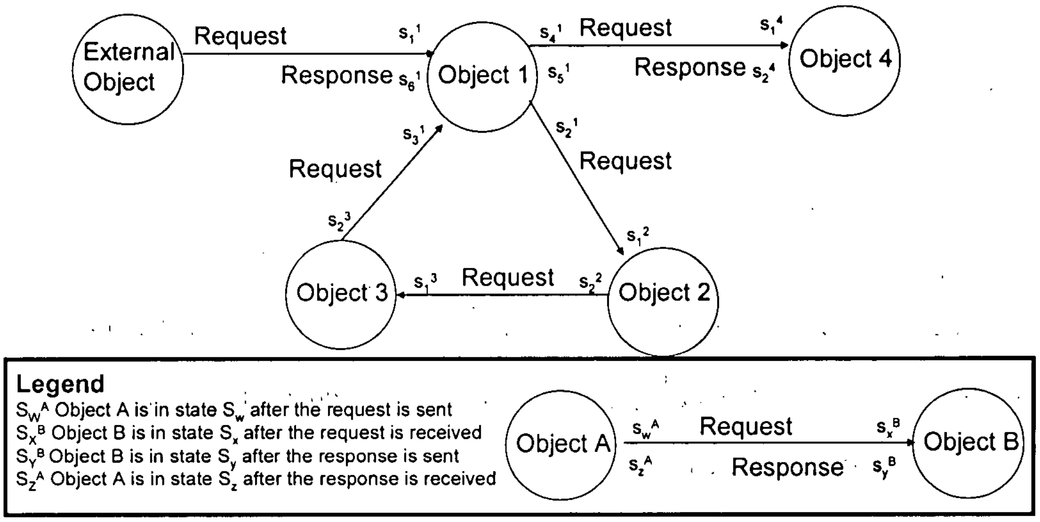


Figure 4-3 The Enterprise Modelling Approach

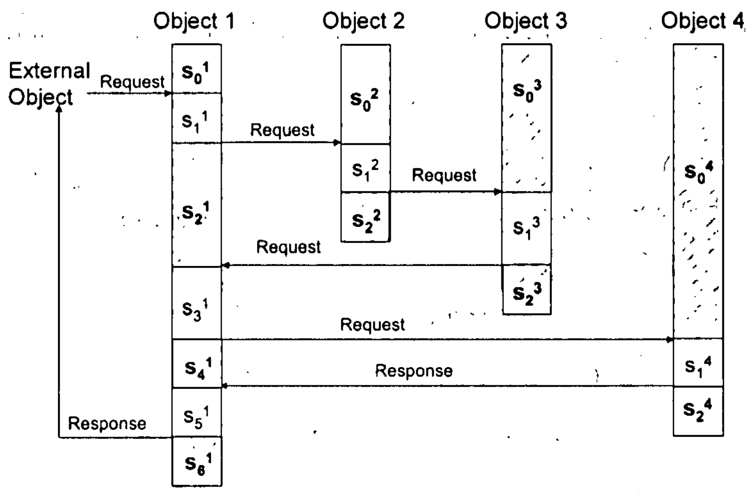


Figure 4-4 OOEM reinterpreted as a state change view

We can look at the exact same series of happenings from the process modeling approach. As illustrated below in figure 4-5, an external agent causes a change in Agent 1 causing it to go from s_0^1 to s_1^1. Each time there is a state change it will be noted in the diagram. In response to the change activity 1 one is triggered. During activity 1 of agent 1 a change is caused in agent 2, and so forth. We can then also reinterpret the ABPM approach into a state change view as illustrated in figure 4-6 below.

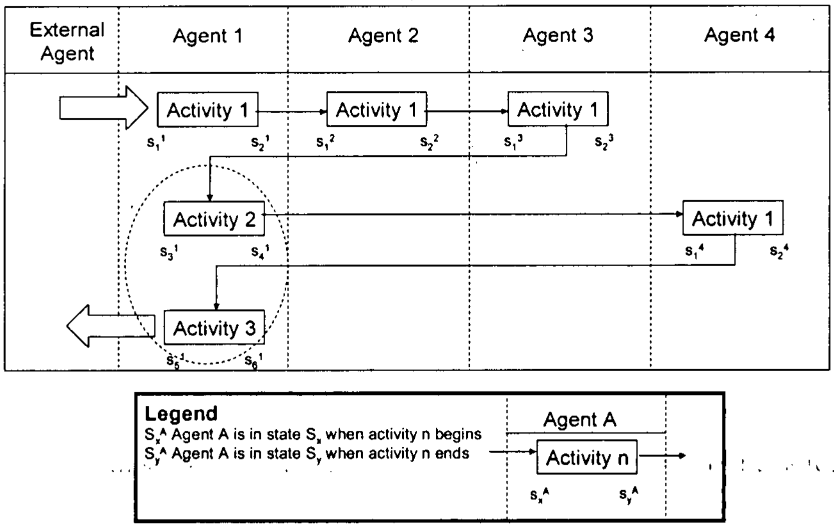


Figure 4-5 The Process Modeling Approach

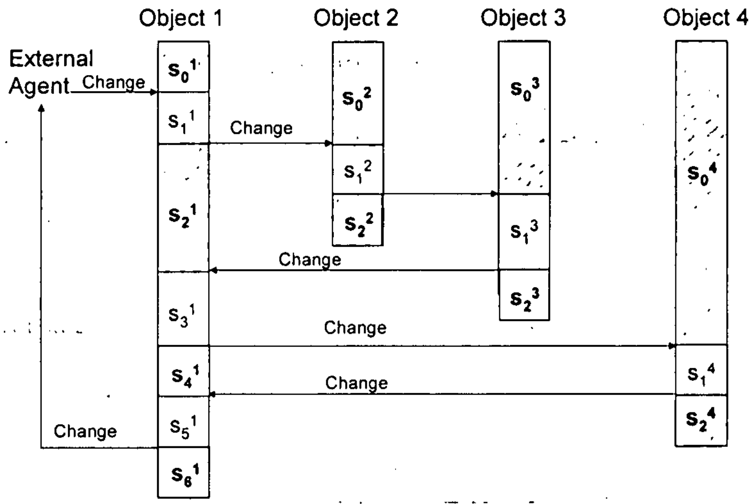


Figure 4-6 ABPM reinterpreted as a state change view

When we take these happenings back to the ontological level basically the same happenings occur in both grammars. As illustrated below in figure 4-7 an external thing modifies some mutual attribute that it shares with some thing internal to the system.

Internal thing 1 goes from being in a stable state s_0^1 to an unstable state s_1^1. Internal thing 1 will according to its laws transition from unstable state s_1^1 to stable state s_2^1, during which time it will modify some mutual attribute it shares with internal thing 2 causing internal thing 2 to transition from stable state s_0^2 to s_1^2, and so forth.

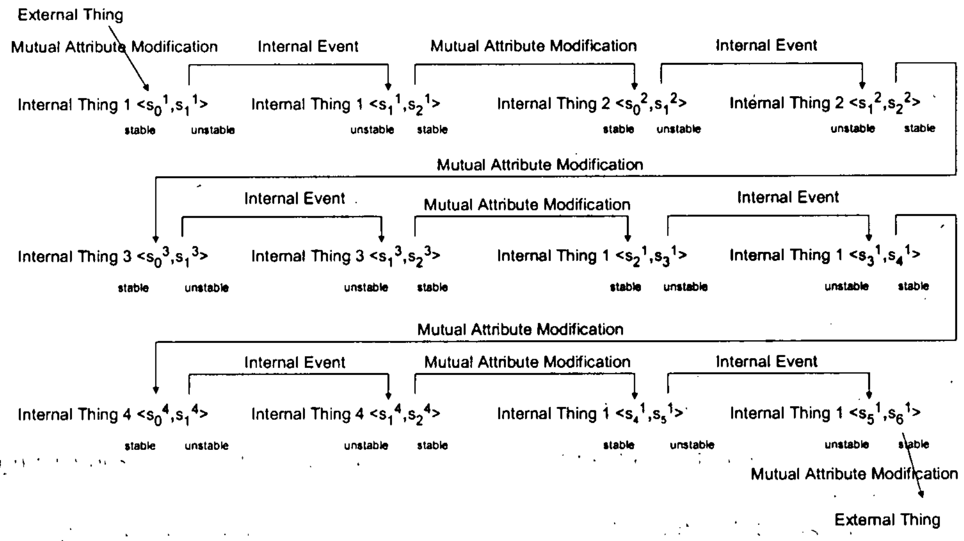

External Thing
Mutual Attribute Modification
Internal Event
Mutual Attribute Modification
Internal Event
Internal Thing 1 <s0^1,s1^1> stable unstable
Internal Thing 1 <s1^1,s2^1> unstable stable
Internal Thing 2 <s0^2,s1^2> stable unstable
Internal Thing 2 <s1^2,s2^2> unstable stable
Mutual Attribute Modification
Internal Event
Mutual Attribute Modification
Internal Event
Internal Thing 3 <s0^3,s1^3> stable unstable
Internal Thing 3 <s1^3,s2^3> unstable stable
Internal Thing 1 <s2^1,s3^1> stable unstable
Internal Thing 1 <s3^1,s4^1> unstable stable
Mutual Attribute Modification
Internal Event
Mutual Attribute Modification
Internal Event
Internal Thing 4 <s0^4,s1^4> stable unstable
Internal Thing 4 <s1^4,s2^4> unstable stable
Internal Thing 1 <s4^1,s5^1> stable unstable
Internal Thing 1 <s5^1,s6^1> unstable stable
Mutual Attribute Modification
External Thing

Figure 4-7 The Ontological Meaning Approach

Consider the dotted oval in figure 4-5, this could be where there is what we call an OOEM service yet they are distinct ABPM activities. The only way to know is to consider the interactions and what they mean on an ontological level.

From the ontological level the main difference between the two grammars is that in ABPM you can expand the internal events and include information that you cannot include in an OOEM. If we map the OOEM and ABPM constructs back to their ontological meanings we get the conversions in table 4-1.

OOEM Construct	**Ontological Model of a process (BWWP) Construct**	**ABPM Construct**
Object	Thing	Agent
Object	Simple Thing	Component Agent
Composite Object	Composite Thing	Composite Agent
	Actor*	Agent

	Non-actor*	Resource (An Agent with no services)
	Actuator*	Agent with at least one outgoing interface attribute
	Propagator*	An actuator with at least one outgoing interface attribute affecting an actor
Attribute	Property	Attribute
Internal attribute	Attribute representing an intrinsic property	Internal attribute
Interface attribute	Attribute representing a mutual property	Incoming Interface Attribute + Outgoing Interface Attribute
Inherited properties of a subclass	Hereditary property	Inherited properties of a subagent
Composite object emergent properties	Emergent property	Composite agent emergent properties
Attribute values	State	Attribute values
No services required by an object	Stable state	No operations required by an Agent
Services required by an object	Unstable state	Operations required by an agent
State Change	Event	State change
Execution of a service	Internal event	Change of an internal attribute, performed by an operation
Receiving a request or Receiving a response	External event	Change of an incoming interface attribute
Service	Transformation	Operation
part of a service	Activity	A sequence of transformations
Law	Law	Law
State law	State law	State Law
	Transformation law	Transformation Law
What happens in a system from an initial request to the final response	Process*	The changes the interacting agents and resources undergo from when one agent becomes unstable to all agents once again being in stable states
Request/Response	Interaction	Change
x and y have a shared interface attribute modifiable by x only	x acts-on y	x has an outgoing interface attribute connected to an incoming interface attribute of y, only x can modify the outgoing interface attribute and thus by extension the incoming interface attribute of y
Definition of a class	Functional Schema	Definition of an agent

Table 4-1 Conversion Table
***Denotes a construct from the BWWP ontology not in the BWW Ontology**

An OOEM transformation deals with only 1 incoming interface attribute and 1 service, whereas the ABPM transformation laws are constraints on two or more operations hence for this instance there is no mapping of a transformation law in table 4-1.

Keep in mind, in an ABPM to OOEM conversion we are moving from a detailed view to a more abstract view. As mentioned above an OOEM service represents a well-defined series of actions, while ABPM delves into what those actions are. Hence we will be losing details in an ABPM to OOEM conversion. Also since an ABPM diagram is a more detailed diagram than an OOEM, it may not be possible to fully construct an ABPM based on an OOEM. Where needed, we will explicitly mention what information needs to be retrieved from domain knowledge that cannot be found in the OOEM diagram.

The fact ABPM is more detailed than OOEM also leads to issues of duplication. Duplication is caused in ABPM due to the possibility of the same event being initiated via different activities, which does not occur in OOEM. Consider our solution to the ACME case from the OOEM perspective, the response from warehouse to office clerk of "Approve/Decline + Reason" is spawned in only 1 service, the service process withdrawal requests. In ABPM the same interaction of notifying the office clerk if the order is approved or declined may occur in the first, third, or fourth activity of the warehouse.

4.3 ABPM to OOEM Conversion

Due to the fact that an ABPM is more detailed than an OOEM the conversion process may create duplicate requests, responses, or attributes. Duplicates can be left out, since they are not needed in OOEM. We will use the ACME case as a running example of the conversion process.

Table 4-2 illustrates the questions of interest and conversion steps that answer them.

Question Of Interest	Conversion Step
How do we determine the external objects?	Step 1. External Object Conversion
How do we determine the internal objects?	Step 2. Internal Object Conversion
How do we determine requests?	Step 3. Request/Response Identification
How do we determine responses?	Step 3. Request/Response Identification
How do we determine interface attributes?	Step 4. Interface Attribute Conversion
How de we determine internal attributes?	Step 5. Internal Attribute Conversion
How do we determine services?	Step 6. Service Creation

Table 4-2 Questions of interest in an ABPM to OOEM Conversion

We will now summarize the steps in the ABPM to OOEM conversion process. Following the summarization will be the explanation behind each step and a short example to illustrate each step.

4.4.1 ABPM To OOEM Conversion Steps

The Steps for converting from an ABPM to an OOEM are:

Step 1. External Object Conversion. Every ABPM external agent becomes an OOEM external object.

Step 2. Internal Object Conversion. Every ABPM internal agent becomes an OOEM internal object

Step 3. Request and Response Identification. Outgoing interface attributes become requests. However if the last outgoing interface attribute is going to the same agent which initiated the activity in which the outgoing interface attribute is found the and the outgoing interface attribute used to initiate the activity became a request; the outgoing interface attribute becomes a response. This step may encounter duplicate changes, duplicates can be left out.

Step 4. Interface Attribute Conversion. An incoming interface attribute to handle a change from an agent becomes an interface attribute to handle a request. This only applies to those incoming interface attributes of which the associated outgoing interface attribute became a request.

Step 5. Internal Attribute Conversion. An ABPM internal attribute becomes an OOEM internal attribute. This step may generate duplicate internal attributes, duplicates can be left out.

Step 6. Service Creation. Create a service to process every request. This step requires the modeler to create a service themselves since the service is not present in the process model.

4.4.2 ABPM To OOEM Conversion Step Derivation

Step 1. External Object Conversion. Every ABPM external agent becomes an OOEM external object.

The scope of a system does not change depending on what view of it is used. In our example there is only one external ABPM agent, hence there is only one external OOEM object

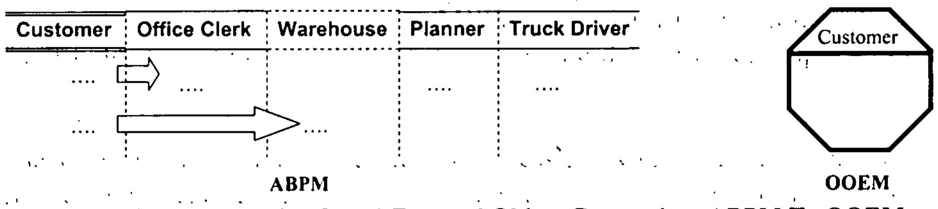


Figure 4-8 Demonstrating Step 1 External Object Conversion; ABPM To OOEM

Step 2. Internal Object Conversion. Every ABPM internal agent becomes an OOEM internal object

The scope of a system does not change depending on what view of it is used. As well, things of interest are still things of interest regardless of level. In the ACME case there are four ABPM internal agents hence there will be four OOEM internal objects.

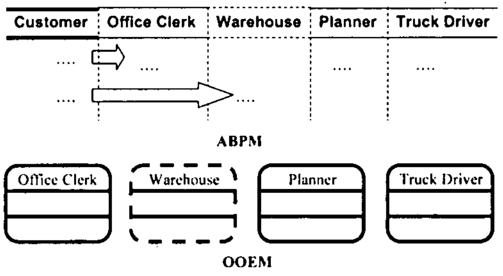
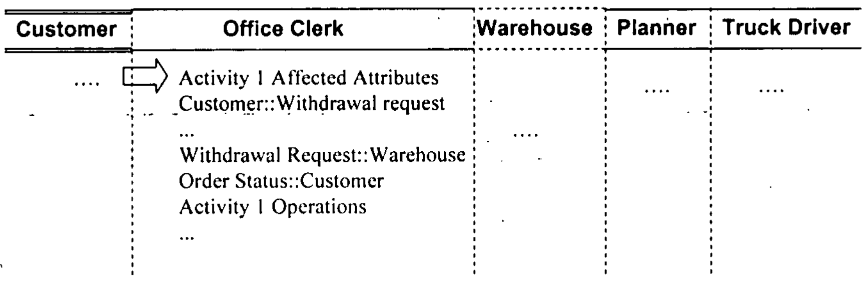


Figure 4-9 Demonstrating Step 2 Internal Object Conversion; ABPM To OOEM

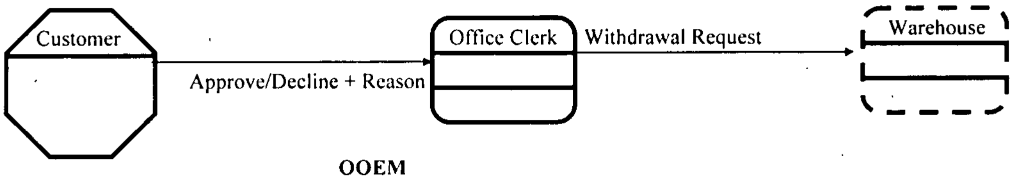
Step 3. Request and Response Identification. Outgoing interface attributes become requests. However if the last outgoing interface attribute is going to the same agent which initiated the activity in which the outgoing interface attribute is found the and the outgoing interface attribute used to initiate the activity became a request; the outgoing interface attribute becomes a response[10]. This step may encounter duplicate changes, duplicates can be left out.

According to our mapping in table 4-1 a change of an incoming interface attribute is ontologically equivalent to either a request or a response. An incoming interface attribute is changed via changing the outgoing interface attribute that is associated with it. Duplication may be caused by branching which occurs in process modeling that does not occur in enterprise modeling.

[10] This is a complete assumption on our part, made for simplicity. We recognize that an object may receive a response from an object it never made a request to, thus it should be possible to extend this step to be more robust, however we will use this simple form.



ABPM



OOEM

Figure 4-10 Demonstrating Step 3 Request and Response Identification; ABPM To OOEM
***Recall a request appears at the tail of an arrow, a response at the head of an arrow**

Step 4. Interface Attribute Conversion. An incoming interface attribute to handle a change from an agent becomes an interface attribute to handle a request. This only applies to those incoming interface attributes of which the associated outgoing interface attribute became a request.

An incoming interface attribute is how an agent handles (i.e. receives) external events. An external event is handled (received) by an object through an interface attribute. For those incoming interface attributes which the associated outgoing interface attributes became a response an interface attribute is not needed in the object since responses do not have interface attributes associated with them[11].

[11] Responses can actually have interface attributes to handle them, however due to convention they typically do not.

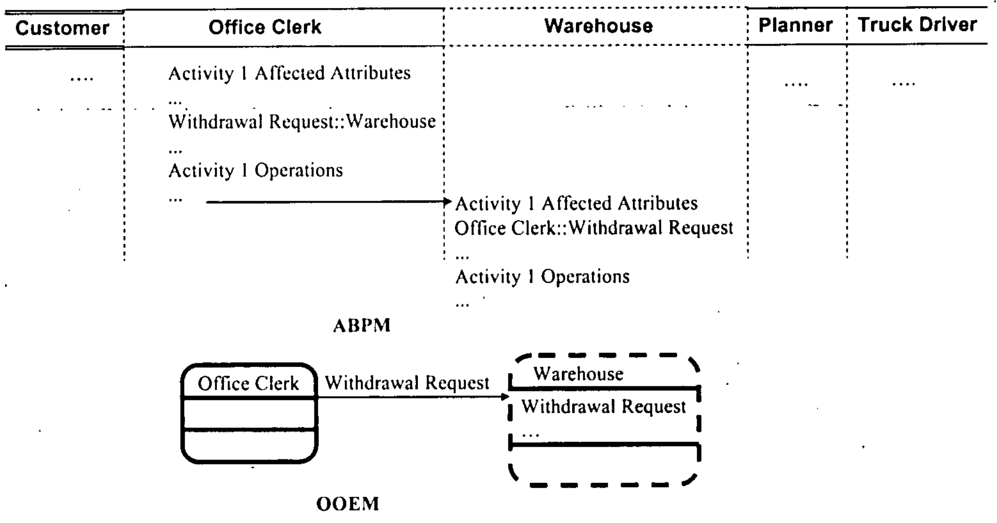


Figure 4-11 Demonstrating Step 4 Interface Attribute Conversion; ABPM To OOEM

Step 5. Internal Attribute Conversion. An ABPM internal attribute becomes an OOEM internal attribute. This step may generate duplicate internal attributes, duplicates can be left out.

According to our mapping in table 4-1, an internal attribute is the same in both grammars. The duplication is caused by branching that is present in process models but not in enterprise modeling.

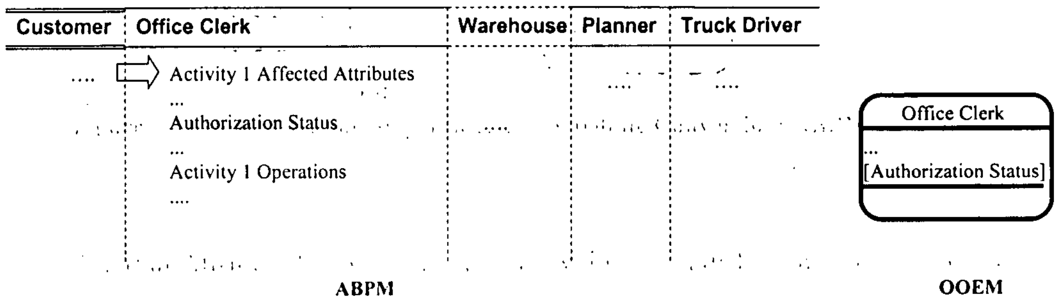


Figure 4-12 Demonstrating Step 5 Internal Attribute Conversion; ABPM To OOEM

Step 6. Service Creation. Create a service to process every request. This step requires the modeler to create a service themselves since the service is not present in the process model.

The important thing to keep in mind in that in OOEM every request requires a service. What we know is that every request started as an outgoing interface attribute. If there are duplicate outgoing interface attributes we know the activities in which the outgoing interface attribute is found are part of the same service, since a request is spawned by only one service. Thus a potential candidate for a service is the operation that changes the outgoing interface attribute, and when there are duplicate outgoing interface attributes there are multiple candidate operations for a service. It is still a modeler's decision to either use one of the candidate operations as a service or to create a service from scratch.

