Object-Oriented Enterprise Modelling:

by

Hao Zhao

B.Comm., Memorial University of Newfoundland, 1992

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF

THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

in

THE FACULTY OF GRADUATE STUDIES

(Faculty of Commerce and Business Administration)

We accept this thesis as conforming

to the required standard

THE UNIVERSITY OF BRITISH COLUMBIA

July 1995

Department of Commerce

The University of British Columbia
Vancouver, Canada

Date 95/07/28

ABSTRACT

This thesis presents a method for business analysis: Object-Oriented Enterprise Modelling (OOEM). The purpose of enterprise modelling is to assist user to understand, capture, and communicate both structural and dynamic aspects of business activities in an organization. Based on the research by Wand and Woo in this area, this thesis derives from ontology a formal model of an object and object-oriented system. It proposes a Request Propagation Algorithm, a systematic method for finding objects and creating an enterprise model. It also discusses the use of Object Communication Diagram to model the interactions among objects and Object Detail Template to specify services, attributes and request propagation of a single object. To illustrate the way in which the method can be applied, a reasonably realistic case analysis is presented. Finally, the thesis examines the contributions of OOEM to the field of object-oriented modelling by comparing it with Coad and Yourdon OOA, Jacobson's business modelling method, and Rumbaugh's OMT.

Table of Contents

Table of Figures

Table of Tables

Acknowledgments

This thesis would not be possible without the invaluable directions, suggestions, critiques, and encouragement of Professor Yair Wand, my thesis advisor. I am very grateful for his academic and personal support which made not only the thesis, but also my Master's study successful. The theoretical portion of the thesis is based on earlier research work of Yair Wand and Carson Woo. I want to thank both of them for the insightful discussions which stimulated many ideas presented in this thesis. I am also very thankful to Dean Uyeno who made important comments to the structure and contents of this thesis. Thanks also go to my fellow graduate student William Tan with whom I discussed many times about the suitability of the method for a CASE tool development.

Finally, I want to thank my wife Qing for her emotional and financial support throughout the whole program. Without her encouragement and sacrifice, this thesis would not have been possible.

1. Introduction

Human beings understand the world by translating external events into internal models and reason by manipulating these symbolic representations [Craik, 1943]. In the information systems field, the literature shows an increased interest in what is termed "Enterprise Modelling"[Glykas , 1993; Gorman, 1994]. The common thrust of research in this area is to model the structural and behavioral aspects of an enterprise with the purpose of helping management and system designers to understand the organization and reason about it. An enterprise model facilitates the communication of knowledge about the organization and forms a framework on which integrated information strategies can be established.

Object-Orientation was first developed as a programming and design approach to solve specific software problems such as simulation modelling [Dahl et. al., 1968] and graphical user interface [Graham, 1994]. Recently, Object-Oriented principles have been adapted to information systems analysis and subsequently to enterprise modelling. Essentially, system analysts are interested in modelling the organization activities using object-oriented concepts. Graham [1994] argues that "object-oriented methods can be applied to modelling the structure of an organization and the information flows within it, with the aim of providing a model of the business that could be used to simulate different organizational strategies". It is claimed that object-oriented enterprise modelling can be easily communicated to non-modelling experts in organizations[Glykas, et. al., 1993].

However, literature research shows that some of the modelling constructs in the object-oriented approach(OOA) are vague and ill-defined. For example the most fundamental *object,* most fundamental construct in OOA, is defined in a rather informal manner such as "An object is a person, place, or thing. An object needs not be physical. In natural language, nouns and noun phrases represent objects." [Embley et. al., 1992]. Rumbaugh [1991, p21] defines an object as "..a concept, abstraction, or thing with crisp boundaries and meaning for the problem at hand." Such an ambiguous definition does not provide analysts with a systematic and concrete means to find objects, the most crucial step in object-oriented analysis. We believe such state of confusion is the result of a lack of theoretical foundation for the essential object-oriented modelling constructs.