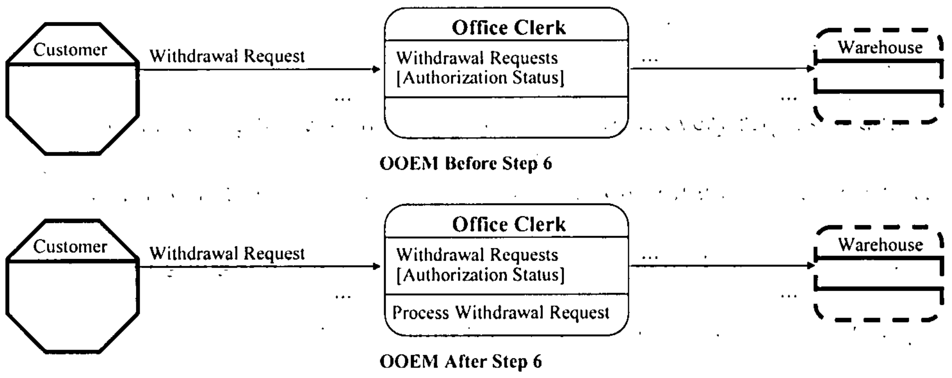


Figure 4-13 Demonstrating Step 6 Service Creation; ABPM To OOEM

At this point we are finished the conversion. If we compare our solution after the conversion to that in section 4.2.4 above we see the difference is the post conversion solution has more internal attributes. Since internal attributes are optional in OOEM the solution is the same.

4.5 OOEM to ABPM Conversion

The following questions are of interest in an ABPM to OOEM conversion:

Question Of Interest	Conversion Step
How do we determine the external agents?	Step 1. External Agent Conversion
How do we determine the internal agents	Step 2. Internal Agent Conversion
How do we determine activities?	Step 5. Activity Creation
How do we determine outgoing interface attributes?	Step 3. Outgoing Interface Attribute Identification
How do we determine incoming interface attributes?	Step 4. Incoming Interface Attribute Identification
How de we determine internal attributes?	Step 7. Internal Attribute Identification
How do we determine operations?	Step 8. Operation Identification.
How do we determine state laws?	Step 5. Activity Creation
How do we determine transformation laws?	Step 5. Activity Creation Step 8. Operation Identification
How do we determine resources?	Step 9. Resource Identification

Table 4-3 Questions of interest in an OOEM to ABPM Conversion

We will now summarize the steps in the OOEM to ABPM conversion process. Following the summarization will be the explanation behind each step and a short example to illustrate each step.

4.5.1 OOEM To ABPM Conversion Steps

The Steps for converting from an OOEM to an ABPM are:

Step 1. External Agent Conversion. Every OOEM external object becomes an external ABPM agent.

Step 2. Every OOEM internal object becomes an ABPM internal agent

Step 3. Outgoing Interface Attribute Identification. All requests and responses become outgoing interface attributes.

Step 4. Incoming Interface Attribute Identification. All outgoing interface attributes identified in step 3 require an incoming interface attribute. As well when we create the incoming interface attribute we need to show the agents share a mutual attribute.

Step 5. Activity Creation. At this point we can create activities. Each time there is an incoming interface attribute without a transformation law requiring other incoming interface attributes to change to start the operations of the activity, it is the beginning of a new activity. We can create transformation and state laws for the incoming interface attributes using domain knowledge.

Step 6. Outgoing Interface Attribute Assignment. We can now assign the outgoing interface attributes to the appropriate activity using domain knowledge. Duplicate assignments are allowed.

Step 7. Internal Attribute Identification. Every OOEM internal attribute becomes an ABPM internal attribute. We can the assign them to the appropriate activity using domain knowledge. Additional internal attributes may need to be created using domain knowledge. Duplication may occur

Step 8. Operation Identification. Create the operations that will be invoked to change the outgoing interface and internal attributes using domain knowledge. We also have to create transformation laws using domain knowledge that show if there are any restrictions on the operations of the activity.

Step 9. Resource Identification. In the event there are agents only have incoming interface attributes in all of their activities they are resources and need to be converted to the appropriate resource notation.

4.5.2 OOEM To ABPM Conversion Step Derivations

Step 1. External Agent Conversion. Every OOEM external object becomes an external ABPM agent.

The scope of a system does not change depending on what view of it is used. In our example there is only one external OOEM object, hence there is only one external ABPM agent.

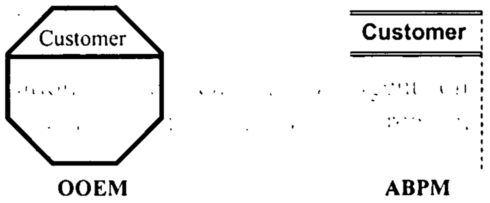


Figure 4-14 Demonstrating Step 1 External Agent Conversion; OOEM To ABPM

Step 2. Internal Agent Conversion. Every OOEM internal object becomes an ABPM internal agent

The scope of a system does not change depending on what view of it is used. As well, things of interest are still things of interest regardless of level. In the ACME case there are four OOEM internal objects hence there will be four ABPM internal agents.

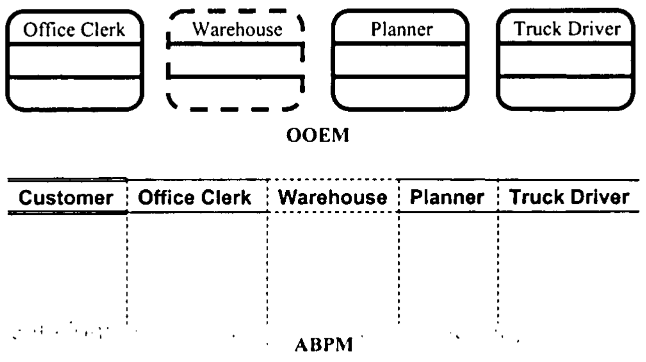
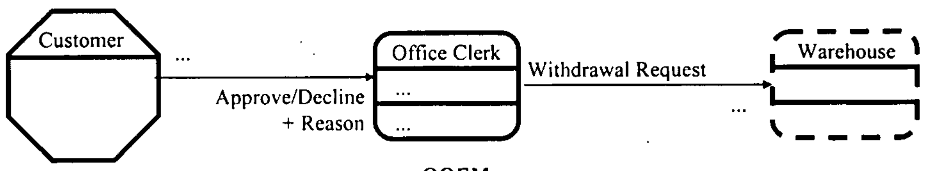


Figure 4-15 Demonstrating Step 2 Internal Agent Conversion; OOEM To ABPM

Step 3. Outgoing Interface Attribute Identification. All requests and responses become outgoing interface attributes.

Since requests and responses model interaction between two things in OOEM, they need to model interaction between two things in ABPM. A change models interaction in ABPM. A change is the initiation of an event external to an agent. External events change incoming interface attributes. Since incoming interface attributes are only changed when the associated outgoing interface attributes are changed by operations in other agents, requests and responses must become outgoing interface attributes.



OOEM

Customer	Office Clerk	Warehouse	Planner	Truck Driver
	Order Status::Customer Withdrawal Request::Warehouse			

ABPM

Figure 4-16 Demonstrating Step 3 Outgoing Interface Attribute Identification; OOEM To ABPM

Step 4. Incoming Interface Attribute Identification. All outgoing interface attributes identified in step 3 require an incoming interface attribute. As well when we create the incoming interface attribute we need to show the agents share a mutual attribute.

By definition all outgoing interface attributes need an associated incoming interface attribute. We do not need additional information from the OOEM for this step.

Customer	Office Clerk	Warehouse	Planner	Truck Driver
	Order Status::Customer Withdrawal Request::Warehouse			

ABPM, before step 4

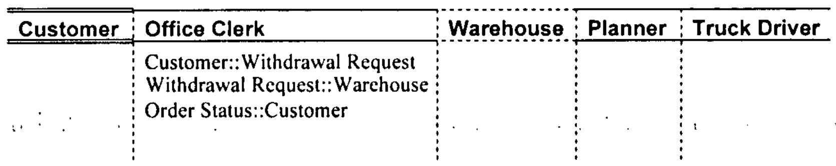
Customer	Office Clerk	Warehouse	Planner	Truck Driver
Office Clerk::Order Status	← Order Status::Customer			
	Withdrawal Request::Warehouse →	Office Clerk::Withdrawal Request		

ABPM, after step 4

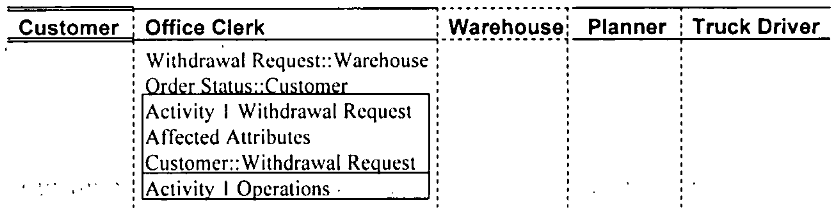
Figure 4-17 Demonstrating Step 4 Incoming Interface Attribute Identification; OOEM To ABPM

Step 5. Activity Creation. At this point we can create activities. Each time there is an incoming interface attribute without a transformation law requiring other incoming interface attributes to change to start the operations of the activity, it is the beginning of a new activity. We can create transformation and state laws for the incoming interface attributes using domain knowledge.

As noted in our mapping in table 4-1 the transformation law information is not present in OOEM. Thus we will have to use our domain knowledge. At this point (in the ACME case) there do not appear to be any transformation laws. That means every incoming interface attribute will be the beginning of a new activity. We need to create the state laws using domain knowledge since they are not represented directly in OOEM



ABPM, before step 5



ABPM, after step 5

Figure 4-18 Demonstrating Step 5 Activity Creation; OOEM To ABPM
***note the attributes above the activity are outgoing interface attribute that have not yet been assigned to an activity**

Step 6. Outgoing Interface Attribute Assignment. We can now assign the outgoing interface attributes to the appropriate activity using domain knowledge. Duplicate assignments are allowed.

Duplicate assignments can occur since the same transformation may be triggered in different activities. We must use the domain knowledge to assign the outgoing interface attributes to the appropriate activity since the expanded activity view is not present in OOEM.

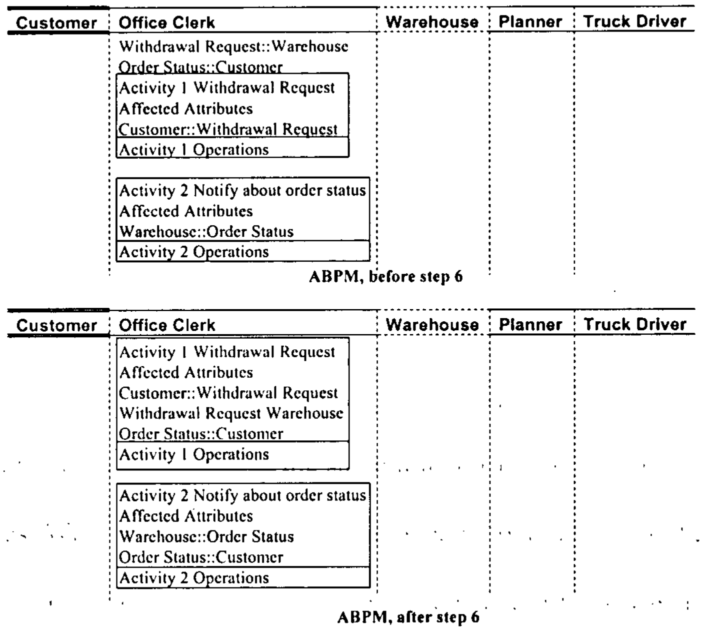


Figure 4-19 Demonstrating Step 6 Outgoing Interface Attribute Assignment; OOEM To ABPM

Step 7. Internal Attribute Identification. Every OOEM internal attribute becomes an ABPM internal attribute. We can the assign them to the appropriate activity using domain knowledge. Additional internal attributes may need to be created using domain knowledge. Duplication may occur

According to our mapping an internal attribute is the same in both grammars. However in OOEM internal attributes are optional, thus not all the ABPM internal attributes may be identified. A modeler will have to use their domain knowledge (in our case that's the ACME Case) to identify any other internal attributes.

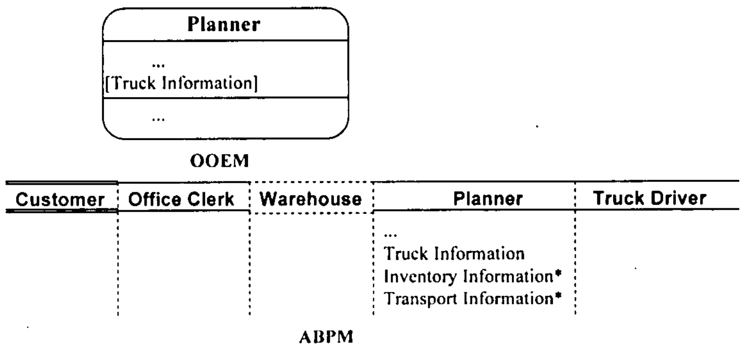
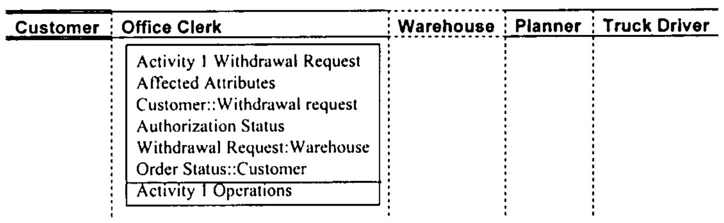


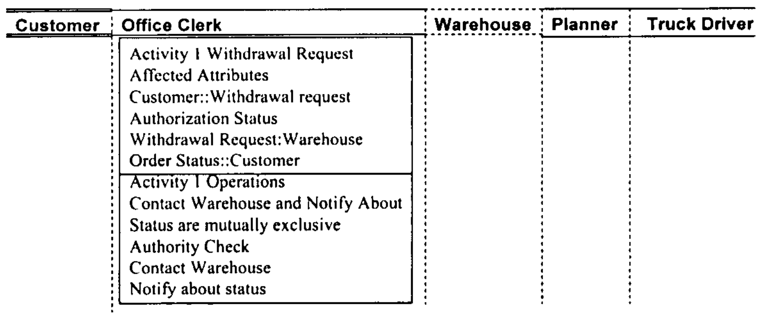
Figure 4-20 Demonstrating Step 7 Internal Attribute Identification; OOEM To ABPM
*** denotes an attribute created using domain knowledge**

Step 8. Operation Identification. Create the operations that will be invoked to change the outgoing interface and internal attributes using domain knowledge. We also have to create transformation laws using domain knowledge that show if there are any restrictions on the operations of the activity.

We cannot use OOEM services to create ABPM operations since as mentioned above a service may encompass many operations. The only way outgoing interface and internal attributes are changed is via operations. As noted in our mapping in table 4-1 the transformation law information is not present in OOEM. Thus we will have to use our domain knowledge.



ABPM, before step 8



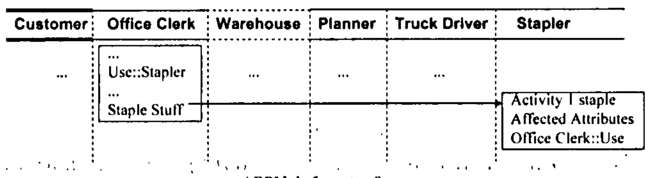
ABPM, after step 8

Figure 4-21 Demonstrating Step 8 Operation Identification; OOEM To ABPM

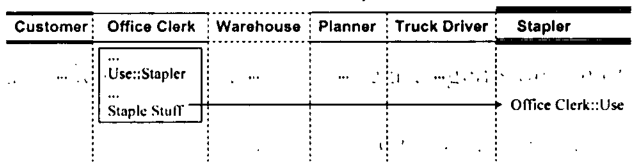
Step 9. Resource Identification. In the event there are agents only have incoming interface attributes in all of their activities they are resources and need to be converted to the appropriate resource notation.

Resources are the only things that can receive change that are not agents.

Note: for our example there are no outgoing interface attributes that meet the criteria of step 9, however an example would be something like figure 4-22 below.



ABPM, before step 9



ABPM, after step 9

Figure 4-22Demonstrating Step 9 Resource Identification; OOEM To ABPM

4.6 Converting Decompositions and Compositions

It should be noted that in our running conversion examples above we neglected to mention and demonstrate the warehouse decomposition. The reason behind that is there is no need to show it. The Steps apply the same way in the decomposed view or the composite view since in the decomposed view the component things are still things that interact and will still interact in either OOEM or ABPM, the difference is the information included.

4.7 Converting Subagents/Superagents And Subclasses/Superclasses

It should also be noted that in our running conversion examples above we neglected to mention and demonstrate the generalization/specialization conversions between agents and classes and vice versa. They are still things so thus the steps apply the same way to the things in these relationships. The modeler needs to ensure that the correct generalization and specialization notation for the grammar is used post conversion.

4.8 Summary

This chapter explained OOEM. It delved into the relationship between OOEM and ABPM and introduced means to convert form one view to the other. This chapter also used the examples from the ACME case to demonstrate the conversion process.

5. Design Principles of an ABPM CASE Tool

5.1 Introduction

One of the goals of this thesis is to take OBPM from an algorithm to a graphical modeling grammar. To make it more useful as a modeling grammar a tool that supports the ABPM modeling process can be developed. This chapter presents a set of design principles for taking ABPM a step further and implementing it as a CASE tool. ABPM has very specific semantics that proper design of a CASE tool can take advantage of. Our only assumption is that there already exists some sort of CASE tool upon which OOEM has been implemented.

5.2 System Goals

The main goal of the CASE tool development is to implement ABPM in a visual manner. This manner should be consistent with both the semantics and the activity-based graphical representations presented back in chapter 3. However as a secondary goal, it should also support the ABPM to OOEM conversion and OOEM to ABPM conversion from chapter 4.

5.3 System Requirements

Based on Zhang's requirements for a visual OOEM CASE tool (Zhang, 1998), we can say the CASE tool will be required to support the entire ABPM model creation process. In the initial stage the tool should allow the process modeler to gather and organize information into an understandable model. The tool should then be capable of applying the ABPM rules to a developed model to evaluate its semantic correctness. Once evaluated it should be possible to change the model as needed until the rules are met. Finally the tool should allow for the modeler to suspend activity at any time and resume it later. No information should be lost during these actions (Zhang, 1998).

Also, the case tool should be able to support the analysis stage. That is, the user should be able to change elements as desired so that they can see what happens when changes are made. (Zhang, 1998) This directly supports process redesign efforts.

The CASE tool should also support the model conversion process from ABPM to OOEM and from OOEM to ABPM without losing any information during the conversion process.

In order to meet these requirements we need to consider two main areas: what constructs to represent in the CASE tool and what interactions a user will have with the CASE tool.

5.4 Constructs to Represent

If we want to build upon the existing OOEM CASE tool then when it comes to constructs to represent we can first need to consider what is already present in an OOEM CASE tool. We can then compare that with the constructs in ABPM. After the comparision we should then know what needs to be added to the CASE tool.

The basis of comparison will be the respective metamodels of both grammars. The OOEM metamodel, as adapted from Tan (Tan, 1997) and his work on an OOEM CASE tool to show attributes are either internal or interface and responses can be spawned by services, is presented below in figure 5-1.

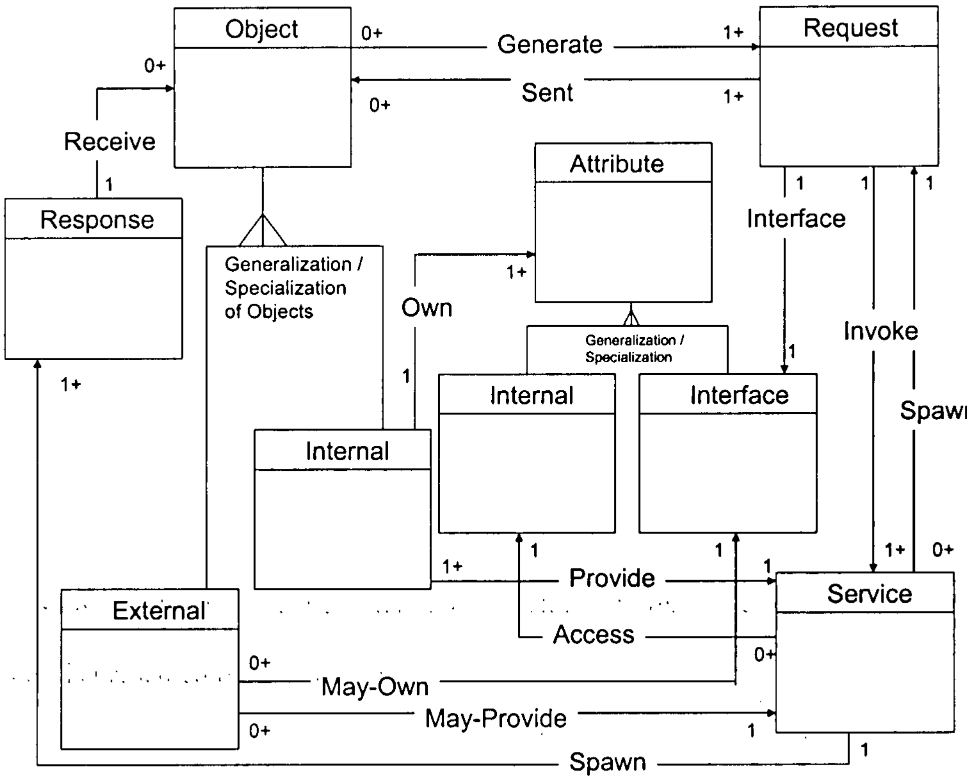


Figure 5-1 The OOEM Metamodel (adapted from Tan 1997)

We developed the ABPM metamodel back in chapter 3. The question now is how do we relate the metamodels to create a CASE tool metamodel? The question is answered the same way as how we related the two modeling grammars, use their ontological foundations to establish what is there and what is not. Consider figure 5-2 below, as an OOEM Metamodel with the ontological meanings of the constructs added.

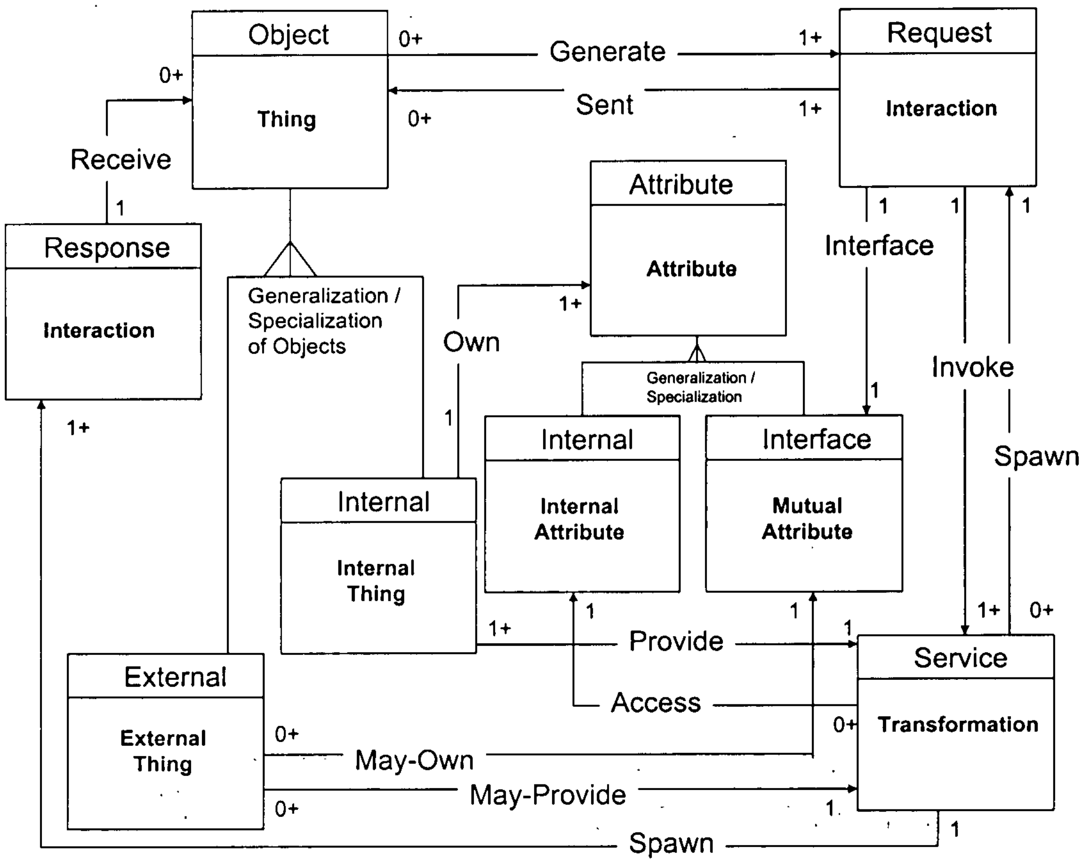


Figure 5-2 OOEM Metamodel with ontological meaning included

When we look at the ABPM model with the ontological meanings of the constructs added we get the diagram in figure 5-3 below. Since an activity has no direct ontological mapping we did not include its ontological meaning. The boxes that are double-lined denote the constructs that are directly equivalent, and are thus already represented. The dotted box denotes that although not directly equivalent on their own, combined an outgoing interface attribute and incoming interface attribute have an ontological meaning of interaction.

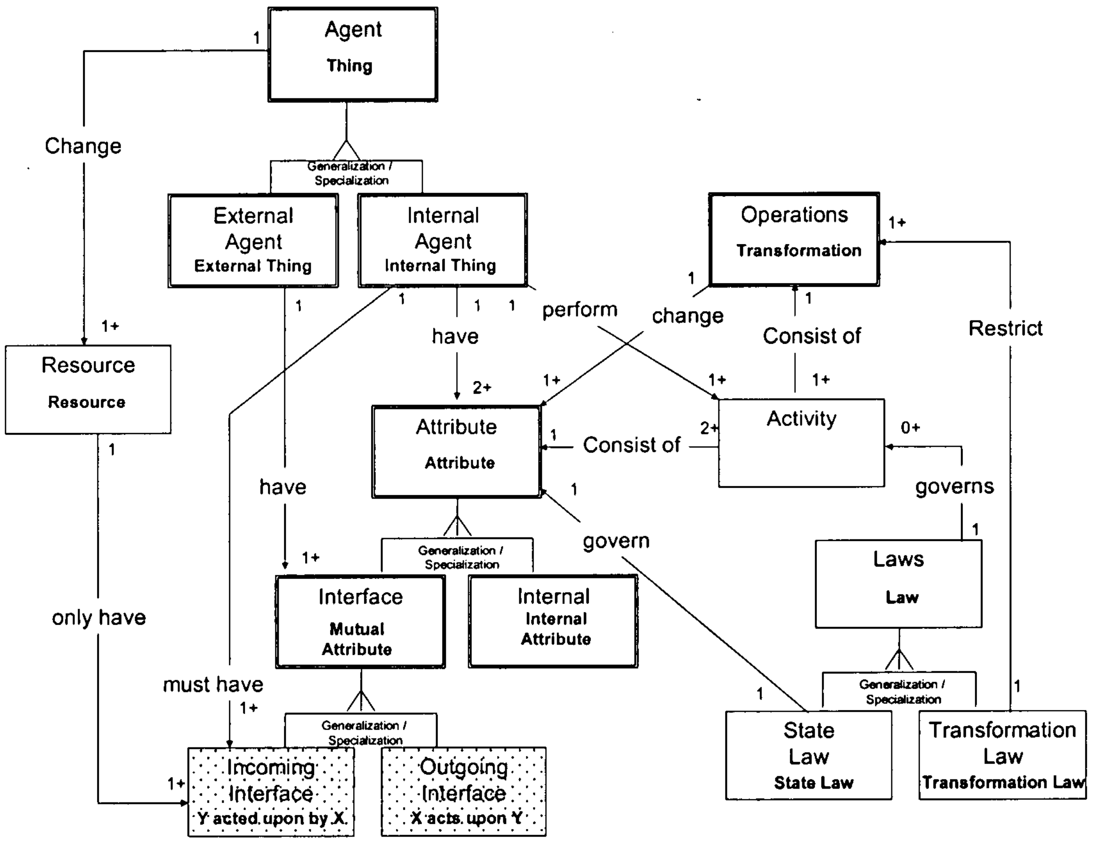


Figure 5-3 ABPM metamodel with ontological meaning included

From the two metamodels we can thus see, an ABPM includes all the constructs presented in an OOEM. However the ABPM has additional constructs not present. Thus the additional constructs to be represented are: Resource, Activity, Laws, State Laws, and Transformation Laws, While the constructs that are already represented (ontologically) but need to be configured to accept input as ABPM constructs are: Agent, External Agent, Internal Agent, Attribute, Internal Attribute, Operations, Incoming Interface Attributes, and Outgoing Interface Attributes..

5.5 User Interactions

The user interactions for the ABPM CASE tool are those related to drawing ABPM diagrams, converting ABPM and OOEM diagrams, display option, and semantic checker.

5.5.1 Drawing ABPM Diagram Interactions

For all elements a user should be able to add, delete, and modify as desired. For the elements where it is not obvious how to add, delete, or modify it will be discussed below.

5.5.1.1 Internal and External Agents

We recommend that when the agents and resources are implemented, that each construct will have the associated agent template implemented with it as well. This will allow the user to work from the view that is more convenient for them. When the user wishes to make a modification they should merely have to specify which view they wish to work in. If chosen, changes made in the agent template view should be reflected in the diagram when the template is closed.

5.5.1.2 Agents Sharing A Mutual Attribute

When an incoming interface attribute has an outgoing interface attribute associated with it, an arrow from the operation that changes the outgoing interface attribute to the incoming interface attribute should drawn. In the event the same outgoing interface attribute occurs in more than one activity, then when they are associated with the incoming interface attribute a function will have to run that automatically generates a double-lined arrow. When compressed without operations the arrows can automatically be redrawn to connect directly from the outgoing interface attributes to the incoming interface attributes. Mores is said about compression in section 5.5.2 below.

5.5.1.3 Superagents And Subagents

If it is decided to turn an agent into a superagent, nothing special happens. When it is decided to create a superagent from existing (sub)agents. A function to automatically extract the agent activity template of the superagent should be called. The function would pull out the attributes and operations of each subagent that they have in common. A notification (reminder) that a subagent needs to have additional attributes not present in the superagent may need to occur[13]. Any changes that are now routed to the superagent should automatically be rerouted from the subagents in the diagram. The third possibility is that existing agents could be grouped such that one of them is set to be a superagent and the rest are subagents. In this event the superagent would need to have its agent activity template updated, much like in the aforementioned second possibility, from the common attributes and operations of the subagents. The original properties of the superagent and those added from the subagent(s) will need to be organized into activities by the user. A prompt should be issued to remind the user to reorganize the properties of the super agent. All attributes and operations of the superagent that exist in the subagents should be removed. The attributes and operations of the subagent then need to be reorganized into activities. A prompt (if necessary) should be issued to remind a user that

[13] This reminder may also help identify typos made in the subagents preventing a superagent from having a non-empty agent template.

a superagent cannot have an empty agent activity template and that each subagent must have its own attributes and operations that are not present in the superagent.

5.5.1.4 Composite and Component Agents

A user should be able to switch between the composite and components view of an agent at any point they desire.

There are three possibilities with composite and component agents. The first possibility is there is a composite agent that is decomposed down into its components. In this case a prompt is issued to the user for how many components there are and what there names will be. Since we already know what the incoming and outgoing changes to the composite agent are we can issue a prompt for both which incoming changes (actually the interface attribute associated with the incoming change) go to which agent, then the entire activity is assigned to the component. The last prompt necessary is for what changes (if any) do the components use to interact with each other. The user then needs a reminder the component interactions all require activities.

The second case is that several components are combined to create a composite object. In this case the agent activity template of the composite is populated from the agent templates of the components. Any interactions between the components are left out. When the components have the same activity it is only entered in the composite once. When the components have an activity that is the same except for the incoming interface attributes then only one activity will be entered in the composite, but both incoming interface attributes will be placed in the activity and the user will be prompted for the transformation law that exists. The user should then be prompted for the emergent attributes and operations of the composite. Lastly, the user should be asked if they wish to view the composite in the composite view or the decomposed view in the diagram.

The last case is that from a collection of agents one is decided to be a composite agent and the rest component agents. In that event the agent activity template of the composite agent needs to be altered to: show its attributes and operations are emergent and to include the attributes, and operations of the component agents (as aforementioned in creating a composite) and to denote which component agent contributed which attribute and operation. If a component and the composite agent have the same attribute or operation it is denoted as being contributed from the component not the composite. Lastly, the user should be asked if they wish to view the composite in the composite view or the decomposed view in the diagram.

5.5.2 Display Option

A display option should be included to allow the ABPM to be shown as is, or in any compressed view the user desires. Since the information is stored in the agent templates viewing the compressed agents will not lose any information. Specifically, the user should be able to choose the display method for any agent in the model. This can be done for various reasons such some large agents may need to be compressed in order to capture the entire system in a confined area (e.g an 8.5"x11" page), a user may only be interested in viewing the interactions, etc.

5.5.3 OOEM And ABPM Conversions

As established in chapter 4 it is possible to convert an OOEM to an ABPM and vice versa. We assume there already exists some sort of CASE tool which has implemented OOEM. The conversion process should be automated as much as possible to prevent human error. However at steps where there is the possibility of ambiguity or lack of information a user could be prompted to for the appropriate information. Each model should have its own window. As well, the source file used to create the second diagram should not be changed by the conversion process.

5.5.3.1 ABPM to OOEM Converter

For the sake of convenience we will assume the user has developed an ABPM that has passed a semantic integrity check. The starting point is the user has selected the option to convert the ABPM to an OOEM. The CASE tool should initialize a new OOEM window that is linked to the current ABPM. Using the ABPM to OOEM conversion process from chapter 4, we will discuss the conversion steps and highlight those that require user intervention.

Step 1. External Object Conversion. Every ABPM external agent becomes an OOEM external object.

This is a step that can be automated. The system can automatically create a new empty external object template for each external object. As well it can create the associated graphical construct for an external object and place it in the diagram.

Step 2. Internal Object Conversion. Every ABPM internal agent becomes an OOEM internal object

This is a step that can be automated. The system can automatically create a new empty internal object template for each external object. As well it can create the associated graphical construct for an internal object and place it in the diagram.

Step 3. Request and Response Identification. Outgoing interface attributes become requests. However if the last outgoing interface attribute is going to the same agent which initiated the activity in which the outgoing interface attribute is found the and the outgoing interface attribute used to initiate the activity became a request; the outgoing interface attribute becomes a response. This step may encounter duplicate changes, duplicates can be left out.

This step can be automated. The system can first eliminate duplicate changes. Then it can convert then to requests or responses as dictated. The requests and responses can then be automatically placed in the diagram.

Step 4. Interface Attribute Conversion. An incoming interface attribute to handle a change from an agent becomes an interface attribute to handle a request. This only applies to those incoming interface attributes of which the associated outgoing interface attribute became a request.

This step can be automated. By keeping track of the changes that became requests in the previous step the system can know which incoming interface attributes to convert to interface attributes. The interface attributes can the be automatically placed in the diagram.

Step 5. Internal Attribute Conversion. An ABPM internal attribute becomes an OOEM internal attribute. This step may generate duplicate internal attributes, duplicates can be left out.

This step can be automated. The system can first eliminate duplicate attributes. Then it can put the internal attributes into the object template. Then the internal attributes can automatically be placed in the diagram.

Step 6. Service Creation. Create a service to process every request. This step requires the modeler to create a service themselves since the service is not present in the process model.

The system can automatically place one of the operations that are candidates for a service in the diagram. The user can reminded that the services may not be correct, and that they may either need to use a different one of the candidates for a service or create a new service altogether for each request.

5.5.3.2 OOEM to ABPM Converter

For the sake of convenience we will assume the user has developed an OOEM that has passed a semantic integrity check. The starting point is the user has selected the option to convert the OOEM to an ABPM. The CASE tool should initialize a new ABPM window that is linked to the current OOEM. Using the OOEM to ABPM conversion process from chapter 4, we will discuss the conversion steps and highlight those that require user intervention.

Step 1. External Agent Conversion. Every OOEM external object becomes an external ABPM agent.

This is a step that can be automated. The system can automatically create a new empty external agent template for each external agent. As well it can create the associated graphical construct for an external agent and place it in the diagram.

Step 2. Internal Agent Conversion. Every OOEM internal object becomes an ABPM internal agent

This is a step that can be automated. The system can automatically create a new empty internal agent template for each internal agent. As well it can create the associated graphical construct for an internal agent and place it in the diagram.

Step 3. Outgoing Interface Attribute Identification. All requests and responses become outgoing interface attributes.

This is a step that can be automated. The system can do the conversion and then put them into the first activity of the agent template to temporarily store them.

Step 4. Incoming Interface Attribute Identification. All outgoing interface attributes identified in step 3 require an incoming interface attribute. As well when we create the incoming interface attribute we need to show the agents share a mutual attribute.

This step can be automated. The outgoing interface attributes can be dissected for the 'attribute changed' part of the incoming interface attribute, while the agent possessing the outgoing interface attribute is the 'agent doing the change' part. The incoming interface attribute can then be stored in the agent template of the agent that possesses it.

Step 5. Activity Creation. At this point we can create activities. Each time there is an incoming interface attribute without a transformation law requiring other incoming interface attributes to change to start the operations of the activity, it is the beginning of a new activity. We can create transformation and state laws for the incoming interface attributes using domain knowledge.

The system can automatically put each incoming interface attribute into a new activity. The user can then be prompted for any transformation laws that exist and what incoming interface attributes belong to the same activity.

Step 6. Outgoing Interface Attribute Assignment. We can now assign the outgoing interface attributes to the appropriate activity using domain knowledge. Duplicate assignments are allowed.

This is a manual process (unless there is only one activity in which case the assignment can be automatically done done), in which the user can be prompted for which activity the which outgoing interface attributes is altered in.

Step 7. Internal Attribute Identification. Every OOEM internal attribute becomes an ABPM internal attribute. We can the assign them to the appropriate activity using domain knowledge. Additional internal attributes may need to be created using domain knowledge. Duplication may occur

This step can be partially automated. The system can do the conversion for the existing OOEM internal attributes, however the user will have to be prompted for any extra attributes. The user also needs to be prompted for which activities the attributes are used in.

Step 8. Operation Identification. Create the operations that will be invoked to change the outgoing interface and internal attributes using domain knowledge. We also have to create transformation laws using domain knowledge that show if there are any restrictions on the operations of the activity.

It would drive a user insane if they were prompted for an operation for every outgoing interface and internal attribute. We suggest one reminder about the fact the user needs to create operations for every outgoing interface and internal attribute. At this point the diagram would be in a compressed view that has no operations. The user can then rely on the semantic checker (discussed in the next section) to ensure they have an operation for every outgoing and interface attribute.

Step 9. Resource Identification. In the event there are agents only have incoming interface attributes in all of their activities they are resources and need to be converted to the appropriate resource notation.

This is a step that can be automated. The system can check if there are any agents with that only have incoming interface attributes in its activities. If there are it can automatically convert the agent to a resource.

5.5.4 Semantic Checker

The requirement of a semantic checker is rather intuitive. Modern tools (for example Microsoft Word) provide facilities that perform error checking for the user. This makes the tool more useful to the user. As well a tool that can check whether an ABPM diagram is correct will help to reduce errors and lead to better models being created. How are errors introduced to an ABPM? Errors are introduced by violating the semantic integrity of the language, that is, by violating the rules. Hence a semantic checker will be a useful part of the tool to ensure the semantic integrity rules are followed. When a rule is violated the error can be highlighted for the user.

The rules (from chapter 3) and how they should be implemented in the system are below:

1. Every change in the system should be able to be traced back to an initial change from an external agent.

The incoming interface attribute for each activity can be back tracked to the outgoing interface attribute. The system can then backtrack the incoming interface attribute of the activity that the outgoing interface is a part of until an external agent is reached. If an external agent is not reached then there is an error.

2. Every agent must have at least one activity.

The system can check if the activity count for each agent is greater than or equal to one. If it is not, there is an error.

3. Every Activity must have at least one operation

The system can check if the operation count for each activity is greater than or equal to one. If it is not, there is an error.

4. Every resource must only have incoming interface attributes.

The system can check each entry in the resource template. If they are not incoming interface attributes (notationally in the form: agent doing the change::attribute changed), there is an error.

5. Every attribute is changed by one operation.

In the agent template every outgoing interface attribute and interface attribute should have one and only one operation. The system can check if this true, if not there is an error.

6. Every activity can only have one incoming interface attribute unless governed by a transformation law.

In the agent template if the transformation law associated with an incoming interface attribute is blank there should only be one incoming interface attribute for each activity. The system can check if this is true, if not there is an error.

7. Every outgoing interface attribute must have a corresponding interface attribute and vice versa.

Two checks occur here. For each outgoing interface attribute, there has to be a check in the activities of the agent it lists the on the 'agent being changed' part of the outgoing interface attribute that there is in fact an incoming interface attribute for it. If not there is an error. This check should keep track of the incoming interface attributes it found, if there are other incoming interface attributes not found in during the check then there is an error since those incoming interface attributes do not have an outgoing interface attribute associated with them.

8. Every composite agent must possess emergent attributes and operations not present in the component agents

The system can check if there additional attributes and operations not present in the components. If there are not any then there is an error

9. Every subagent must possess attributes and operations that are unique to the subagent and are not inherited from the superagent.

The system can check if there additional attributes and operations not present in the superagents. If there are not any then there is an error

5.6 Design Limitations

The first implementation limitation is that the proposed design guidelines use the notation developed in chapter 3. A user may wish to develop an entirely different notation for the constructs. Even if that is the case, the requirements developed previously will still hold. They merely need to be implemented using the new notation. The second limitation is that ABPM itself has not yet been rigorously tested, thus errors in ABPM will have propagated through the design. The third limitation is that the ABPM CASE tool design was based partially on the design of an existing OOEM CASE tool. This may have introduced flaws of its own since an OOEM lacks information relative to ABPM.

5.7 Summary

This chapter proposed design principles for ABPM as a CASE tool. It defined the constructs that needed to be added to an OOEM CASE tool. It also discussed user interactions with such a CASE tool. The last thing this chapter considered was possible design limitations.

6 Conclusions And Future Research

6.1 Thesis Summary

This thesis presented a new method for modeling organizational processes; Object-Oriented Activity-Based Process Modeling (ABPM). We first looked at business process modeling in general and why improvements were needed.

We then proceeded to further develop a modeling algorithm proposed by Wang (Wang 2002) into a graphical modeling grammar. We created the grammar by combining the constructs of the Ontology-Based Process Modeling (OBPM) algorithm with object oriented constructs. We then tied the new ABPM constructs to specific graphical constructs. The essential constructs in ABPM are agent, attribute, and operation.

We then developed a modeling process for using the graphical ABPM constructs. The modeling process was based partially on the work by Wang (Wang 2002) and partially on work by Wand and Woo (Wand and Woo, 2002). The modeling process is based upon rules for: model scope, agent identification, operation inclusion, attribute inclusion, attribute ownership, composite agents, sub-classification, and new activities. We also include a set of semantic integrity rules that can be used to check if a created ABPM is semantically correct.

Based upon their similar theoretical foundations and purposes we then delved into the relationship between Object-Oriented Enterprise Modeling (OOEM) and ABPM. We developed a means to convert from one model to another, noting along the way that

ABPM contains more information than OOEM which should be the case since ABPM is a more detailed view of organizational activity.

At this point we created design principles for the implementation of ABPM as a CASE tool. We discussed the functional and non-functional requirements of such a CASE tool. Next we proposed a possible development platform for ABPM. We also considered the possible limitations of the proposed design architecture.

6.2 Contributions

This thesis developed the OBPM algorithm into an objected graphical modeling language and process. The ABPM constructs have specific and well-defined semantics for real world business process representation.

A noteworthy contribution is the refinement of the change propagation algorithm which is based upon a set of ontologically derived rules to create a systematic process for modeling a business process. The strength of the algorithm is from its ontological real world foundations rather than programming or data design rules of thumb.

This thesis also makes a contribution by exploring the relationship of ABPM and OOEM. Both languages are designed to model a specific view of organizational activity, irrespective of how a later information system artifact will be built. By relating the two grammars using ontological foundations we can acquire greater understanding of an organization without losing information.

Our last contribution the development of a design principles for an ABPM CASE tool that is implementation independent means that no matter how one decides to implement ABPM if they follow our requirements they will be able to create a tool to fully support the business process model generation process.

6.3 Limitations And Future Research

The foundations of ABPM were established by previous research. However ABPM itself is a new grammar and methodology. It has been applied to very few cases and testing so far has been limited. It also needs to be tested beyond an academic setting and in the real world. This will allow for validation of the grammar and method.

We also make a simplifying assumption for our ABPM to OOEM conversion process. During step 3 (request and response identification), regarding how to identify responses, we assume that an object can never receive a response from an object that has never made a request to it. Future research can be done on the ontological nature of both requests and responses, so that a better form of the request and response identification step during the ABPM to OOEM conversion process can be developed.

Another possible limitation is that we talk about both analysis principles (e.g. operation, activity) and design principles (e.g. attributes) in the same grammar. This is due to the fact that we use ontological principles that have no business meaning, with a goal of representing processes. Therefore when using the grammar for analysis and/or design we may have to convert ontological concepts to business concepts. For example attributes may need to be converted to documents. Further research could be done to determine the extent of the conversions necessary, if any.

Other areas of future research could focus on:

- ABPM does not allow for new state variables. The model ABPM is based upon does not allow for creating and eliminating things. All changes are modeled as changes of attribute values. Hence ABPM does not allow for creating and eliminating resources. There are two possible responses to this observation. First we can analyze everything in the domain, if all resources have been identified beforehand, they cannot be changed if they do not exist. Second, ontologically nothing appears or disappears, we just change its name. Research could be undertaken to determine which would be the more appropriate manner to deal with new or disappearing state variables.
- Data is disregarded is OBPM and thus by extension in ABPM. However computerized information systems tend to primarily pass data. An ontologically based data modeling grammar and method that is related back to ABPM would fill this gap.
- Wand, Woo, and Jung (Wand, Woo, and Jung, 2000) proposed a means to convert from an OOEM to a logical design of an information system. It should be possible to convert from an ABPM to a logical design of an information system since we can convert an ABPM to an OOEM and then a logical design of an information system. The question arises is it possible for a direct conversion from ABPM to a

logical design? Another question also arises concerning if it is possible to create a logical directly design from an ABPM will it be the same as the logical design from an OOEM, and if they are different which one is better?

- Although we have proposed a set of design principles for a CASE tool, the question remains of if it is implemented will it actually be useable and useful to modelers?
- Research can be done to compare ease of use, understanding, and quality of ABPM models in relation to other business process modeling grammars.