One of the research goals in this thesis is to build an object-oriented enterprise model based on a set of well-defined constructs with clear semantics. This is the continuation of research efforts in this direction [Wand, 1989], [Wand and Weber, 1990a, 1990b], [Takagaki, 1990], [Takagaki and Wand, 1991], [Parsons and Wand, 1993], [Wand and Woo, 1992, 1993]. One of the result of these researches is a set of modelling constructs and rules based on ontology - a branch of philosophy dealing with modelling the existence of things. In particular, we adapt the ontology proposed by Mario Bunge [1977, 1979]. For example, the concept of *object* is based on the model of *thing* and *request and message passing* is defined in terms of *interactions* between things. These well-formalized modelling constructs provide clear and unambiguous semantics to the enterprise model.

1.1. Thesis Objectives

As a continuation of the research work in the Faculty of Commerce and Business Administration at UBC on developing an ontologically-based Object-Oriented Enterprise Modelling (OOEM) method, this thesis is set out to achieve the following:

- to define the domain and purpose of object-oriented enterprise modelling.
- to define OOEM modelling constructs and modelling rules.
- to describe OOEM representation and model creation method.
- to demonstrate how OOEM method can be applied to a case.
- to determine the contributions of OOEM to the field of OOA.

*1.2. **Thesis Outline***

Chapter 2 introduces the concepts of enterprise modelling and discusses its benefits. Chapter 3 defines OOEM modelling constructs and rules based on ontological principles. Chapter 4 operationalizes these constructs and rules by describing OOEM object representations, system representation, and model construction algorithm. The semantics of OOEM notations are also presented and defined in this chapter. Chapter 5 puts the theories and methods into practice by presenting a case study where a manufacturing organization is analyzed and modelled. The process of creating the OOEM model and reasons for modelling decisions are also discussed in detail. Chapter 6 compares OOEM with three other OOA methods: Coad and Yourdon, Jacobson and Rumbaugh. Finally, Chapter 7 summarizes the entire thesis and discusses its contributions, limitations, and future research directions.

2. Enterprise Modelling

This chapter defines the domain and purpose of enterprise modelling. It should be noted that the field of enterprise modelling is relatively new. The term "enterprise modelling" appears sparsely in the IS literature. Few give a precise definition of the term. Here I attempt to define the term enterprise modelling based on literature research in two fields: cognitive psychology and systems analysis. Then I will briefly discuss potential benefits of enterprise modelling. This chapter lays the foundation for the rest of the thesis.

2.1. Models as Means of Understanding

It is claimed that humans understand the world by building models about it. In his book, *The Nature of Explanation*, Kenneth Craik [1943] argued that human beings translate external events into internal models and reason by manipulating these symbolic representations. He wrote [Craik 1943 p51], "By a model we thus mean any physical or chemical system which has a similar relation-structure to that of the processes it initiates. By 'relation-structure' I do not mean some obscure physical entity which attends the model, but the fact that it is a physical working model which works in the same way as the processes it parallels, in the aspects under consideration at any moment. ...". Johnson-Laird [1989] continued this line of thinking by arguing that people become experts in a particular domain by developing richer and more abstract models of that domain. "An important difference between the way a novice and an expert reason about a physical situation is that the novice's model represents objects in the world and simulates processes that occur in real time, whereas an expert can construct a model that represents highly abstract relations and properties [Johnson-Lard, 1989 p.485-486].

Therefore the key concept of a model is that it is a symbolic system which has similar structural and dynamic properties as the real system it simulates. A deeper understanding of the domain can be achieved by creating a model which captures the abstract relations and properties of the real system.

2.2. Enterprise Modelling and System Analysis

Systems analysis is the process of understanding the organizational environment of an information system and specifying the requirements of it. In order to specify requirements of a system correctly, a system analyst must first understand the related business areas by developing a model of the enterprise. Understanding the operations of a business enterprise is crucial for system analysis. Therefore modelling an enterprise is the logical first step for systems analysis and for formulating an enterprise wide information system strategy[Gorman, 1994].

2.3. Enterprise Modelling Defined

For the purpose of this thesis, I defined the term "Enterprise Model" as

> *a symbolic system which captures the essential and abstract relations and properties, both structure and dynamic, of an organization, usually as a whole.*

It must be stressed that model is an abstraction of the business enterprise in the real world, not a model of the software system to be implemented on certain technological platform. The distinction between enterprise modelling and systems analysis lies in the different universes of discourse. Enterprise modelling defines its domain as the physical enterprise itself, whereas traditional systems analysis methods typically grew out of the attempt to model the information content and system components of the application system to be constructed.

2.4. Benefits of Enterprise Modelling

2.4.1. Systematically capture the essentials of an organization

In the system development process, business knowledge must be solicited each time a new system is developed. Usually systems analysis is done by different teams with different set of users in different time. This leads to several models of different parts of the same organization. Disintegrated systems within an organization are the natural consequence of this method of system development.

In enterprise modelling, analysts must shift emphasis from system specific modelling to modelling the organizational activities. Enterprise modelling techniques can guide analysts to systematically explore the entire spectrum of the business activities in the organization. For example, the "request propagation" process as discussed in later chapters can be used as a guide to construct an enterprise model.

2.4.2. Capture the business knowledge explicitly

Real business knowledge in most organizations exists only in some employees' minds. This situation leaves the organization vulnerable to personnel changes. By capturing business knowledge and expressing it in a well-defined form, an organization can insure the consistency and continuity of its operation.

2.4.3. Assist Business Process Re-engineering Efforts

The major challenge that application developers are facing is the change in the organization activities the information system is designed to support. In an era of increasing competition and rapid technology changes, companies are forced to re-engineer their business processes. Traditional systems analysis methods are inadequate for the business re-engineer purpose since they presume a specific information processing problem which is usually related to one part of the organization. These methods are concerned more with data processing than with the business activities. Moreover, they are concerned with how things are done, not with radical changes. An enterprise model which captures the underlying business processes can assist management and re-engineering teams to understand, analyze, and improve these business processes.

3. OOEM Constructs and Rules

Wand and Woo [1993] proposed a set of constructs and modelling rules that can be used by analysts to construct an object-oriented model of an organization. This set of constructs and rules forms the foundation of the Object-Oriented Enterprise Modelling (OOEM[1]) method.

This chapter first reviews the object model in current object-oriented method literature with the purpose of providing a background for subsequent sections. The literature review shows a general absence of theoretical foundation in current object oriented methods since they are driven by practical software implementation issues. In search of a formal foundation, this chapter introduces the ontological principles and constructs proposed by Mario Bunge [1976, 1979]. Based on this formal model of the world, we define the concept of an *object*, the basic building block of our OOEM. Next we discuss important modelling constructs associated with an object such as *attributes, requests, services* and *request propagation.* Finally with these basic concepts defined, we present a set of modelling rules that gives practical guide to analysts. These rules are, in turn, the basis for the OOEM method described in the next chapter.

3.1. Current Object Models

The history of object-oriented programming started with the development of the discrete event simulation language Simula in Norway in 1967 [Dahl, et. al. 1968] and continued with the

[1] Here and in the rest of the thesis, OOEM refers to the specific modelling method originally proposed by Wand and Woo and further developed in this thesis.

development of Smalltalk in the 1970s [Kay et. al., 1977]. Since then, there have been many object-oriented programming languages including C++ [Stroustrup, 1986], Objective-C[Cox, 1986; Cox and Novobilski, 1991], Eiffel [Meyer, 1988], and even an object-oriented version of COBOL to name a few.

According to Graham [1994], the development in object-oriented programming has been influenced by the need to solve two major problems in programming: simulation modelling and graphical user-interfaces (GUI) development. Simulation modelling is particularly difficult problem for conventional languages which use functional flow of control whereas the problem is more naturally described in terms of complex objects which change states and influence events from moment to moment. Another reason for the success of object-oriented programming has been partially due to its inherited reusability which is particular useful in tackling the complexity associated with development of the WIMP[2] interface.