Bibliography

Bosilj-Vukšić, V., & Hlupić V. (2001). Petri Nets and IDEF Diagrams: Applicability and Efficacy for Business Process Modeling. *Informatica 25*(1), 123-133.

CIO Magazine. (1997). *Anatomy of a Failure*. Retrieved August 29, 2004, from http://www.cio.com/archive/enterprise/111597_data.html

Grause, D.C., & Weinberg, G, M. (1989). *Exploring Requirements: Quality Before Design*. New York, NY: Dorset House Publishing.

Huckvale, T. & Ould, M. (1994). Process Modeling: Why, What, and How. In K.Spurr, P. Layzell, L. Jennison, & N. Richards (Eds.) *Software Assistance for Business Re-engineering* (pp. 81-97). West Sussex, England: John Wiley & Sons Ltd.

Hui, S. (1997). *An Object-Oriented Workflow Management System*. M.Sc. Dissertation. Faculty of Commerce and Business Administration, University of British Columbia.

I. Jacobson, Object-Oriented Software Engineering, Addison-Wesley, 1992

Introduction to Petri Nets. (2004). *Introduction to Petri Nets.* Retrieved April 22, 2004 from http://worldserver.oleane.com/adv/elstech/petrinet.htm

Jensen, K: *A Brief Introduction to Coloured Petri Nets.* In: E. Brinksma (ed.): Tools and Algorithms for the Construction and Analysis of Systems. Proceeding of the TACAS'97 Workshop, Enschede, The Netherlands 1997, Lecture Notes in Computer Science Vol. 1217, Springer-Verlag 1997, 203-207

Kemper, P. (2004, February 5). *Lecture on Petri-Nets.* Retrieved April 22, 2004, from http://www.iai.inf.tu-dresden.de/ms/lvbeschr/vwahl_petri.html

Kluge, W.: The Kicking Horse Pass Problem *Petri Net News Letters* No. 54, (1998), pp. 3-15

Knowledge Based Systems, Inc. (2000, June, 23). IDEF3 Method Report. Retrieved April 22, 2004, from www.idef.com/downloads/pdf/idef3_fn.pdf

Tan, W. (William Tan Khoon Lee) (1997). *A semantically-enhanced object-oriented CASE tool for enterprise modeling.* M.Sc. Dissertation. Faculty of Commerce and Business Administration, University of British Columbia.

Parsons, J. & Wand, Y. (1997). Using Objects for Systems Analysis. *Communications of the ACM, 40(12),* 104-110.

Petri Nets. (2004). *Petri Nets.* Retrieved April 22, 2004 from http://www.petrinets.info/

Romney, M. (1994, October). Business Process Re-engineering. The CPA Journal Online. Retrieved August 29, 2004, from http://www.nysscpa.org/cpajournal/old/16373954.htm

The Standish Group. (1994) *The Chaos Report.* Retrieved August 29, 2004, from http://www1.standishgroup.com//sample_research/chaos_1994_1.php

Value Based Management.net. (2004) *Business Process Reengineering*. Retrieved August, 29, 2004, from http://www.valuebasedmanagement.net/methods_bpr.html

Wand, Y. & Wang, R. (1996). Anchoring Data Quality Dimensions in Ontological Foundations. *Communications of the ACM, 39(11)* 86-95.

Wand, Y. & Weber, R. (1990). Mario Bunge's Ontology as a Formal Foundation for Information Systems Concepts. In Dorn, G. & Weingartner, P. (Eds.), *Studies in Bunge's Treatise on Basic Philosophy, the Poznan Studies in the Philosophy of the Sciences and the Humanities* (pp123-150). Rodopi, Amersterdam

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

Wand, Y. & Weber, R. (1995). On the deep structure of information systems. *Information Systems Journal, 5,* 203-223.

Wand, Y. & Weber, R. (2002). Research Commentary: Information Systems and Conceptual Modeling – A Research Agenda. *Information Systems Research,* 13(4), 363-376.

Wand Y. & Woo. C. (1999). Ontology-Based Rules for Object-Oriented Enterprise Modeling. Working paper. Faculty of Commerce and Business Administration, University of British Columbia.

Wand, Y., Woo, C., & Jung, D. (2000). Object-Oriented Modeling: From Enterprise to Logical Design. *Proceedings of the Tenth Annual Workshop on Information Technologies and Systems (WITS'00, December 9-10, Brisbane Australia),* 25-30.

Wang, Q. (2002). *A Proposal for a Process Modeling Methodology*. M.Sc. Dissertation. Faculty of Commerce and Business Administration, University of British Columbia.

Wart, S., Wand, Y. & Woo, C. (1993). *Object-Oriented Systems Analysis: An Introduction.* Faculty of Commerce and Business Administration, University of British Columbia.

Zhang, X. (1998). *The Visualization of Object-Oriented Enterprise Modeling.* M.Sc. Dissertation. Faculty of Commerce and Business Administration, University of British Columbia.

Zhao, H. (1995). *Object-Oriented Enterprise Modeling.* M.Sc. Dissertation. Faculty of Commerce and Business Administration, University of British Columbia.

Zhou, M and Zurawski R: *Introduction to Petri Nets in Flexible and Agile Automation.* In: M. Zhou(ed.): Petri Nets in Flexible and Agile Automation, Kluwer Academic Publishers, 1999, pp.1-23

Zimmerman, A. (2004) *Petri Nets.* Retrieved April 22, 2004, from http://pdv.cs.tu-berlin.de/~azi/petri.html

Appendix A – Step By Step Derivation Of The ABPM For The ACME Warehouse Management Case

The purpose of this appendix is to illustrate the development of an ABPM following the ABPM algorithm. We will use the ACME Warehouse Management Inc. case for this demonstration. The case will be italicized to minimize confusion with the ABPM process

At this point we are at step 1 of the algorithm: Identify the external agents

ACME Warehouse Management Inc. offers storage facilities and redistribution services (between their different warehouses) across the nation. A customer can request space in a particular warehouse, request items to be transferred to another warehouse, or request withdrawal of items from a particular warehouse (even for items not stored there).

For the purpose of this case, we only look at the activities involved in processing a withdrawal request. A customer contacts ACME headquarters to request a withdrawal.

The scope of the process has been defined as the activities involved in processing a withdrawal request at the ACME Warehouse Management facilities. Where do withdrawal requests come from? The customer. Hence the customer must be an external agent.

We then move onto step 2 of the algorithm: for each external agent identify the changes generated. Customer generates 2 external changes: *a customer contacts ACME headquarters to request a withdrawal,* and *the customer will come to the warehouse on* *the required date to pick up the items.* Thus, the in terms of our notation changes generated by Customer are 'withdrawal request' and 'arrival'

Step 3 of the algorithm: *for each change:*, means we will look at what happens due to withdrawal request and arrival separately. We will first analyze withdrawal request and then arrival since arrival appears later in the case and logically the customer should not arrive to pick up items until a request for the items has been made.

We are at step 3.1 Identify the agent or resource that was changed by withdrawal request *A customer contacts ACME headquarters to request a withdrawal. An office clerk checks whether the customer has the authority to withdraw the items.* This raises the question of was ACME headquarters or Office Clerk the agent changed by Customer? The answer is to look at what actually becomes unstable. Does ACME headquarters or the Office Clerk act next? Thus the answer is Office Clerk. This does raise the possibility however that there could be some sort of composite agent ACME headquarters of which Office Clerk is a component agent. This possibility will be discussed when we get to step 4.

We are at step 3.2. If a resource was changed identify the incoming interface attribute. Office Clerk is an agent since it performs an action.

We are at step 3.3 If the agent is an internal agent:, Office clerk is an internal agent since Office Clerk is in the domain of interest.

We are at step 3.3.1 If this is the first change to an agent, or the last activity of an agent has gone through a sequence of instability-change-stability create a new activity. This is the first change for Office Clerk so a new activity will be created.

We are at step 3.3.2: Identify the incoming interface attributes that were modified. Office Clerk needs some sort of interface attribute to handle the withdrawal request. According to our modeling grammar it will be Customer::Withdrawal Request.

We are at step 3.3.3: Identify any state laws that may restrict change. *An office clerk checks whether the customer has the authority to withdraw the items. The clerk then passes the withdrawal request to the warehouse where the customer wants to pick up the items and the office clerk will notify the customer as to the status of the request (approved, or declined due to lack of authority, no inventory, or no transportation).* And *Once the office clerk has recorded the items to be withdrawn, he or she forwards the request to the manager (foreman) of the warehouse.* There appear to be no state laws on the incoming interface attributes

We are at step 3.3.4 Identify any transformation laws that may exist for the incoming interface attributes. There do not appear to be any state laws restricting further change since the office clerk checks the customer authorization immediately upon receiving the withdrawal request.

We are at step 3.3.5: If an agent becomes unstable:, Office Clerk does become unstable because it immediately performs an action upon receiving the withdrawal request.

We are at step 3.3.5.1: Identify the operations that may occur. *An office clerk checks whether the customer has the authority to withdraw the items. The clerk then passes the withdrawal request to the warehouse where the customer wants to pick up the items* and the *office clerk will notify the customer as to the status of the request (approved, or declined due to lack of authority, no inventory, or no transportation).* And *Once the office clerk has recorded the items to be withdrawn, he or she forwards the request to the manager (foreman) of the warehouse. The warehouse manager is responsible for directing the redistribution of items between warehouses.*

The Office Clerk needs to perform an authorization status check of that customer and then needs to either contact the appropriate warehouse or notify the customer that they are refused due to not passing the authorization check. In terms of our modeling grammar notation the services performed are Authority Check and either Contact Warehouse or Notify About Status. The clerk recording items is actually part of the incoming request from Customer[14].

We are at step 3.3.5.2 Identify any transformation laws that may affect what operations occur. *An office clerk checks whether the customer has the authority to withdraw the items. The clerk then passes the withdrawal request to the warehouse where the customer wants to pick up the items* and t*he office clerk will notify the customer as to the status of*

[14] Since ABPM does not deal with data we do not need to indicate a form being filled out.

the request (approved, or declined due to lack of authority, no inventory, or no transportation). And *Once the office clerk has recorded the items to be withdrawn, he or she forwards the request to the manager (foreman) of the warehouse. The warehouse manager is responsible for directing the redistribution of items between warehouses.* Contact Warehouse and Notify About Status are mutually exclusive[15]. This means there is a transformation law restricting which operation will happen. Since it is freeform we can have the law read as Contact Warehouse and Notify About Status are mutually exclusive.

We are at step 3.3.5.3: Identify the internal attributes that will be affected. *An office clerk checks whether the customer has the authority to withdraw the items. The clerk then passes the withdrawal request to the warehouse where the customer wants to pick up the items and the office clerk will notify the customer as to the status of the request (approved, or declined due to lack of authority, no inventory, or no transportation).* And *Once the office clerk has recorded the items to be withdrawn, he or she forwards the request to the manager (foreman) of the warehouse. The warehouse manager is responsible for directing the redistribution of items between warehouses.*

The office clerk only uses its own internal information in the authority check operation, the other operations involve interaction with other agents. The notation according to our grammar for this information on customer authorization status will be Authorization Status.

[15] We assume the office clerk will not pass a withdrawal request to the warehouse when the customer has not passed an authorization check.

We are at step 3.3.5.4: Identify the outgoing interface attributes that were modified. *An office clerk checks whether the customer has the authority to withdraw the items. The clerk then passes the withdrawal request to the warehouse where the customer wants to pick up the items* and t*he office clerk will notify the customer as to the status of the request (approved, or declined due to lack of authority, no inventory, or no transportation).* And *Once the office clerk has recorded the items to be withdrawn, he or she forwards the request to the manager (foreman) of the warehouse. The warehouse manager is responsible for directing the redistribution of items between warehouses.* Contact Warehouse modifies an interface attribute that affects another agent; Warehouse Manager. Notify About Status modifies an interface attribute that affects Customer. According to our grammar the notation will be Withdrawal Request::Warehouse Manager and Order Status::Customer

We are at step 3.3.4.5: Repeat steps 3.3.4.1 to 3.3.4.4 until the agent becomes stable. Since Office Clerk performs no more actions as a direct result of the incoming external request it can at this point be considered stable.

Since we have reached a state where an agent is stable it is useful to show what we have developed. Figure A-1 shows where we currently are. Warehouse Manager is purposely left nebulous is this case since we do not know if it is an internal agent or an external agent, or a resource.

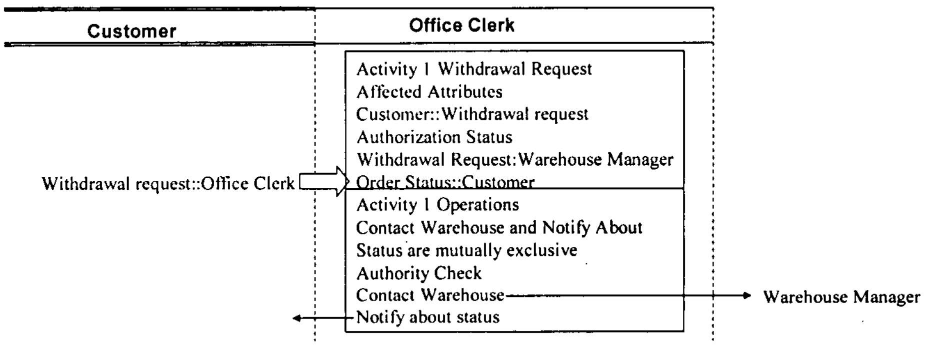

Customer
Office Clerk
Activity 1 Withdrawal Request
Affected Attributes
Customer::Withdrawal request
Authorization Status
Withdrawal Request:Warehouse Manager
Order Status::Customer
Withdrawal request::Office Clerk
Activity 1 Operations
Contact Warehouse and Notify About
Status are mutually exclusive
Authority Check
Contact Warehouse
Notify about status
Warehouse Manager

Figure A-1 Office Clerk Is Stable

We are at step 3.3.6: repeat step 3 for each outgoing interface attribute of an agent that was changed in step 3.3.5.4. The outgoing interface attributes from step 3.3.4.4 were Order Status::Customer and Withdrawal Request::Warehouse. For the sake of conciseness from this point forward we will use a tabular format wherever possible. *If the warehouse does not have the items or does not have enough quantity of the items, the warehouse manager will contact other warehouses for the requested items. If the items are located the warehouse manager will ask the planner to arrange for transportation for the requested items* and *the warehouse manager will notify the office clerk if the request can be fulfilled or not, and the reason. The office clerk will notify the customer as to the status of the request (approved, or declined due to lack of authority, no inventory, or no transportation)* and *The warehouse manager is responsible for directing the redistribution of items between warehouses. If the items are not all available in the warehouse, transport requests are issued. The warehouse manager fills out a redistribution form with the following information: items to be moved, place from which to take the items, warehouse to transport the items to, quantity to be moved, and the date by when the redistribution must be done. The warehouse manager forwards the form to the planner to organize the interwarehouse transportation of the items*

Change: Order Status::Customer	
Step	**Output**
3.1	Customer is an existing agent
3.2	Customer is an agent
3.3	Customer is an external agent

Table A-1 Step 3 for Order Status::Customer

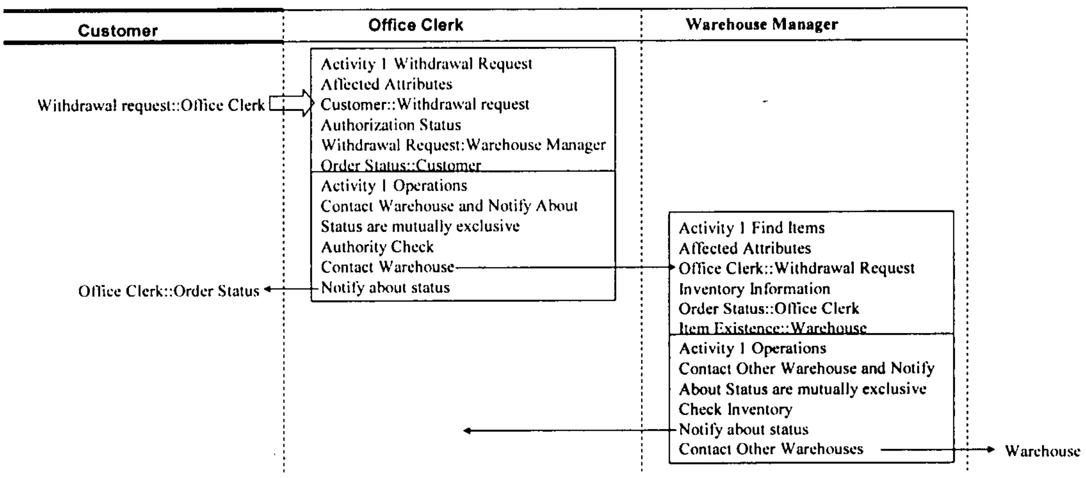
Change: Withdrawal Request::Warehouse Manager	
Step	**Output**
3.1	Warehouse Manager is a new agent
3.2	Warehouse Manager is an agent
3.3	Warehouse Manager is an internal agent
3.3.1	This is a new activity for Warehouse Manager
3.3.2	Office Clerk::Withdrawal Request
3.3.3	No state laws found
3.3.4	No transformation laws found
3.3.5	Warehouse Manager is unstable
3.3.5.1	Check Inventory Contact Other Warehouses Notify About Status
3.3.5.2	Contact Other Warehouse and Notify About Status are mutually exclusive
3.3.5.3	Inventory Information
3.3.5.4	Item Existence::Warehouse Order Status::Office Clerk
3.3.5.5	Warehouse Manager is currently stable

Table A-2 Step 3 For Withdrawal Request::Warehouse Manager

Since we have reached a state where an agent is stable it is useful to show what we have developed. Figure A-2 shows where we currently are. Planner is purposely left nebulous is this case since we do not know if it is an internal agent or an external agent. Warehouse is purposely left nebulous is this case since we do not know if it is an internal agent or an external agent[16].

From this point forward we will not show the ABPM after each step in order to be more concise. We will however show the full ABPM after all the changes have been addressed.

[16] Recall that even though many warehouses may be contacted we only need to show one since we are showing agents and not instances of agents in this diagram.



Customer
Office Clerk
Warehouse Manager
Withdrawal request::Office Clerk
Activity 1 Withdrawal Request
Affected Attributes
Customer::Withdrawal request
Authorization Status
Withdrawal Request:Warehouse Manager
Order Status::Customer
Activity 1 Operations
Contact Warehouse and Notify About
Status are mutually exclusive
Authority Check
Contact Warehouse
Notify about status
Office Clerk::Order Status
Activity 1 Find Items
Affected Attributes
Office Clerk::Withdrawal Request
Inventory Information
Order Status::Office Clerk
Item Existence::Warehouse
Activity 1 Operations
Contact Other Warehouse and Notify
About Status are mutually exclusive
Check Inventory
Notify about status
Contact Other Warehouses
Warehouse

Figure A-2 After Step 3 For Order Status::Customer And Withdrawal Request::Warehouse

We are at step 3.2.5: repeat step 3 for each outgoing interface attribute of an agent that was changed in step 3.3.5.4. The outgoing interface attributes from step 3.3.5.4 were Item Existence::Warehouse, Transport Form::Planner, and Order Status::Office Clerk

If the warehouse does not have the items or does not have enough quantity of the items, the warehouse manager will contact other warehouses for the requested items. If the items are located the warehouse manager will ask the planner to arrange for transportation for the requested items.

Change: Item Existence::Warehouse	
Step	**Output**
3.1	Warehouse is a new agent
3.2	Warehouse is an agent
3.3	Warehouse is an internal agent
3.3.1	This is a new activity for Warehouse
3.3.2	Warehouse Manager::Item Existence
3.3.3	No state laws found
3.3.4	No transformation laws found
3.3.5	Warehouse is unstable
3.3.5.1	Check Inventory Notify About Search Results
3.3.5.2	No transformation laws found
3.3.5.3	Inventory Information
3.3.5.4	Search Results::Warehouse Manager
3.3.5.5	Warehouse is currently stable

Table A-3 Step 3 For Item Existence::Warehouse

We are at step 3.2.5: repeat step 3 for each outgoing interface attribute of an agent that was changed in step 3.3.5.4. The outgoing interface attribute from step 3.3.5.4 was Search Results::Warehouse Manager.

If the warehouse does not have the items or does not have enough quantity of the items, the warehouse manager will contact other warehouses for the requested items. If the items are located the warehouse manager will ask the planner to arrange for transportation for the requested items and *the warehouse manager will notify the office clerk if the request can be fulfilled or not, and the reason.* And *If the items are not all available in the warehouse, transport requests are issued. The warehouse manager fills out a redistribution form with the following information: items to be moved, place from which to take the items, warehouse to transport the items to, quantity to be moved, and the date by when the redistribution must be done. The warehouse manager forwards the form to the planner to organize the interwarehouse transportation of the items.*

Change: Search Results::Warehouse Manager	
Step	**Output**
3.1	Warehouse Manager is an existing agent
3.2	Warehouse Manager is an agent
3.3	Warehouse Manager is an internal agent
3.3.1	This is a new activity for Warehouse Manager
3.3.2	Warehouse Manager::Search Results
3.3.3	No state laws found
3.3.4	No transformation laws found
3.3.5	Warehouse Manager is unstable
3.3.5.1	Contact Planner Notify About Status
3.3.5.2	Contact Planner and Notify About Status are mutually exclusive
3.3.5.3	No internal attributes found
3.3.5.4	Transport Form::Planner Order Status::Office Clerk
3.3.5.5	Warehouse is currently stable

A-4 Step 3 For Search Results::Warehouse Manager

We are at step 3.2.5: repeat step 3 for each outgoing interface attribute of an agent that was changed in step 3.3.5.4. The outgoing interface attributes from step 3.3.5.4 were Transport Form::Planner and Order Status::Office Clerk.

The planner's responsibility is to schedule the company's truck fleet to accommodate requests for transportation, taking into account the existing schedule of each truck and its capacity. The warehouse manager will be notified whether the transportation request can or cannot be satisfied and *the planner issues transport orders to truck drivers* and *The warehouse manager forwards the form to the planner to organize the interwarehouse transportation of the items. The items to be moved are marked as move-pending, and the planner initiates a plan to have the items at the appropriate warehouse at the given date. Once interwarehouse transport plans are finalized, transport requests are issued to the truck drivers.*

Change: Transport Form::Planner	
Step	**Output**
3.1	Planner is a new agent
3.2	Planner is an agent
3.3	Planner is an internal agent
3.3.1	This is a new activity for planner
3.3.2	Warehouse Manager::Transport Form
3.3.3	No state laws found
3.3.4	No transformation laws found
3.3.5	Planner is unstable
3.3.5.1	Mark Items Schedule Trucks Notify About Transport Issue Orders
3.3.5.2	No transformation laws found
3.3.5.3	Inventory Information Truck Information
3.3.5.4	Transport Schedule::Warehouse Manager Transport Orders::Truck Driver
3.3.5.5	Planner is currently stable

Table A-5 Step 3 For Transport Form::Planner

We are at step 3.2.5: repeat step 3 for each outgoing interface attribute of an agent that was changed in step 3.3.5.4. The outgoing interface attributes from step 3.3.5.4 were Transport Schedule::Warehouse Manager and Transport Orders::Truck Driver.

The planner's responsibility is to schedule the company's truck fleet to accommodate requests for transportation, taking into account the existing schedule of each truck and its capacity. The warehouse manager will be notified whether the transportation request can or cannot be satisfied.

The warehouse manager will notify the office clerk if the request can be fulfilled or not, and the reason.

Change: Transport Schedule::Warehouse Manager	
Step	**Output**
3.1	Warehouse Manager is an existing agent
3.2	Warehouse Manager is an agent
3.3	Warehouse Manager is an internal agent
3.3.1	This is a new activity for Warehouse Manager
3.3.2	Planner::Transport Schedule
3.3.3	No state laws found
3.3.4	No transformation laws found
3.3.5	Warehouse Manager is unstable
3.3.5.1	Notify About Status
3.3.5.2	No transformation laws found
3.3.5.3	No internal attributes found
3.3.5.4	Order Status::Office Clerk
3.3.5.5	Warehouse Manager is currently stable

Table A-6 Step 3 For Transport Schedule::Warehouse Manager

We are at step 3.2.5: repeat step 3 for each outgoing interface attribute of an agent that was changed in step 3.3.5.4. The outgoing interface attribute from step 3.3.5.4 was Order Status::Office Clerk.

The warehouse manager will notify the office clerk if the request can be fulfilled or not, and the reason. The office clerk will notify the customer as to the status of the request (approved, or declined due to lack of authority, no inventory, or no transportation).

Change: Order Status::Office Clerk	
Step	**Output**
3.1	Office Clerk is an existing agent
3.2	Office Clerk is an agent
3.3	Office Clerk is an internal agent
3.3.1	This is a new activity for Office Clerk
3.3.2	Warehouse::Order Status
3.3.3	No state laws found
3.3.4	No transformation laws found
3.3.5	Office Clerk is unstable
3.3.5.1	Notify About Status
3.3.5.2	No transformation laws found
3.3.5.3	No internal attributes found
3.3.5.4	Order Status::Customer
3.3.5.5	Office Clerk is currently stable

Table A-7 Step 3 For Order Status::Office Clerk

We are at step 3.2.5: repeat step 3 for each outgoing interface attribute of an agent that was changed in step 3.3.5.4. The outgoing interface attribute from step 3.3.5.4 was Order Status::Customer. This change was already dealt with. We can now return to the previous change and repeat step 3 for another of the outgoing interface attributes that was changed in step 3.3.5.4. In this case, we backtrack all the way to the change Transport Form::Planner which has another outgoing change of Transport Orders::Truck Driver.

The planner issues transport orders to truck drivers. After receiving a transport order, the truck driver informs the warehouse about the pickup of the items. The warehouse manager will make arrangements to have the items ready when the truck arrives. When the truck arrives at the warehouse the items are loaded. The truck driver then informs the next warehouse about the delivery. When the truck has arrived at the next warehouse, the items are unloaded. A warehouse worker finds space for the items and arranges to *have them moved to the allocated space. The worker updates the warehouse's inventory information. Truck drivers are required to report the status of the truck and the delivery to the planner after each step* and *The truck driver alerts the warehouse manager of the time he or she will be at the warehouse to pick up the items. The warehouse manager gives appropriate requests to the warehouse worker on the date of delivery to have the items ready for when the truck is expected. When the warehouse worker gets a request to fetch items, he or she, at the appropriate time, orders forklift operators to move the items to the loading platform. The forklift operators execute the internal warehouse operation. When the truck driver arrives, the driver notifies the warehouse worker to have the items loaded into the truck. The truck driver notifies the next warehouse manager when it is expected to arrive at the next warehouse. The number of items in the current warehouse decreases, and the transport request is marked as on transport.*

Change: Transport Orders::Truck Driver	
Step	**Output**
3.1	Truck Driver is a new agent
3.2	Truck Driver is an agent
3.3	Truck Driver is an internal agent
3.3.1	This is a new activity for Truck Driver
3.3.2	Planner::Transport Orders
3.3.3	No state laws found
3.3.4	No transformation laws found
3.3.5	Truck Driver is unstable
3.3.5.1	Notify About Pickup Report Truck Status Drive To Pickup Warehouse Notify Ready To Load Report Truck Status
3.3.5.2	No transformation laws found
3.3.5.3	Truck Status
3.3.5.4	Truck Status::Planner Pickup Notification::Warehouse Manager Ready To Load::Warehouse Worker Truck Status::Planner
3.3.5.5	Truck Driver is currently stable

Table A-8 Step 3 For Transport Orders::Truck Driver

We are at step 3.2.5: repeat step 3 for each outgoing interface attribute of an agent that was changed in step 3.3.5.4. The outgoing interface attributes from step 3.3.5.4 were Pickup Notification::Warehouse Manager, Truck Status::Planner, Ready To Load::Warehouse Worker, and Truck Status::Planner.

The planner issues transport orders to truck drivers. After receiving a transport order, the truck driver informs the warehouse about the pickup of the items. The warehouse manager will make arrangements to have the items ready when the truck arrives. When the truck arrives at the warehouse the items are loaded and *The truck driver alerts the warehouse manager of the time he or she will be at the warehouse to pick up the items. The warehouse manager gives appropriate requests to the warehouse worker on the date of delivery to have the items ready for when the truck is expected.*

Change: Pickup Notification::Warehouse Manager	
Step	**Output**
3.1	Warehouse Manager is an existing agent
3.2	Warehouse Manager is an agent
3.3	Warehouse Manager is an internal agent
3.3.1	This is a new activity for Warehouse Manager
3.3.2	Truck Driver::Pickup Notification
3.3.3	No state laws found
3.3.4	No transformation laws found
3.3.5	Warehouse Manager is unstable
3.3.5.1	Notify to Ready Items
3.3.5.2	No transformation laws found
3.3.5.3	No internal attributes found
3.3.5.4	Ready Items::Warehouse Worker
3.3.5.5	Warehouse Manager is currently stable

Table A-9 Step 3 For Pickup Notification::Warehouse Manager

We are at step 3.2.5: repeat step 3 for each outgoing interface attribute of an agent that was changed in step 3.3.5.4. The outgoing interface attribute from step 3.3.5.4 was Ready Items::Warehouse Worker.

The warehouse manager gives appropriate requests to the warehouse worker on the date of delivery to have the items ready for when the truck is expected. When the warehouse worker gets a request to fetch items, he or she, at the appropriate time, orders forklift operators to move the items to the loading platform. The forklift operators execute the internal warehouse operation.

Change: Ready Items::Warehouse Worker	
Step	**Output**
3.1	Warehouse Worker is a new agent
3.2	Warehouse Worker is an agent
3.3	Warehouse Worker is an internal agent
3.3.1	This is a new activity for Warehouse Worker
3.3.2	Warehouse Manager::Ready Items
3.3.3	No state laws found
3.3.4	No transformation laws found
3.3.5	Warehouse Worker is unstable
3.3.5.1	Issue Move Item Orders
3.3.5.2	No transformation laws found
3.3.5.3	No internal attributes found
3.3.5.4	Move Items::Forklift Operator
3.3.5.5	Warehouse Worker is currently stable

Table A-10 Step 3 For Ready Items::Warehouse Worker

We are at step 3.2.5: repeat step 3 for each outgoing interface attribute of an agent that was changed in step 3.3.5.4. The outgoing interface attribute from step 3.3.5.4 was Move Items::Forklift Operator

Change: Move Items::Forklift Operator	
Step	**Output**
3.1	Forklift Operator is a new agent
3.2	Forklift Operator is an agent
3.3	Forklift Operator is an internal agent
3.3.1	This is a new activity for Forklift Operator
3.3.2	Warehouse Worker::Move Items
3.3.3	No state laws found
3.3.4	No transformation laws found
3.3.5	Forklift Operator is unstable
3.3.5.1	Move Items
3.3.5.2	No transformation laws found
3.3.5.3	Inventory Information
3.3.5.4	No Outgoing Interface Attributes found
3.3.5.5	Forklift Operator is currently stable

Table A-11 Step 3 For Move Items::Forklift Operator

We are at step 3.2.5: repeat step 3 for each outgoing interface attribute of an agent that was changed in step 3.3.5.4. There are no outgoing interface attributes from step 3.3.5.4 We can now return to the previous change and repeat step 3 for another of the outgoing interface attributes that was changed in step 3.3.5.4. In this case, we backtrack to the change Transport Orders::Truck Driver which has another outgoing change of Truck Status::Planner.

The planner issues transport orders to truck drivers. After receiving a transport order, the truck driver informs the warehouse about the pickup of the items. The warehouse manager will make arrangements to have the items ready when the truck arrives. When the truck arrives at the warehouse the items are loaded. The truck driver then informs the next warehouse about the delivery. When the truck has arrived at the next warehouse, the items are unloaded. A warehouse worker finds space for the items and arranges to have them moved to the allocated space. The worker updates the warehouse's inventory information. Truck drivers are required to report the status of the truck and the delivery to the planner after each step and *The transportation time is recorded and stored.*

Change: Truck Status::Planner	
Step	**Output**
3.1	Planner is an existing agent
3.2	Planner is an agent
3.3	Planner is an internal agent
3.3.1	This is a new activity for Planner
3.3.2	Truck Driver::Truck Status
3.3.3	No state laws found
3.3.4	No transformation laws found
3.3.5	Planner is unstable
3.3.5.1	Record Time
3.3.5.2	No transformation laws found
3.3.5.3	Transport Information
3.3.5.4	No outgoing interface attributes found
3.3.5.5	Planner is currently stable

Table A-12 Step 3 For Truck Status::Planner

We are at step 3.2.5: repeat step 3 for each outgoing interface attribute of an agent that was changed in step 3.3.5.4. There are no outgoing interface attributes from step 3.3.5.4 We can now return to the previous change and repeat step 3 for another of the outgoing interface attributes that was changed in step 3.3.5.4. In this case, we backtrack to the change Transport Orders::Truck Driver which has another outgoing change of Ready To Load::Warehouse Worker.

After receiving a transport order, the truck driver informs the warehouse about the pickup of the items. The warehouse manager will make arrangements to have the items ready when the truck arrives. When the truck arrives at the warehouse the items are loaded. The truck driver then informs the next warehouse about the delivery and *When the truck driver arrives, the driver notifies the warehouse worker to have the items loaded into the truck. The truck driver notifies the next warehouse manager when it is expected to arrive at the next warehouse. The number of items in the current warehouse decreases, and the transport request is marked as on transport.*

Change: Ready To Load::Warehouse Worker	
Step	**Output**
3.1	Warehouse Worker is an existing agent
3.2	Warehouse Worker is an agent
3.3	Warehouse Worker is an internal agent
3.3.1	This is a new activity for Warehouse Worker
3.3.2	Truck Driver::Ready To Load
3.3.3	No state laws found
3.3.4	No transformation laws found
3.3.5	Warehouse Worker is unstable
3.3.5.1	Load Truck Mark As On Transport
3.3.5.2	No transformation laws found
3.3.5.3	Inventory Information
3.3.5.4	On Transport::Truck Driver
3.3.5.5	Warehouse Worker is currently stable

Table A-13 Step 3 For Ready To Load::Warehouse Worker

We are at step 3.2.5: repeat step 3 for each outgoing interface attribute of an agent that was changed in step 3.3.5.4. The outgoing interface attribute from step 3.3.5.4 was Loaded::Truck Driver.

When the truck arrives at the warehouse the items are loaded. The truck driver then informs the next warehouse about the delivery. When the truck has arrived at the next warehouse, the items are unloaded. A warehouse worker finds space for the items and arranges to have them moved to the allocated space. The worker updates the warehouse's inventory information. Truck drivers are required to report the status of the truck and the delivery to the planner after each step and The truck driver notifies the next warehouse manager when it is expected to arrive at the next warehouse. The number of items in the current warehouse decreases, and the transport request is marked as on transport. And *When the truck has arrived at the next warehouse, the truck driver notifies the warehouse worker to unload the items. The truck driver signs off the job.*

Change: On Transport::Truck Driver	
Step	**Output**
3.1	Truck Driver is an existing agent
3.2	Truck Driver is an agent
3.3	Truck Driver is an internal agent
3.3.1	This is a new activity for Truck Driver
3.3.2	Warehouse::On Transport
3.3.3	No state laws found
3.3.4	No transformation laws found
3.3.5	Truck Driver is unstable
3.3.5.1	Notify About Dropoff Report Truck Status Drive To Dropoff Notify Ready To Unload Report Truck Status
3.3.5.2	No transformation laws found
3.3.5.3	Truck Status
3.3.5.4	Dropoff Notification::Warehouse Manager Truck Status::Planner Ready To Unload::Warehouse Worker Truck Status::Planner
3.3.5.5	Truck Driver is currently stable

Table A-14 Step 3 For On Transport::Truck Driver

We are at step 3.2.5: repeat step 3 for each outgoing interface attribute of an agent that was changed in step 3.3.5.4. The outgoing interface attributes from step 3.3.5.4 were Dropoff Notification::Warehouse Manager, Truck Status::Planner, Ready To Unload::Warehouse Worker, and Truck Status::Planner.

The truck driver then informs the next warehouse about the delivery. When the truck has arrived at the next warehouse, the items are unloaded and *The truck driver notifies the next warehouse manager when it is expected to arrive at the next warehouse. The number of items in the current warehouse decreases, and the transport request is marked as on transport.*

Change: Dropoff Notification::Warehouse Manager	
Step	**Output**
3.1	Warehouse Manager is an existing agent
3.2	Warehouse Manager is an agent
3.3	Warehouse Manager is an internal agent
3.3.1	This is a new activity for Warehouse Manager
3.3.2	Truck Driver::Dropoff Notification
3.3.3	No state laws found
3.3.4	No transformation laws found
3.3.5	Warehouse Manager is unstable[17]
3.3.5.1	Notify About Receiving
3.3.5.2	No transformation laws found
3.3.5.3	No internal attributes found
3.3.5.4	Ready Receiving::Warehouse Worker
3.3.5.5	Warehouse Manager is currently stable

Table A-15 Step 3 For Dropoff Notification::Warehouse Manager

We are at step 3.2.5: repeat step 3 for each outgoing interface attribute of an agent that was changed in step 3.3.5.4. The outgoing interface attribute from step 3.3.5.4 was Ready Receiving::Warehouse Worker

[17] On the assumption the Warehouse Manager will let the Warehouse Worker know a dropoff is coming, much like how the manager let the worker know a pickup was coming.

A warehouse worker finds space for the items and arranges to have them moved to the allocated space.

Change: Ready Receiving::Warehouse Worker	
Step	**Output**
3.1	Warehouse Worker is an existing agent
3.2	Warehouse Worker is an agent
3.3	Warehouse Worker is an internal agent
3.3.1	This is a new activity for Warehouse Worker
3.3.2	Warehouse Manager::Ready Receiving
3.3.3	No state laws found
3.3.4	No transformation laws found
3.3.5	Warehouse Worker is unstable[18]
3.3.5.1	Find Space Arrange Movement
3.3.5.2	No transformation laws found
3.3.5.3	Warehouse Information Movement Schedule
3.3.5.4	No outgoing interface attributes found
3.3.5.5	Warehouse Worker is currently stable

Table A-16 Step 3 For Ready Receiving::Warehouse Worker

We are at step 3.2.5: repeat step 3 for each outgoing interface attribute of an agent that was changed in step 3.3.5.4. There are no outgoing interface attributes from step 3.3.5.4 We can now return to the previous change and repeat step 3 for another of the outgoing interface attributes that was changed in step 3.3.5.4. In this case, we backtrack to the change On Transport::Truck Driver which has another outgoing change of Ready To Unload::Warehouse Worker.

When the truck has arrived at the next warehouse, the items are unloaded. A warehouse worker finds space for the items and arranges to have them moved to the allocated space. The worker updates the warehouse's inventory information and *When the truck has arrived at the next warehouse, the truck driver notifies the warehouse worker to unload*

[18] On the assumption the Warehouse Worker will attempt to have the warehouse prepared before the truck arrives

the items. The truck driver signs off the job. The warehouse workers receive the items and determine a place for them in the warehouse. Forklift operators are told to move the items to the new place in the warehouse. When the truck driver confirms the delivery of the items, the records are updated to reflect the new place for the items. The transportation time is recorded and stored. The redistribution and interwarehouse transport request are marked as performed. The warehouse worker fills in an inventory update form and sends it to the warehouse manager for confirmation and update of the inventory database.

Change: Ready To Unload::Warehouse Worker	
Step	**Output**
3.1	Warehouse Worker is an existing agent
3.2	Warehouse Worker is an agent
3.3	Warehouse Worker is an internal agent
3.3.1	This is a new activity for Warehouse Worker
3.3.2	Truck Driver::Ready To Unload
3.3.3	No state laws found
3.3.4	No transformation laws found
3.3.5	Warehouse Worker is unstable
3.3.5.1	Receive Items Determine Place Notify Forklift Operators Ask For Confirmation
3.3.5.2	No transformation laws found
3.3.5.3	Transport Form Warehouse Information
3.3.5.4	Move Items::Forklift Operator Confirmation Needed::Truck Driver
3.3.5.5	Warehouse Worker is currently stable

Table A-17 Step 3 For Ready To Unload::Warehouse Worker

We are at step 3.2.5: repeat step 3 for each outgoing interface attribute of an agent that was changed in step 3.3.5.4. The outgoing interface attributes from step 3.3.5.4 were Move Items::Forklift Operator and Update Form::Warehouse Manager. Move Items::Forklift Operator has already been dealt with.

The warehouse worker fills in an inventory update form and sends it to the warehouse manager for confirmation and update of the inventory database.

Change: Update Form::Warehouse Manager	
Step	**Output**
3.1	Warehouse Manager is an existing agent
3.2	Warehouse Manager is an agent
3.3	Warehouse Manager is an internal agent
3.3.1	This is a new activity for Warehouse Manager
3.3.2	Warehouse Worker::Update Form
3.3.3	No state laws found
3.3.4	No transformation laws found
3.3.5	Warehouse Manager is unstable
3.3.5.1	Update Inventory Database
3.3.5.2	No transformation laws found
3.3.5.3	Inventory Database
3.3.5.4	No outgoing interface attributes found
3.3.5.5	Warehouse Manager is currently stable

Table A-18 Step 3 For Update Form::Warehouse Manager

We are at step 3.2.5: repeat step 3 for each outgoing interface attribute of an agent that was changed in step 3.3.5.4. There are no outgoing interface attributes from step 3.3.5.4 We can now return to the previous change and repeat step 3 for another of the outgoing interface attributes that was changed in step 3.3.5.4. At this point we have dealt with all the changes caused directly or indirectly by the external change Withdrawal Request::Office Clerk. We now move onto the changes caused by the external change Arrival::Employee

The customer will come to the warehouse on the required date to pick up the items. A warehouse employee will check all the necessary documents and will deliver the items with an accompanying documentation to the customer and *When the customer has* *fetched the items the warehouse workers mark the withdrawal as ready. The items are removed (decreased) from the information system.*

Change: Arrival::Employee	
Step	**Output**
3.1	Employee is a new agent
3.2	Employee is an agent
3.3	Employee is an internal agent
3.3.1	This is a new activity for Employee
3.3.2	Customer::Arrival
3.3.3	No state laws found
3.3.4	No transformation laws found
3.3.5	Employee is unstable
3.3.5.1	Check Documents Fetch Items Deliver Items And Documentation Remove Items from the System
3.3.5.2	No transformation laws found
3.3.5.3	Customer Information Inventory Information Inventory Database
3.3.5.4	Items And Documentation::Customer
3.3.5.5	Employee is currently stable

Table A-19 Step 3 For Arrival::Employee

We are at step 3.2.5: repeat step 3 for each outgoing interface attribute of an agent that was changed in step 3.3.5.4. The outgoing interface attribute from step 3.3.5.4 was Items And Documentation::Customer.

Change: Items And Documentation::Customer	
Step	**Output**
3.1	Customer is an existing agent
3.2	Customer is an agent
3.3	Customer is an external agent

Table A-20 Step 3 For Items And Documentation::Customer

At this point we have handled all changes directly and indirectly created due to all external agents. The following tree shows the changes that were generated. Always traversing the tree left shows the order in which the changes were handled. The changes with a box around them are duplicate changes, and the number is used to match the duplicates.

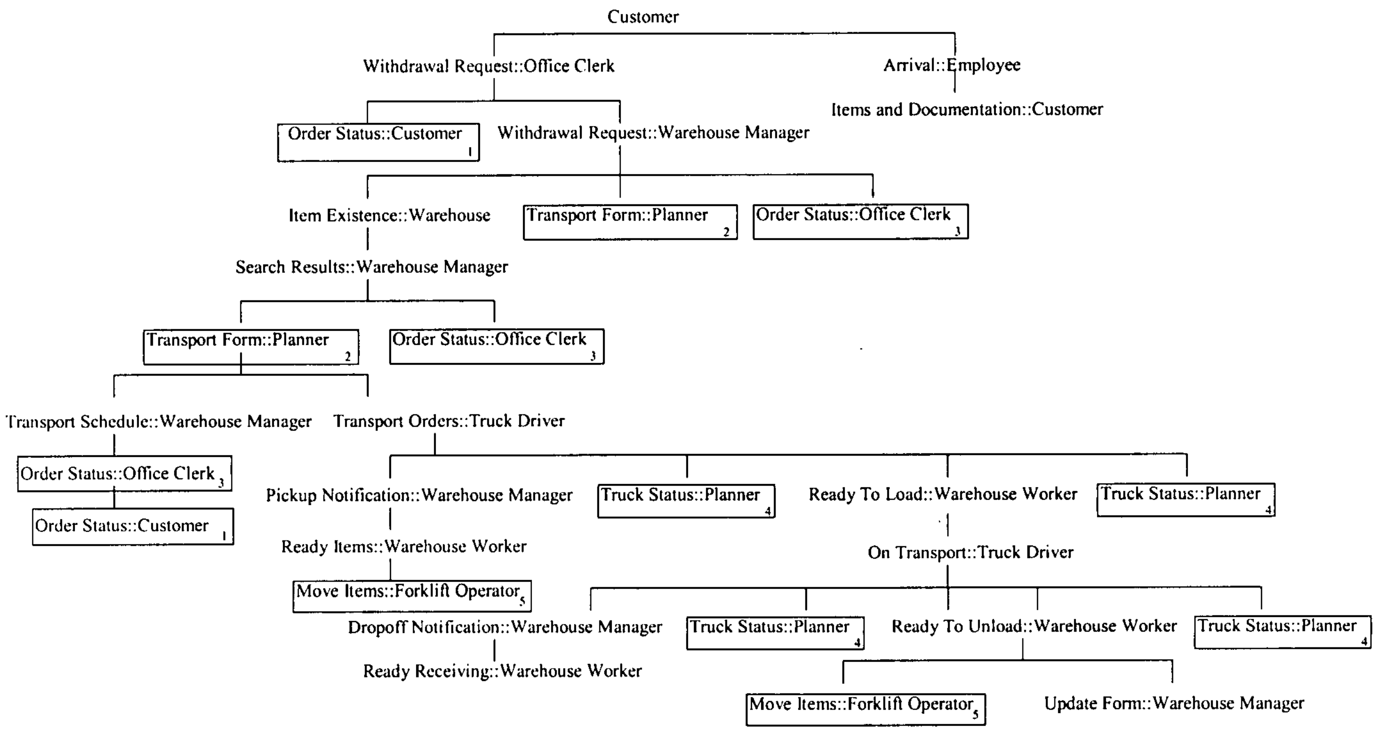

Customer
Withdrawal Request::Office Clerk
Arrival::Employee
Items and Documentation::Customer
Order Status::Customer 1
Withdrawal Request::Warehouse Manager
Item Existence::Warehouse
Transport Form::Planner 2
Order Status::Office Clerk 3
Search Results::Warehouse Manager
Transport Form::Planner 2
Order Status::Office Clerk 3
Transport Schedule::Warehouse Manager
Transport Orders::Truck Driver
Order Status::Office Clerk 3
Order Status::Customer 1
Pickup Notification::Warehouse Manager
Truck Status::Planner 4
Ready To Load::Warehouse Worker
Truck Status::Planner 4
Ready Items::Warehouse Worker
On Transport::Truck Driver
Move Items::Forklift Operator 5
Dropoff Notification::Warehouse Manager
Truck Status::Planner 4
Ready To Unload::Warehouse Worker
Truck Status::Planner 4
Ready Receiving::Warehouse Worker
Move Items::Forklift Operator 5
Update Form::Warehouse Manager

Figure A-3 A Tree Showing All Of The Changes

Figure A-4 shows the solution we have developed at this point

We are now at step 4: If needed identify super and subagents using the internal agents. If we stop and consider for a second what it means to be an employee of the warehouse, we can see that it is in fact a super agent with the subagents warehouse worker, warehouse manager, and forklift operator. That is, these agents are all employees that do work at something called a warehouse (More about warehouse will be said in step 5 below). Figure A-5 shows where we currently are.