Essential object-oriented programming concepts include *Objects, Encapsulation, Messages, Classification, Polymorphism*. These concepts can be summarised as the following:

- **Object:** The basic units of construction, be it for conceptualisation, design or programming, are instances organised into *classes* with common features. These features comprise *attributes* and procedures, called *operations* or *methods*.
- **Encapsulation:** The data structures and implementation details of an object are hidden from other objects in the system. The only way to access an object's state is to send a message that causes one of the methods to execute.

[2] WIMP stands for Windows, Icons, Mice and Pointers and refers to a style of Graphical User Interface which makes use of them.

- **Messages:** Objects, classes and their instances communicate by message passing.
- **Classification:** Instances possess all the features of classes they belong to, but it is also possible in an object-oriented system to allow classes to inherit features from more general super-classes.
- **Polymorphism:** The ability to use the same expression to denote different operations is referred to as polymorphism. The same message could be sent to objects of different classes, and produce quite different results. Polymorphism is often implemented by *dynamic binding* in programming. [Graham 1994]

Given the historical background in programming, these concepts are clearly driven by implementation considerations. Main purpose of encapsulation, for instance, is to ensure that changes to data structures within objects do not propagate their effects to other parts of the system, and hence, improve the maintainability of the software system.

Wand [1989] suggests that the perspective of the object-oriented paradigm be shifted from implementation-driven to modelling-driven. He proposes that "the significance of objects [should] extend beyond the improvement of data and control transfer in programs, the portability and reusability of program components, and the advantages of information hiding. Rather, objects gain their importance because they reflect a 'natural' view of the world we are modelling (abstracting) in our software."(page 538). With the purpose of establishing a formal foundation of modelling reality instead of software system, we turn to ontology to seek a formal basis for the object-oriented approach. Ontology is a branch of philosophy that deals with modelling the existence of things in the world. In particular, we use ontological approach developed by Bunge in his "Treatise on Basic Philosophy" Ontology I [Bunge 1977] and Ontology II [Bunge 1979].

3.2. Ontology: A Formal Model of the World

3.2.1. Ontological Principles

Bunge provides general ontological principles to model the world [Bunge 1977, pp. 16-17]. These can be summarized as the following.

- *The world is composed of things.* Consequently, the sciences of reality (natural or social) study things, their properties and changes.
- *Things have properties.* (a) we study and modify properties by examining things and forcing them to change, and (b) properties are represented by predicates (e.g., functions) defined on domains that are, at least in part, sets of concrete objects.
- *Every thing abides by laws which are invariant relations among properties of the thing.*
- *Interacting things form systems or aggregates.* There is nothing that fails to be a part of at least one system. There are no independent things.
- *Everything changes and every change is a change of things.*

3.2.2. Ontological Constructs

Modelling constructs in Bunge's Ontology and their extensions to IS by Wand and Weber [1990] that are related to the purpose of information system can be organized into four categories.

1. Static model of a thing. These include thing, property, state, composite thing, law, and class;
2. Dynamic model of an individual. These include event, transformation, and history;
3. Static model of a system. These include coupling, system, composition, environment, structure, and subsystem;

4. Dynamic model of a system. These include stable state and unstable state, external event and internal event, and well-defined event and poorly defined event.

These constructs are informally defined and summarized in Appendix I. We next use them to define constructs in objects-oriented approach in terms of this ontology.

3.3. Object Defined

We define an *object* to be a model of a substantial thing in the problem domain that interacts with other objects. A class of objects is a set of objects that have common properties. First, an object is defined as a model of substantial thing. This reflects the ontological principle that the world is composed of things. Substantial things are usually physical entities, e.g., a person, or a computerized system. In the next subsection, we will further discuss the distinctions between substantial things and constructs to clarify some confusions surrounding the two concepts. Secondly, this definition specifies that, to be qualified as an object, a substantial thing must interact with other things in the modelling domain. Interaction models the *coupling* between things. Interaction with another object can be modelled as either submitting a *request*, or providing a *service* to it. In other words, objects in the enterprise model actively participate in the organization's activities, and are not merely subjects about which information is kept (i.e., entities). In the following subsection, we will distinguish the concept of object from that of entity in the ER model. Thirdly, this definition emphasizes modelling the aspects of the world that are relevant to purpose of the model. Such emphasis is in accordance with the ontological principle which recognize the difference between properties of thing and attributes assigned to things by the observer. As later chapters will show, OOEM only includes attributes and actions that are relevant to the purpose of modelling, not all properties of the things.