We are now at step 5: If needed identify composite and component agents using the internal agents. We noted above in two places where the possibility of composite agents was suggested.

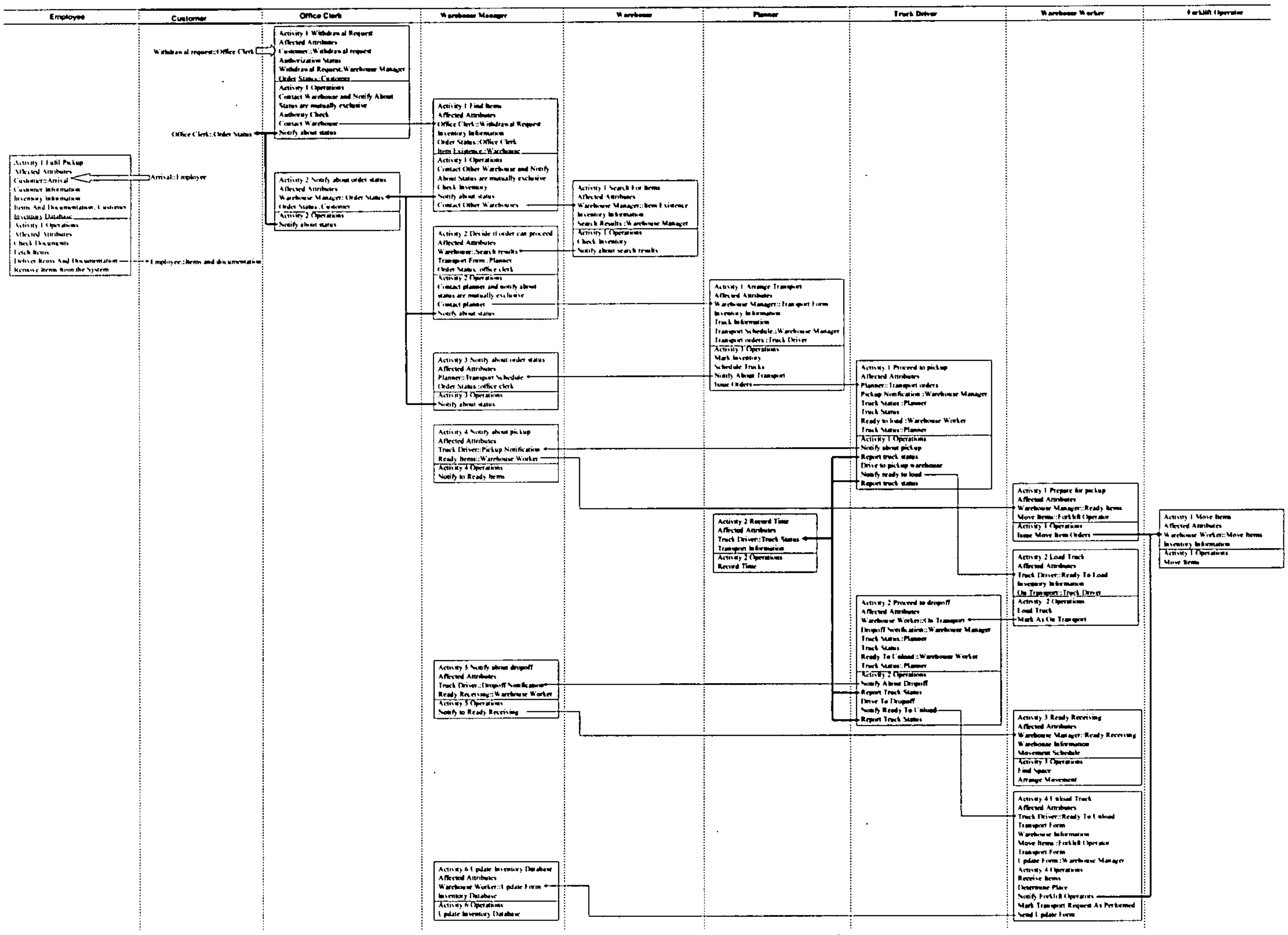


Figure A-4 Solution to the ACME Warehouse Case

There is no need to create a composite agent for ACME headquarters. There is only one agent that could be part of such a composition, Office Clerk, and thus no further information would be represented by creating a composite agent.

We can create a Warehouse composite agent. The case mentions a warehouse manager, a warehouse worker, and a forklift operator that perform activities at the warehouse, these can become component agents. The attributes and operations of the already existing Warehouse agent can become the emergent attributes and operations of the new Warehouse composite agent.

Figure A-6 shows the composite agent warehouse in the decomposed view. When we collapse the decomposed warehouse agent into its aggregate form we get figure A-7.

We have now gone through all the steps and are finished.

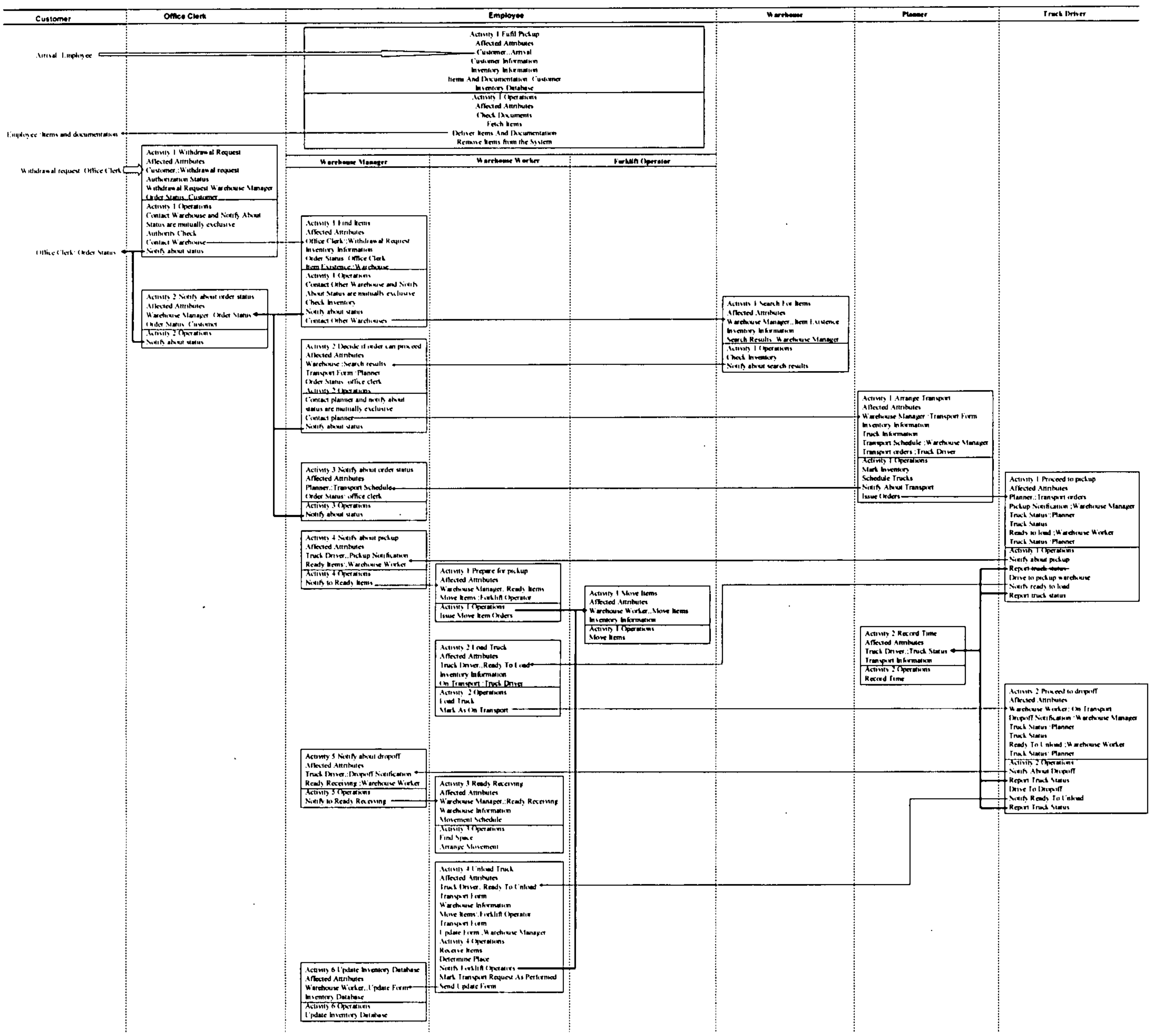


Figure A-5 Creating An Employee Superagent

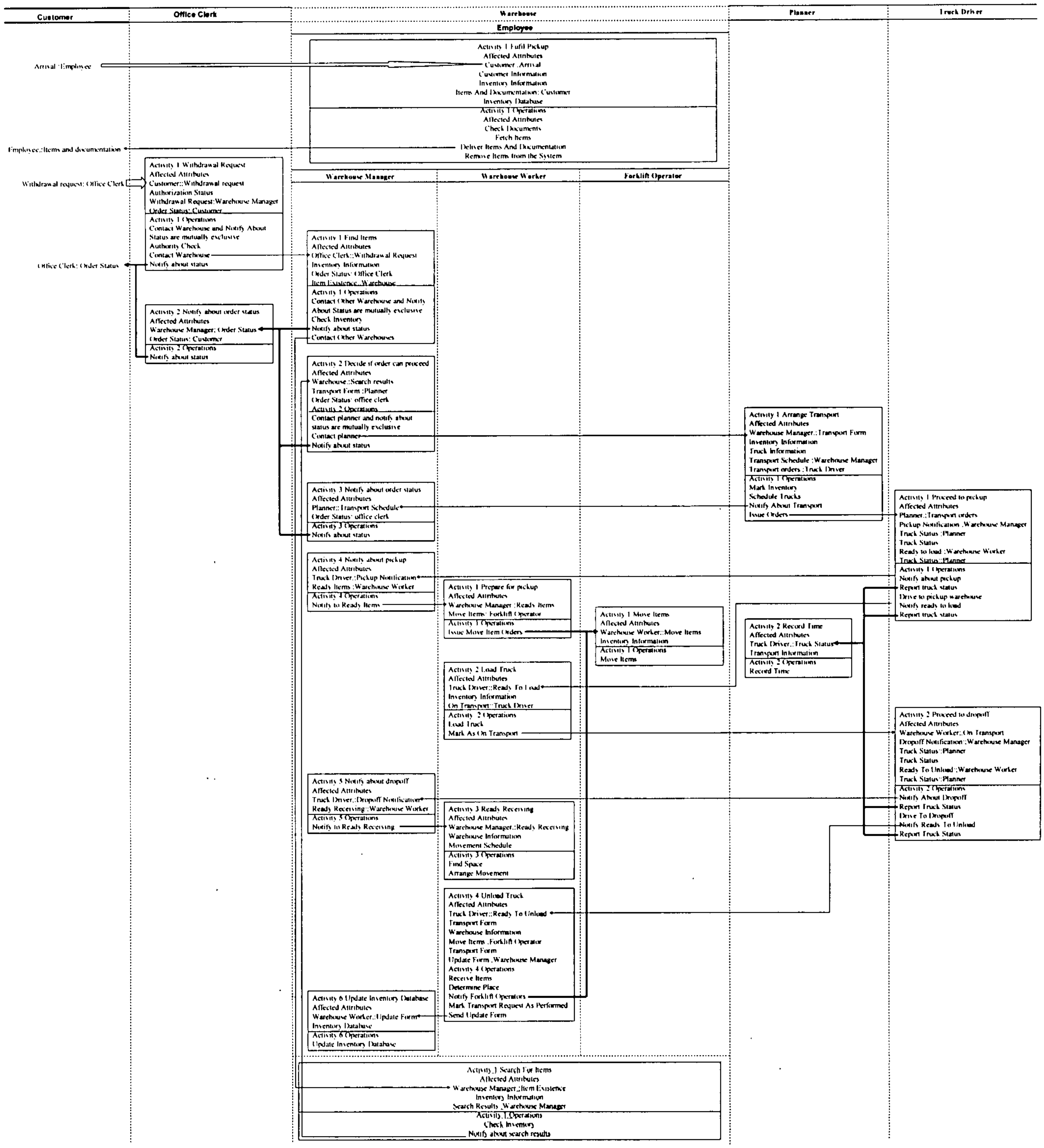


Figure A-6 The Decomposition of Warehouse

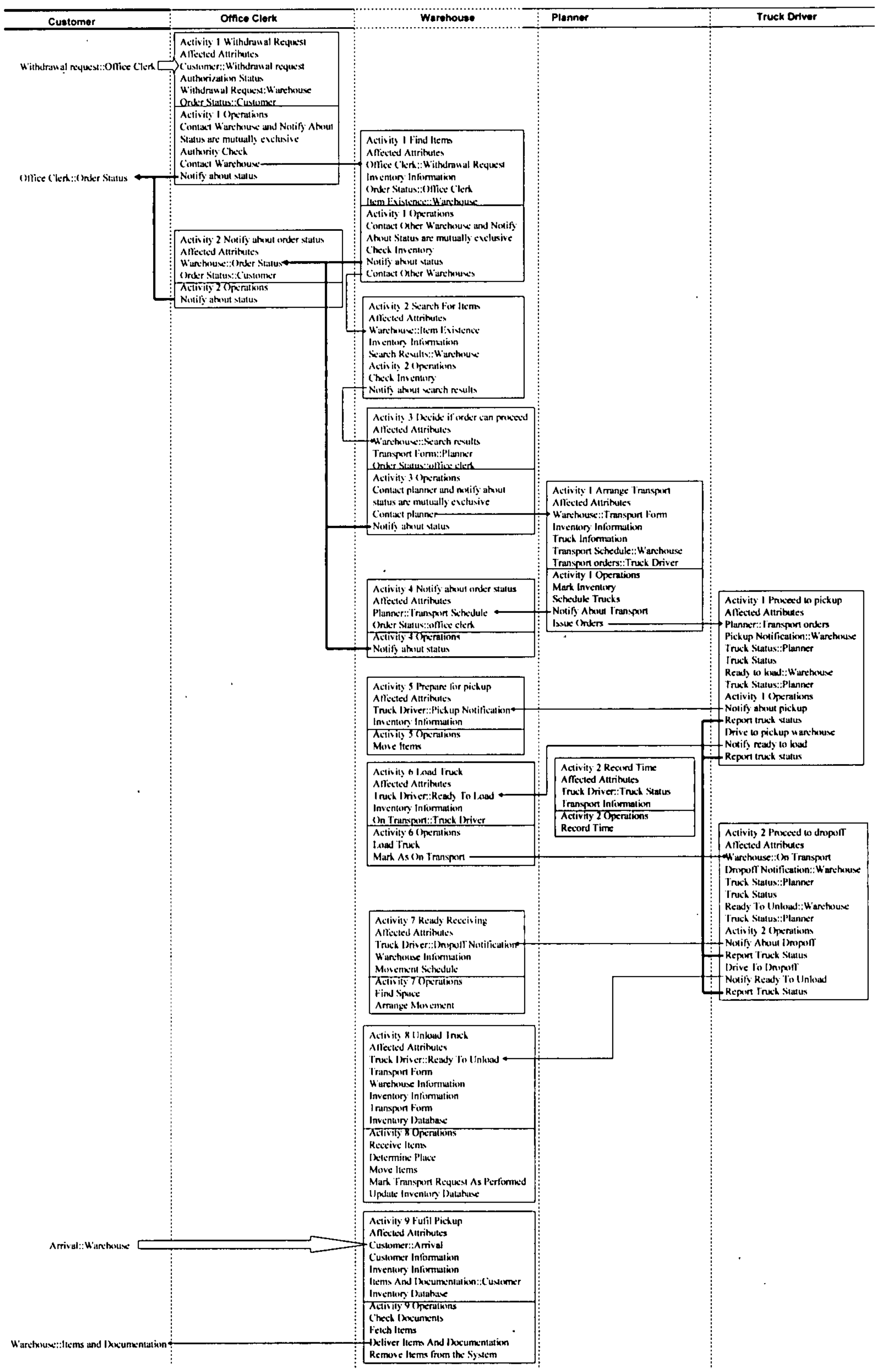


Figure A-7Warehouse is a composite agent

Appendix B – Agent Templates For The ACME Warehouse Management Case

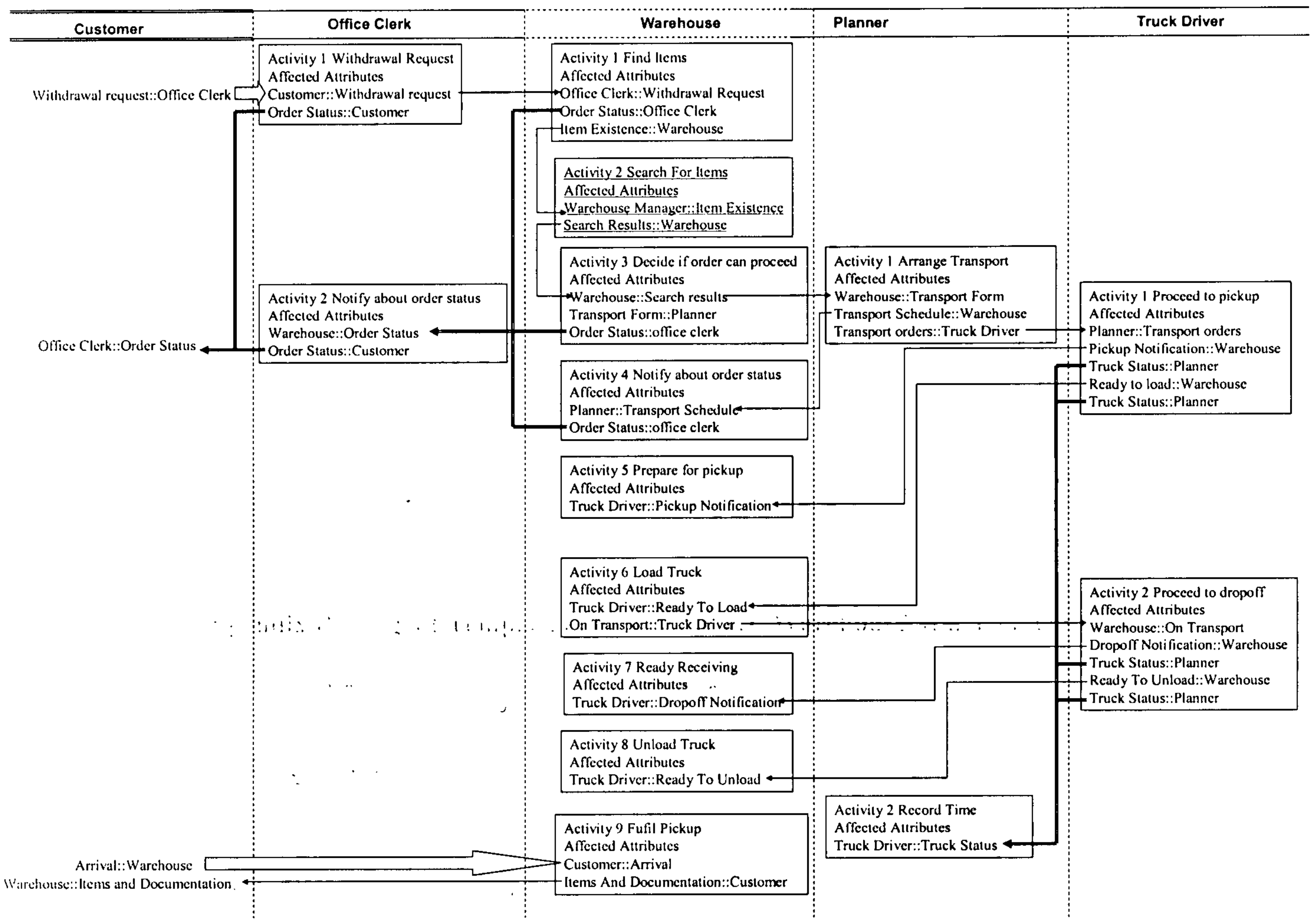

Customer
Office Clerk
Warehouse
Planner
Truck Driver
Withdrawal request::Office Clerk
Activity 1 Withdrawal Request
Affected Attributes
Customer::Withdrawal request
Order Status::Customer
Activity 1 Find Items
Affected Attributes
Office Clerk::Withdrawal Request
Order Status::Office Clerk
Item Existence::Warehouse
Activity 2 Search For Items
Affected Attributes
Warehouse Manager::Item Existence
Search Results::Warehouse
Activity 3 Decide if order can proceed
Affected Attributes
Warehouse::Search results
Transport Form::Planner
Order Status::office clerk
Activity 1 Arrange Transport
Affected Attributes
Warehouse::Transport Form
Transport Schedule::Warehouse
Transport orders::Truck Driver
Activity 1 Proceed to pickup
Affected Attributes
Planner::Transport orders
Pickup Notification::Warehouse
Truck Status::Planner
Ready to load::Warehouse
Truck Status::Planner
Activity 2 Notify about order status
Affected Attributes
Warehouse::Order Status
Order Status::Customer
Office Clerk::Order Status
Activity 4 Notify about order status
Affected Attributes
Planner::Transport Schedule
Order Status::office clerk
Activity 5 Prepare for pickup
Affected Attributes
Truck Driver::Pickup Notification
Activity 6 Load Truck
Affected Attributes
Truck Driver::Ready To Load
On Transport::Truck Driver
Activity 2 Proceed to dropoff
Affected Attributes
Warehouse::On Transport
Dropoff Notification::Warehouse
Truck Status::Planner
Ready To Unload::Warehouse
Truck Status::Planner
Activity 7 Ready Receiving
Affected Attributes
Truck Driver::Dropoff Notification
Activity 8 Unload Truck
Affected Attributes
Truck Driver::Ready To Unload
Activity 2 Record Time
Affected Attributes
Truck Driver::Truck Status
Activity 9 Fufil Pickup
Affected Attributes
Customer::Arrival
Items And Documentation::Customer
Arrival::Warehouse
Warehouse::Items and Documentation

Figure B-1 Compressed Agent Interaction Diagram

Forklift Operator					
Superagent: Employee					
	Attributes			Operations	
Activity	State Law	Interface Attributes	Internal Attributes	Transformation Law	Operation
1		Warehouse Worker::Move Items			
			Inventory Information		Move Items

Figure B-2 Forklift Operator Agent Template

Office Clerk					
	Attributes			Operations	
Activity	State Law	Interface Attributes	Internal Attributes	Transformation Law	Operation
1		Customer::Withdrawal request			
			Authorization Status		Authority Check
		Withdrawal Request::Warehouse Manager		Contact Warehouse and Notify About Status are mutually exclusive	Contact Warehouse
		Order Status::Customer			Notify about status
2		Warehouse Manager::Order Status			
		Order Status::Customer			Notify about status

Figure B-3 Office Clerk Agent Template

Truck Driver					
	Attributes			Operations	
Activity	State Law	Interface Attributes	Internal Attributes	Transformation Law	Operation
1		Planner::Transport orders			
		Pickup Notification::Warehouse Manager			Notify about pickup
		Truck Status::Planner			Report truck status
			Truck Status		Drive to pickup warehouse
		Ready to load::Warehouse Worker			Notify ready to load
		Truck Status::Planner			Report truck status
2		Warehouse Worker::On Transport			
		Dropoff Notification::Warehouse Manager			Notify About Dropoff
		Truck Status::Planner			Report Truck Status
			Truck Status		Drive To Dropoff
		Ready To Unload::Warehouse Worker			Notify Ready To Unload
		Truck Status::Planner			Report Truck Status

Figure B-4 Truck Driver Agent Template

Planner					
	Attributes			Operations	
Activity	State Law	Interface Attributes	Internal Attributes	Transformation Law	Operation
1		Warehouse Manager::Transport Form			
			Inventory Information		Mark Inventory
			Truck Information		Schedule Trucks
		Transport Schedule::Warehouse Manager			Notify About Transport
		Transport orders::Truck Driver			Issue Orders
2		Truck Driver::Truck Status			
			Transport Information		Record Time

Figure B-5 Planner Agent Template

Customer	
Incoming Changes	**Outgoing Changes**
Office Clerk::Order Status	Withdrawal Request::Office Clerk
Warehouse::Items and Documentation	Arrival::Employee

Figure B-6 Customer Agent Template

Employee					
	Attributes			Operations	
Activity	State Law	Interface Attributes	Internal Attributes	Transformation Law	Operation
1		Customer::Arrival			
			Customer Information		Check Documents
			Inventory Information		Fetch Items
		Items And Documentation::Customer			Deliver Items And Documentation
			Inventory Database		Remove Items from the System

Figure B-7 Employee Agent Template

Warehouse					
Component Agent 1: Employee					
Component Agent 2: Warehouse Manager					
Component Agent 3: Warehouse Worker					
Component Agent 4: Forklift Operator					
	Attributes			**Operations**	
Activity	State Law	Interface Attributes	Internal Attributes	Transformation Law	Operation
1		Warehouse Manager::Item Existence			
			Inventory Information		Check Inventory
		Search Results::Warehouse Manager			Notify about search results

Figure B-8 Warehouse Agent Template

Warehouse (Composited)					
Component Agent 1: Employee					
Component Agent 2: Warehouse Manager					
Component Agent 3: Warehouse Worker					
Component Agent 4: Forklift Operator					
	Attributes			**Operations**	
Activity	State Law	Interface Attributes	Internal Attributes	Transformation Law	Operation
1		Office Clerk::Withdrawal Request			
			Inventory Information		Check Inventory
		Item Existence::Warehouse		Contact Other Warehouses and Notify About Status are mutually exclusive	Contact Other Warehouses
		Order Status::Office Clerk			Notify about status
2		*Warehouse::Item Existence*			
			Inventory Information		*Check Inventory*
		Search Results::Warehouse			*Notify about search results*
3		Warehouse::Search results			
		Transport Form::Planner		Contact planner and notify about status are mutually exclusive	Contact planner
		Order Status::office clerk			Notify about status
4		Planner::Transport			

		Schedule			
		Order Status::office clerk			Notify about status
5		Truck Driver::Pickup Notification			
			Inventory Information		Move Items
6		Truck Driver::Ready To Load			
			Inventory Information		Load Truck
		On Transport::Truck Driver			Mark As On Transport
7		Truck Driver::Dropoff Notification			
			Warehouse Information		Find Space
			Movement Schedule		Arrange Movement
8		Truck Driver::Ready To Unload			
			Transport Form		Receive Items
			Warehouse Information		Determine Place
			Inventory Information		Move Items
			Transport Form		Mark Transport Request As Performed
			Inventory Database		Update Inventory Database
9		Customer::Arrival			
			Customer Information	Check Documents	
			Inventory Information	Fetch Items	
			Items And Documentation::Customer	Deliver Items And Documentation	
			Inventory Database	Remove Items from the System	

Figure B-9 Warehouse Agent Template in 'composited' view

Warehouse Worker					
Superagent: Employee					
	Attributes			**Operations**	
Activity	**State Law**	**Interface Attributes**	**Internal Attributes**	**Transformation Law**	**Operation**
1		Warehouse Manager::Ready Items			
		Move Items::Forklift Operator			Issue Move Item Orders
2		Truck Driver::Ready To Load			
			Inventory Information		Load Truck
		On Transport::Truck Driver			Mark As On Transport
3		Warehouse Manager::Ready Receiving			
			Warehouse Information		Find Space
			Movement Schedule		Arrange Movement
4		Truck Driver::Ready To Unload			
			Transport Form		Receive Items
			Warehouse Information		Determine Place
		Move Items::Forklift Operator			Notify Forklift Operators
			Transport Form		Mark Transport Request As Performed
		Update Form::Warehouse Manager			Send Update Form

Figure B-10 Warehouse Worker Agent Template

Warehouse Manager					
Superagent: Employee					
	Attributes			**Operations**	
Activity	**State Law**	**Interface Attributes**	**Internal Attributes**	**Transformation Law**	**Operation**
1		Office Clerk::Withdrawal Request			
			Inventory Information		Check Inventory
		Item Existence::Warehouse		Contact Other Warehouses and Notify About Status are mutually exclusive	Contact Other Warehouses
		Order Status::Office Clerk			Notify about status
2		Warehouse::Search results			
		Transport Form::Planner		Contact planner and notify about status are mutually exclusive	Contact planner
		Order Status::office clerk			Notify about status
3		Planner::Transport Schedule			
		Order Status::office clerk			Notify about status
4		Truck Driver::Pickup Notification			
		Ready Items::Warehouse Worker			Notify to Ready Items
5		Truck Driver::Dropoff Notification			
		Ready Receiving::Warehouse Worker			Notify to Ready Receiving
6		Warehouse Worker::Update Form			
			Inventory Database		Update Inventory Database

Figure B-11 Warehouse Manager Agent Template

Appendix C - A Discussion of the BWW Ontology

The following discussion of the BWW Ontology is based on Parsons and Wand (Parsons and Wand, 1997), Wand & Weber (Wand and Weber, 1993, 1995), and Wand and Woo (Wand and Woo, 2002). Ontological constructs are italicized as they are introduced.

C.1 Static Model of things

The real world is comprised of *things*. There are s*imple things*. A *composite thing* is made up of other things (be it other composite things or simple things or some combination thereof).

Things possess *properties*. Properties are either *intrinsic* (possessed solely by one thing) or *mutual* (shared with one or more other things). Properties exist independent of people being aware of their existence. *Attributes* are not necessarily possessed by a thing, but may be assigned to things by people in order to measure the property. For example, a property of a thing could be the ability to reflect a wavelength of light. People will then attribute a color to this thing. Composite things possess *hereditary properties* that belong to its component things. Composite things also possess *emergent properties* that are not possessed by any component thing. As an example a car is a composite thing. It has the hereditary property burns gasoline, from one of its component things the gasoline injected combustion engine. It also possesses the emergent property maximum acceleration which is not possessed by any one component thing.

The set of values for the attributes of a thing comprise the *state* of the thing. A *conceivable state space* for a thing is the set of all possible states a thing may ever assume. *State laws* serve to restrict the values of the properties of a thing to a subset of the conceivable state space. State laws must enforce a restriction due to either natural or human laws. A law is a property. For example, most bank accounts have the restriction (state law) that the balance must be greater than or equal to zero. This is due to the human law that people are only allowed to spend up to the total amount of money that is in their bank account. The *lawful state space* of a thing is the set of states that exist for a thing that comply with its state laws. A lawful state space is usually a subset of the conceivable state space.

A *class* is a set of things that all possess a common property. A *kind* is a set of things that possess two or more common properties. It should be noted that common property does not refer to mutual property. All things that are part of a *natural kind* will possess the same lawful state space.

C.2 Dynamic Model of things

Ontology has the principle that every thing changes and that every change is the change of properties of things. This is an *event*. Since the measures of properties of a thing comprise the state of the thing, and an event is changing properties. An event is really the change of the state of a thing. Ontology follows the principle of nominal invariance. That is, a thing can change and still be the same thing. An event is carried out via *transformation* (defined below). The *event space* is the set of all possible events that can occur in a thing.

A transformation is a state (attribute) change from one state to another state. A *lawful transformation* defines the events that are lawful for a thing. The *lawful event space* is usually a subset of the event space, and defines those events in a thing that are lawful. The *history* of a thing is the chronologically ordered states that it has traversed.

C.3 Static Model of systems

A *coupling* occurs when one thing acts on another thing that is, the existence of one thing affects the history of another thing. When this situation arises among two things they are said to be coupled (or interact). A set of things can be called a *system* if when the set of things is bipartitioned there are couplings among the things in the two subsets. The things that make up the system are its *system composition.*

A *system environment* is composed of the things that are not in the system but interact with things in the system. Systems have a *system structure.* A system structure consists of the couplings among the things in the system, and the couplings among things in the system and things in the system environment.

There exist *subsystems*. A subsystem is a system of which the composition and structure are subsets of the composition and structure of another system. A system may be broken down into a *system decomposition*. A system decomposition is a set of subsystems in which every component of the system is either one of the subsystems, or included in the composition of one of the subsystems in the decomposition. Generally, system decompositions require some sort of *level structure*. A level structure is a partial ordering over the subsystems in the decomposition. The level structure defines which subsystems are components of the system or other subsystems.

C.4 Dynamic Model of systems

Things, subsystems, and systems do not begin to change unless they are given impetus from an *external event* (defined below). The state a thing, subsystem, or system remains in unless it is forced to change by an external event is called a *stable state*. An *unstable state* is a state that must be changed to another state by actions within the thing, subsystem, or system. Unstability occurs due to the transformation laws that exist within a thing, subsystem, or system. The end result of the state change caused due to instability can be either a new unstable state or a stable state.

An external event is an event that affects a thing, subsystem, or system caused by some other thing in the environment of the thing, subsystem, or system. Before an external event occurs the thing, subsystem, or system is in a stable state. After an external event the thing, subsystem, or system may be stable or unstable. An *internal event* occurs in a thing, subsystem, or system due to lawful transformations of the thing, subsystem, or system. Before an internal event the thing, subsystem, or system is in an unstable state. After an internal event the thing, subsystem, or system may be in either a stable state or an unstable state.

Besides being internal or external, events can also be well-defined or poorly defined. A *well-defined event* is one in which the state of the thing, subsystem, or system after the event can be predicted just by knowing the state of the thing, subsystem, or system prior to the event. A *poorly-defined event* is one in which the state of the thing, subsystem, or system after the event cannot be predicted just by knowing the state of the thing, subsystem, or system prior to the event.

Appendix D – Modeling Grammar Examples

The case used in our examples is the same one used in chapter three and is presented again here in its entirety.

The ACME Warehouse Management Inc. Case[18]

ACME Warehouse Management Inc. offers storage facilities and redistribution services (between their different warehouses) across the nation. A customer can request space in a particular warehouse, request items to be transferred to another warehouse, or request withdrawal of items from a particular warehouse (even for items not stored there).

For the purpose of this case, we only look at the activities involved in processing a withdrawal request. A customer contacts ACME headquarters to request a withdrawal. An office clerk checks whether the customer has the authority to withdraw the items. The clerk then passes the withdrawal request to the warehouse where the customer wants to pick up the items.

If the warehouse does not have the items or does not have enough quantity of the items, the warehouse manager will contact other warehouses for the requested items. If the items are located the warehouse manager will ask the planner to arrange for transportation for the requested items.

The planner's responsibility is to schedule the company's truck fleet to accommodate requests for transportation, taking into account the existing schedule of each truck and its capacity. The warehouse manager will be notified whether the transportation request can or cannot be satisfied.

The warehouse manager will notify the office clerk if the request can be fulfilled or not, and the reason. The office clerk will notify the customer as to the status of the request (approved, or declined due to lack of authority, no inventory, or no transportation).

The planner issues transport orders to truck drivers. After receiving a transport order, the truck driver informs the warehouse about the pickup of the items. The warehouse manager will make arrangements to have the items ready when the truck arrives. When the truck arrives at the warehouse the items are loaded. The truck driver then informs the next warehouse about the delivery. When the truck has arrived at the next warehouse, the items are unloaded. A warehouse worker finds space for the items and arranges to have them moved to the allocated space. The worker updates the warehouse's inventory information. Truck drivers are required to report the status of the truck and the delivery to the planner after each step.

[18] Based on a case in I. Jacobson, Object-Oriented Software Engineering, Addison-Wesley, 1992

The customer will come to the warehouse on the required date to pick up the items. A warehouse employee will check all the necessary documents and will deliver the items with an accompanying documentation to the customer.

Supplemental description

Once the office clerk has recorded the items to be withdrawn, he or she forwards the request to the manager (foreman) of the warehouse. The warehouse manager is responsible for directing the redistribution of items between warehouses. If the items are not all available in the warehouse, transport requests are issued. The warehouse manager fills out a redistribution form with the following information: items to be moved, place from which to take the items, warehouse to transport the items to, quantity to be moved, and the date by when the redistribution must be done. The warehouse manager forwards the form to the planner to organize the interwarehouse transportation of the items. The items to be moved are marked as move-pending, and the planner initiates a plan to have the items at the appropriate warehouse at the given date. Once interwarehouse transport plans are finalized, transport requests are issued to the truck drivers.

The truck driver alerts the warehouse manager of the time he or she will be at the warehouse to pick up the items. The warehouse manager gives appropriate requests to the warehouse worker on the date of delivery to have the items ready for when the truck is expected. When the warehouse worker gets a request to fetch items, he or she, at the appropriate time, orders forklift operators to move the items to the loading platform. The forklift operators execute the internal warehouse operation. When the truck driver arrives, the driver notifies the warehouse worker to have the items loaded into the truck. The truck driver notifies the next warehouse manager when it is expected to arrive at the next warehouse. The number of items in the current warehouse decreases, and the transport request is marked as on transport.

When the truck has arrived at the next warehouse, the truck driver notifies the warehouse worker to unload the items. The truck driver signs off the job. The warehouse workers receive the items and determine a place for them in the warehouse. Forklift operators are told to move the items to the new place in the warehouse. When the truck driver confirms the delivery of the items, the records are updated to reflect the new place for the items. The transportation time is recorded and stored. The redistribution and interwarehouse transport request are marked as performed. The warehouse worker fills in an inventory update form and sends it to the warehouse manager for confirmation and update of the inventory database.

When the customer has fetched the items the warehouse workers mark the withdrawal as ready. The items are removed (decreased) from the information system.

D.1 Colored Petri Nets

Figure D-1 shows the Colored Petri Net solution to the ACME Case.

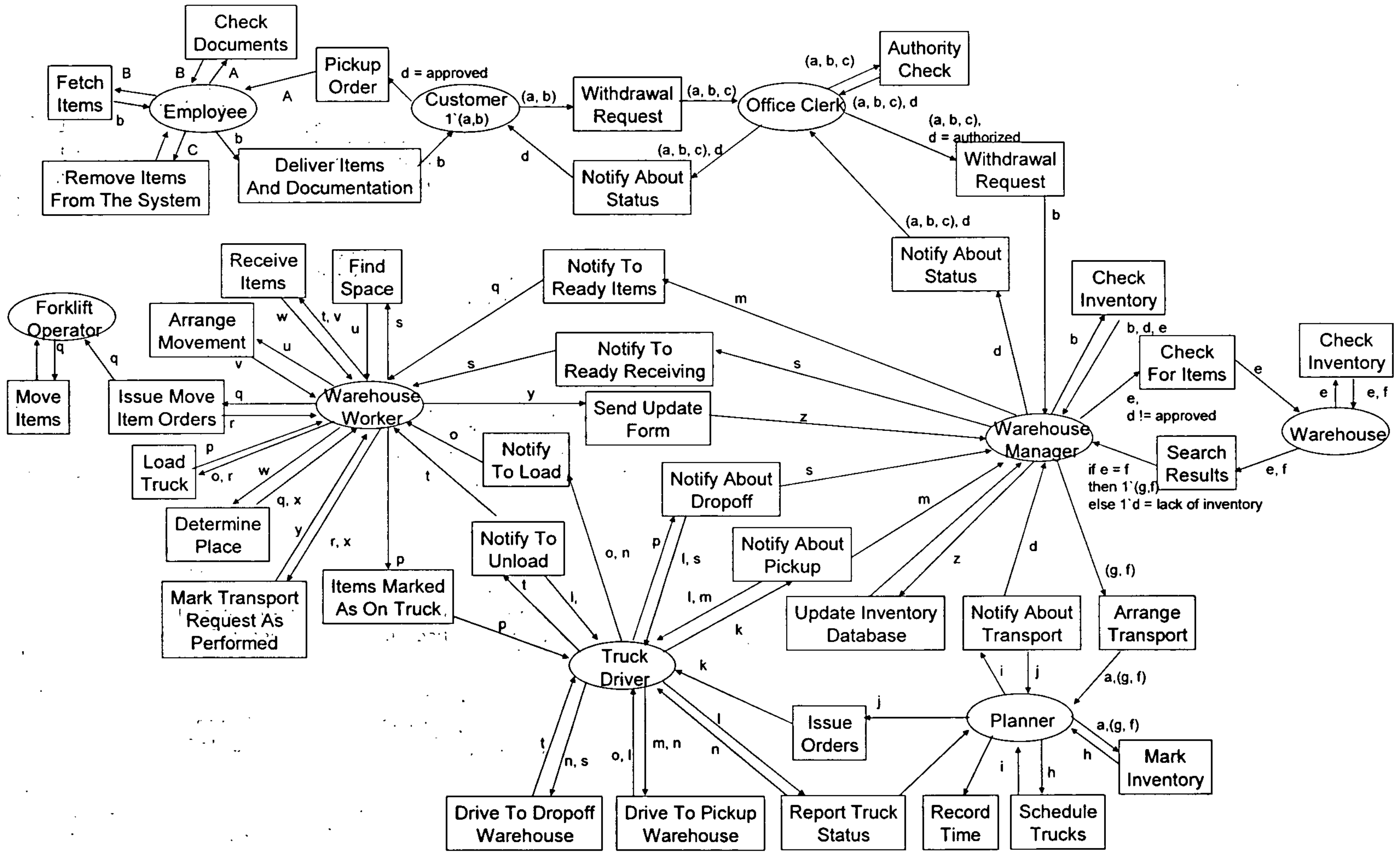


Figure D-1 Colored Petri Net for the ACME Warehouse Case

Declarations:

Token	String Array
var h: Inventory Marked	var a, g: Warehouse
var i: Trucks Scheduled	var b: Items
var j: Notified About Transport	var c: Customer
var k: Truck Schedule	var d: Status
var l: Ready To Report	var e: Items Not Found
var m: Notified About Pickup	var f: Items Found
var n: Report Done	
var o: At Pickup	
var p: Loaded	
var q: Items Need To Be Moved	
var r: Movement Orders Issued	
var s: Notified About Dropoff	
var t: At Dropoff	
var u: Space Found	
var v: Movement Arranged	
var w: Items Received	
var x: Place Determined	
var y: Transport Request Marked	
var z: Update Form	
var A: Customer Arrived	
var B: Documents Checked	
var C: Items And Documentation Delivered	
Withdrawal Request > Notify About Status;	

Table D-1 Declarations for the ACME Warehouse Colored Petri Net

In a Colored Petri Net the ellipses are called places and represent locations or resource stores. They represent the input and output of transitions. The rectangles are called transitions. Transitions are events, activities, or changes of state. Transitions create or destroy tokens. The arrows are directed arcs and represent necessary pre and post conditions of transitions. Each place has markers called tokens that contain a data value. The declarations table tells the type of value each token can takes. A 'token' declaration would be just there whereas a 'string array' declaration can hold a list of data. The declaration table also tells which transitions have priority when two or more of them are enabled at the same time.

D.2 Integrated Definition 3

Figure D-2 contains the process flow description for the ACME case. Figure D-3 contains the object state transition network for the ACME case.

In the process flow description the number box is a unit of behaviour (UOB). A UOB is a situation that happens. The arrows represent precedence links, with what happens at the tail of the arrow occurring before what happens at the head of the arrow. An arrow with a triangle on it means that the item at the tail of the arrow must precede the item at the head of the arrow. An arrow with a star means an the item at the head of the arrow must come after the item at the tail of the arrow and the item at the tail of the arrow must precede the item at the head of the arrow. The boxes with letters in them denote junctions. Junctions can be either and (&), or (O), or exclusive or(X), one vertical line in a junction denotes it is asynchronous, two vertical lines note it is synchronous. A UOB with either one or two vertical lines around the name is not a UOB but a referent. One vertical line denotes it is a call and continue referent, two vertical lines denote it is a call and continue referent.

An Object State Transition Network describes the states an object travels through. The circles represent object states. The arrows are the links. The boxes are referents to UOBs in the process flow description. The circles with letters in them are junctions of either and (&), or (O), or exclusive or (X). Referents connected at the same point begin at the same time. Referents connected to a circle means the order in which the occur is unknown but they can begin to occur then, and happen before other referents further down the line.

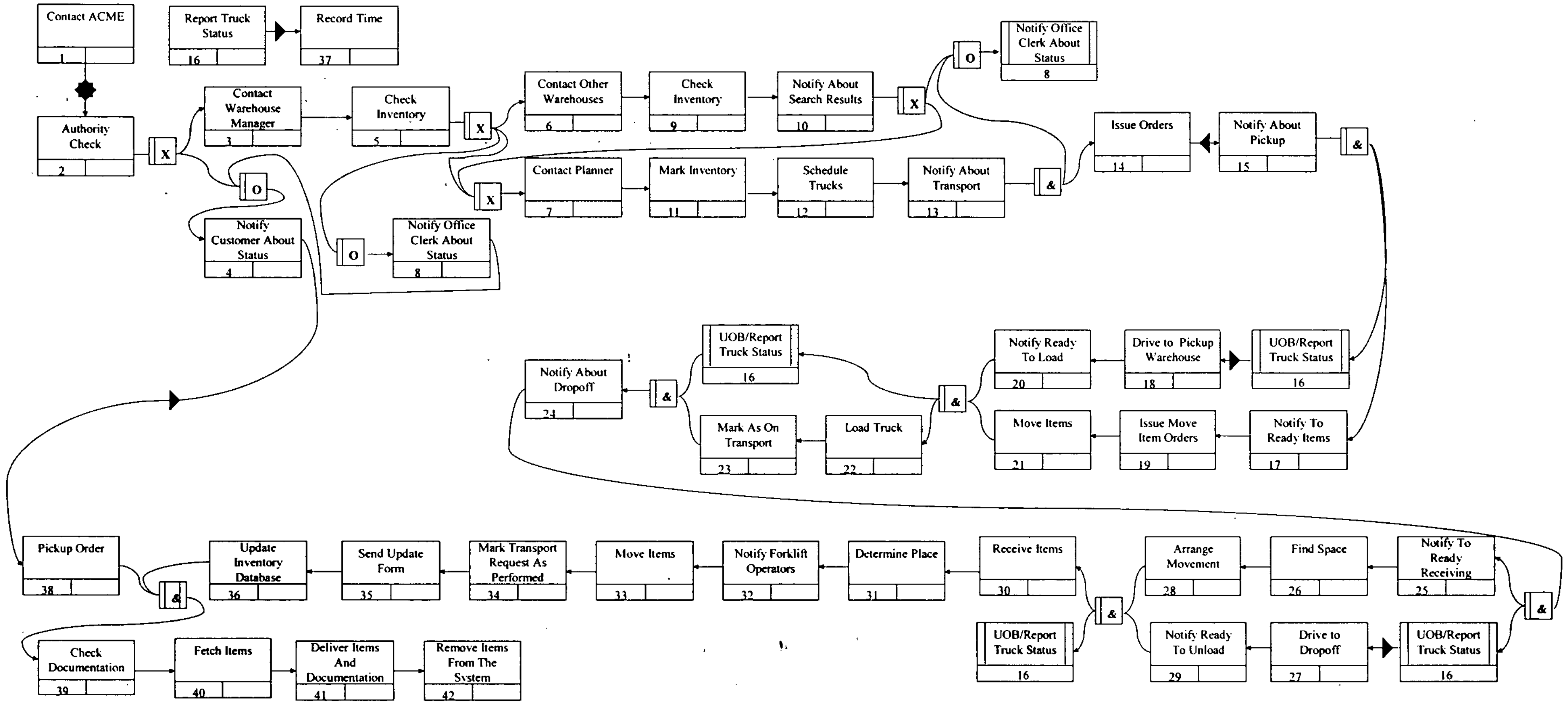
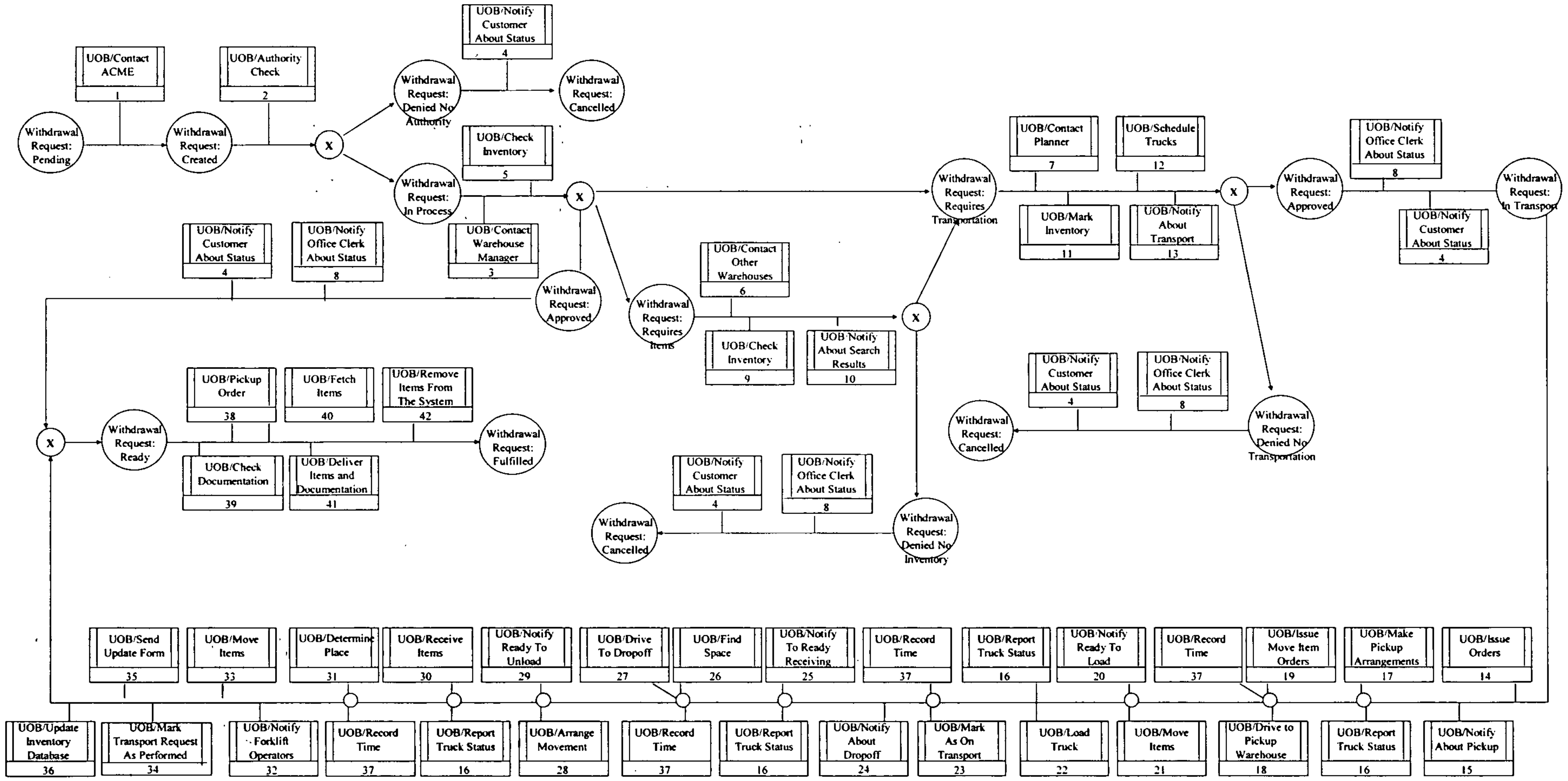

Contact ACME 1
Report Truck Status 16
Record Time 37
Authority Check 2
X
Contact Warehouse Manager 3
O
Notify Customer About Status 4
Check Inventory 5
X
Contact Other Warehouses 6
Check Inventory 9
Notify About Search Results 10
X
O
Notify Office Clerk About Status 8
X
Contact Planner 7
Mark Inventory 11
Schedule Trucks 12
Notify About Transport 13
&
O
Notify Office Clerk About Status 8
Issue Orders 14
Notify About Pickup 15
&
UOB/Report Truck Status 16
Drive to Pickup Warehouse 18
Notify Ready To Load 20
Notify To Ready Items 17
Issue Move Item Orders 19
Move Items 21
&
UOB/Report Truck Status 16
Load Truck 22
Mark As On Transport 23
&
Notify About Dropoff 24
&
Notify To Ready Receiving 25
Find Space 26
Arrange Movement 28
UOB/Report Truck Status 16
Drive to Dropoff 27
Notify Ready To Unload 29
&
Receive Items 30
UOB/Report Truck Status 16
Determine Place 31
Notify Forklift Operators 32
Move Items 33
Mark Transport Request As Performed 34
Send Update Form 35
Update Inventory Database 36
&
Pickup Order 38
Check Documentation 39
Fetch Items 40
Deliver Items And Documentation 41
Remove Items From The System 42

Figure D-2 Process Flow Description for the ACME Warehouse Case



Withdrawal Request: Pending
UOB/Contact ACME 1
Withdrawal Request: Created
UOB/Authority Check 2
X
Withdrawal Request: Denied No Authority
UOB/Notify Customer About Status 4
Withdrawal Request: Cancelled
Withdrawal Request: In Process
UOB/Check Inventory 5
UOB/Contact Warehouse Manager 3
X
Withdrawal Request: Approved
UOB/Notify Customer About Status 4
UOB/Notify Office Clerk About Status 8
Withdrawal Request: Requires Items
UOB/Contact Other Warehouses 6
UOB/Check Inventory 9
UOB/Notify About Search Results 10
X
Withdrawal Request: Denied No Inventory
UOB/Notify Customer About Status 4
UOB/Notify Office Clerk About Status 8
Withdrawal Request: Cancelled
Withdrawal Request: Requires Transportation
UOB/Contact Planner 7
UOB/Schedule Trucks 12
UOB/Mark Inventory 11
UOB/Notify About Transport 13
X
Withdrawal Request: Approved
UOB/Notify Office Clerk About Status 8
UOB/Notify Customer About Status 4
Withdrawal Request: In Transport
Withdrawal Request: Denied No Transportation
UOB/Notify Customer About Status 4
UOB/Notify Office Clerk About Status 8
Withdrawal Request: Cancelled
X
Withdrawal Request: Ready
UOB/Pickup Order 38
UOB/Fetch Items 40
UOB/Remove Items From The System 42
UOB/Check Documentation 39
UOB/Deliver Items and Documentation 41
Withdrawal Request: Fulfilled
UOB/Send Update Form 35
UOB/Move Items 33
UOB/Determine Place 31
UOB/Receive Items 30
UOB/Notify Ready To Unload 29
UOB/Drive To Dropoff 27
UOB/Find Space 26
UOB/Notify To Ready Receiving 25
UOB/Record Time 37
UOB/Report Truck Status 16
UOB/Notify Ready To Load 20
UOB/Record Time 37
UOB/Issue Move Item Orders 19
UOB/Make Pickup Arrangements 17
UOB/Issue Orders 14
UOB/Update Inventory Database 36
UOB/Mark Transport Request As Performed 34
UOB/Notify Forklift Operators 32
UOB/Record Time 37
UOB/Report Truck Status 16
UOB/Arrange Movement 28
UOB/Record Time 37
UOB/Report Truck Status 16
UOB/Notify About Dropoff 24
UOB/Mark As On Transport 23
UOB/Load Truck 22
UOB/Move Items 21
UOB/Drive to Pickup Warehouse 18
UOB/Report Truck Status 16
UOB/Notify About Pickup 15

Figure D-3 Object State Transition Network for the ACME Warehouse Case

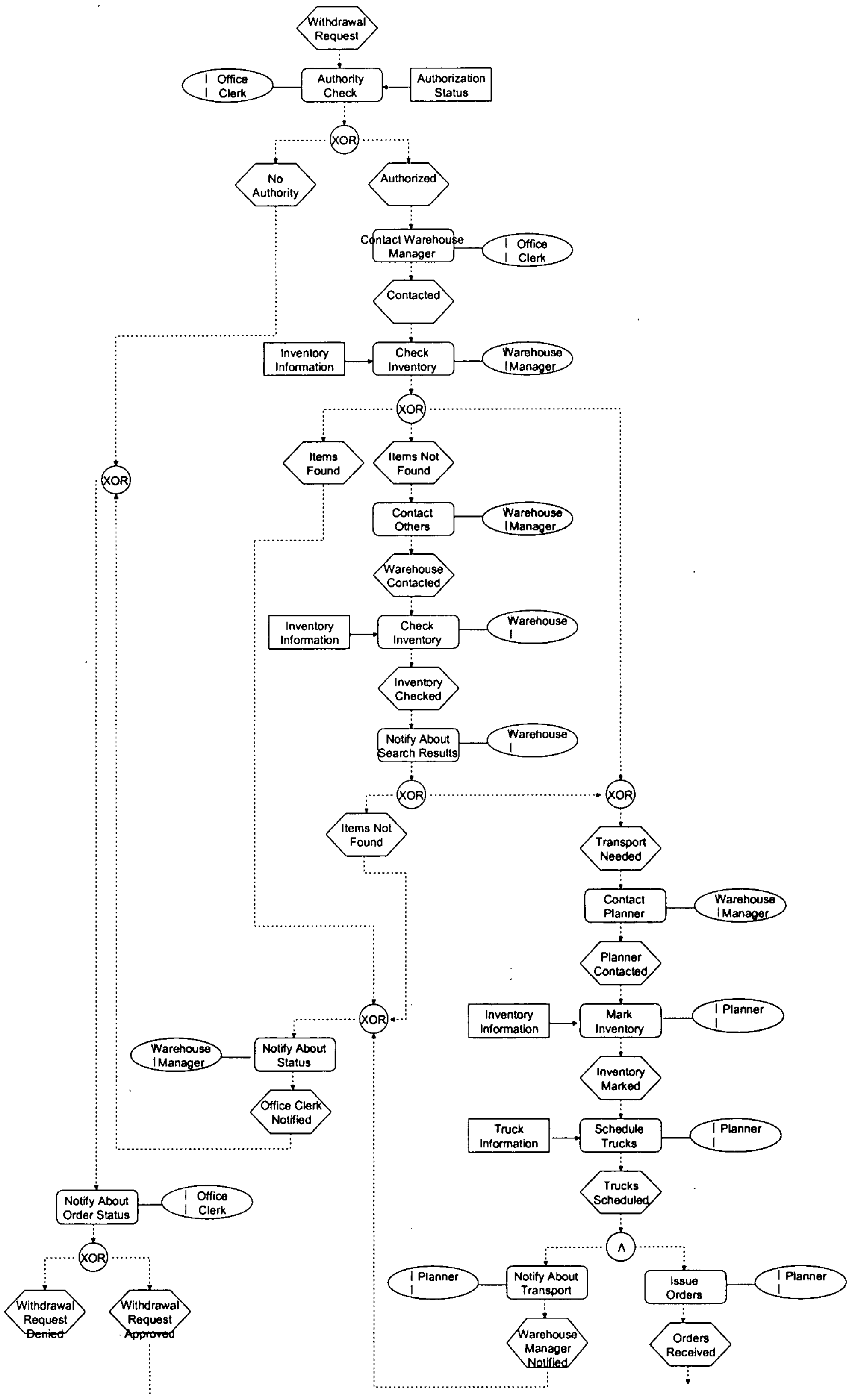
D.3 Event-controlled Process Chains

Figure D-4 contains the EPC for the ACME case.

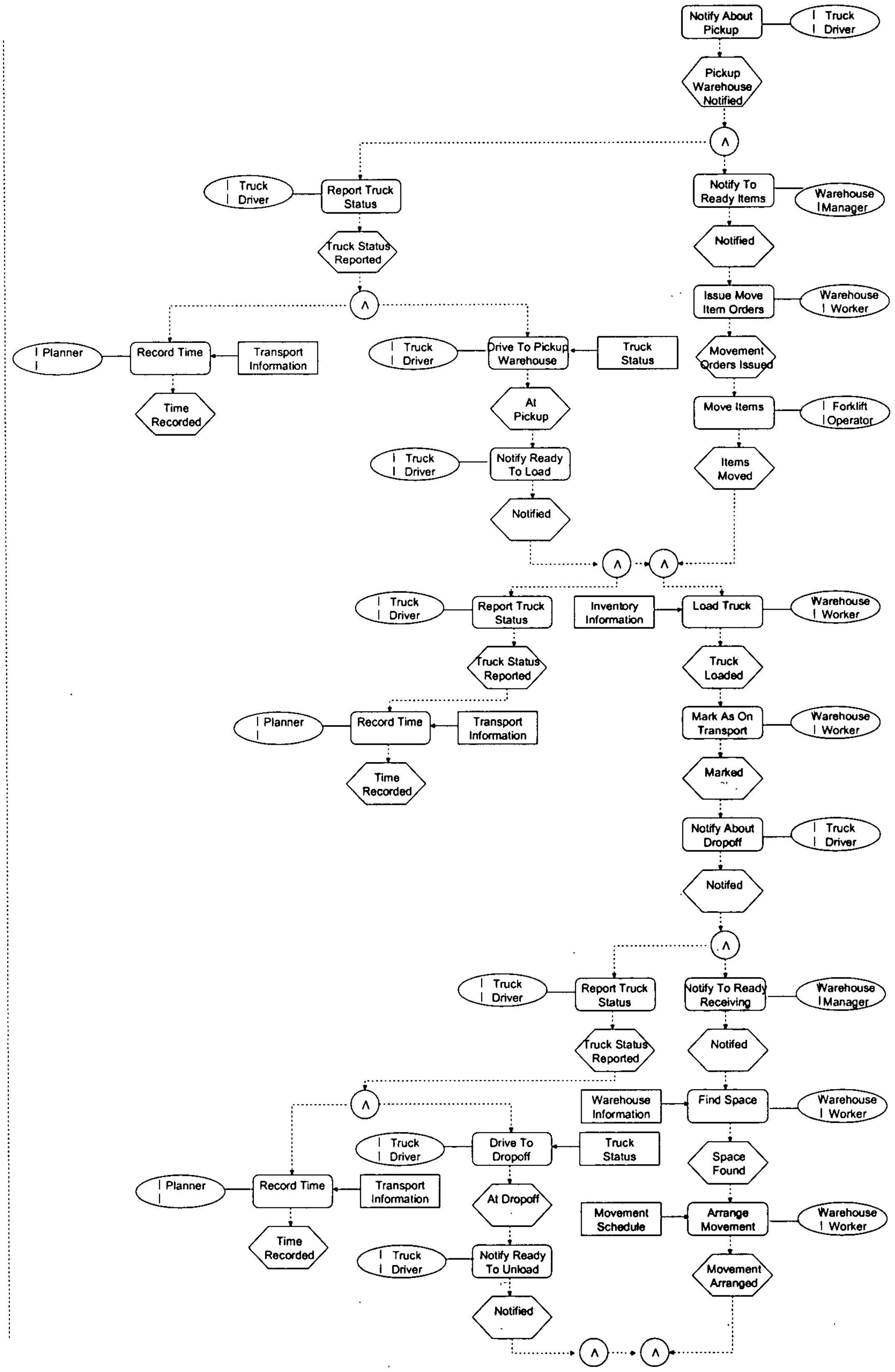
In EPC a rounded rectangle overlapping a hexagon denotes a process. A hexagon denotes an event, when something happens. A rounded rectangle denotes a task, what happens. An oval with a dashed vertical line denotes an organizational unit, who does the task. A rectangle denotes an information object, data accessed, created, or changed.

A dashed arrow denotes a control flow. Control flows connect events to tasks, and tasks to events. A solid arrow denotes an information/material flow, and connects information objects to tasks. A line denotes an organization assignment, and connects organizational units to task.

Events and tasks may be combined using circles with either an inverted v, a v, or XOR in them. These correspond to and, or, and exclusive or.



Withdrawal Request
Office Clerk
Authority Check
Authorization Status
XOR
No Authority
Authorized
Contact Warehouse Manager
Office Clerk
Contacted
Inventory Information
Check Inventory
Warehouse Manager
XOR
Items Found
Items Not Found
XOR
Contact Others
Warehouse Manager
Warehouse Contacted
Inventory Information
Check Inventory
Warehouse
Inventory Checked
Notify About Search Results
Warehouse
XOR
XOR
Items Not Found
Transport Needed
Contact Planner
Warehouse Manager
Planner Contacted
Inventory Information
Mark Inventory
Planner
XOR
Warehouse Manager
Notify About Status
Inventory Marked
Office Clerk Notified
Truck Information
Schedule Trucks
Planner
Trucks Scheduled
Notify About Order Status
Office Clerk
∧
XOR
Planner
Notify About Transport
Issue Orders
Planner
Withdrawal Request Denied
Withdrawal Request Approved
Warehouse Manager Notified
Orders Received



Notify About Pickup
Truck Driver
Pickup Warehouse Notified
Notify To Ready Items
Warehouse Manager
Notified
Issue Move Item Orders
Warehouse Worker
Movement Orders Issued
Move Items
Forklift Operator
Items Moved
Truck Driver
Report Truck Status
Truck Status Reported
Planner
Record Time
Transport Information
Time Recorded
Truck Driver
Drive To Pickup Warehouse
Truck Status
At Pickup
Truck Driver
Notify Ready To Load
Notified
Truck Driver
Report Truck Status
Inventory Information
Load Truck
Warehouse Worker
Truck Status Reported
Truck Loaded
Planner
Record Time
Transport Information
Mark As On Transport
Warehouse Worker
Time Recorded
Marked
Notify About Dropoff
Truck Driver
Notifed
Truck Driver
Report Truck Status
Notify To Ready Receiving
Warehouse Manager
Truck Status Reported
Notifed
Warehouse Information
Find Space
Warehouse Worker
Truck Driver
Drive To Dropoff
Truck Status
Space Found
Planner
Record Time
Transport Information
At Dropoff
Movement Schedule
Arrange Movement
Warehouse Worker
Time Recorded
Truck Driver
Notify Ready To Unload
Movement Arranged
Notified

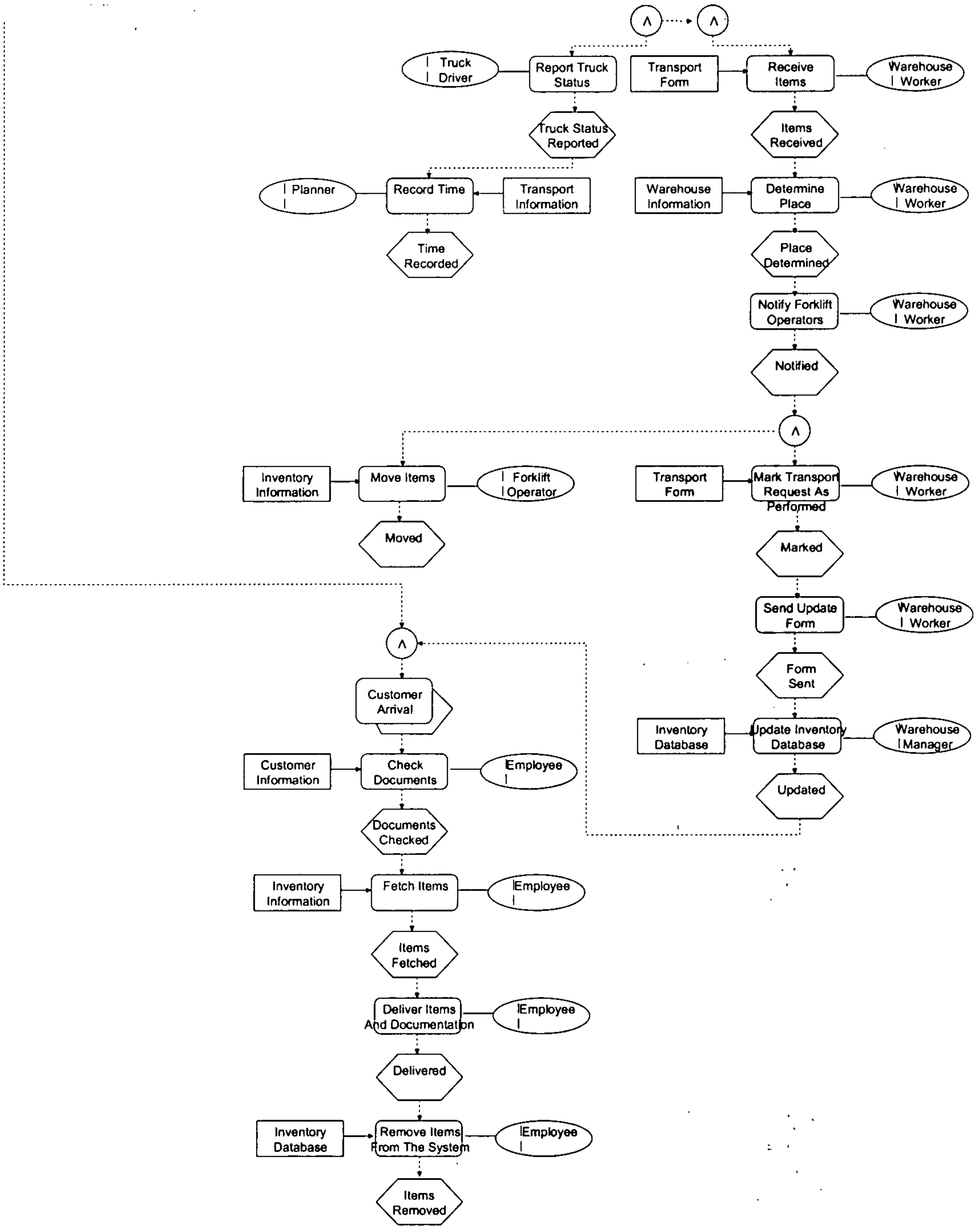


Figure D-4 EPC for the ACME Warehouse Case