3.3.1. Things and Constructs

There should be a clear distinction between an object and a concept. Some object-oriented authors give very loose description of the concept of an object which seems to include all conceivable physical and conceptual entities. For example, Embley et. al. [1992, p18] make the following comment about an object: "An object is a person, place, or thing. An object may be physical or conceptual.... Some examples of conceptual objects [include] a height of 6'4", a temperature of 72°, and the dollar amount $100,000. We may also consider events to be conceptual objects." Such an all-inclusive definition of object does not give analysts effective guidelines to identify object types.

In Bunge's ontology, the world is made of substantial or concrete things which differ from constructs, the creations of human mind. He further distinguishes four basic kinds of constructs: concepts, propositions, contexts, and theories. Concepts such as the notion of a thing, are the building blocks of propositions - such as 'All things change' - which are in turn the constituents of contexts - such as the set of all propositions concerning dogs - and of theories. "We take it that constructs, whether useful or idle, scientific or mythical, are fictions not entities. Hence they are not part of the real world even when they take part in our representations of the latter."[Bunge 1977, p 118]

We adopt the view of Bunge's Ontology and define our objects to be things that response to requests and provide services. We stress that such defined object must be a substantial or concrete thing, not a conceptual one.

3.3.2. Object and Entity

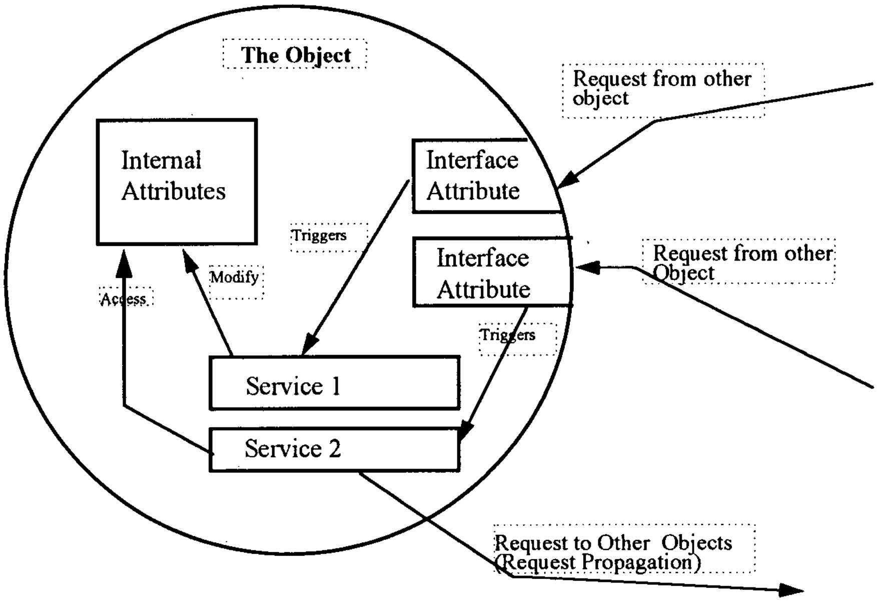
The concept of entity was defined as a thing which can be distinctly identified [Chen, 1976]. This concept forms the foundation of the Entity-Relationship model. Much of the object-oriented literature are influenced by the concept of entity. For example, Coad and Yourdon [1991, p53] define object as "an abstraction of something in a problem domain, reflecting the capabilities of a system to keep information about it, interact with it, or both". The danger of this lack of distinction between the concept of entity and object is that it may lead the analyst to prematurely consider the content of the database when s/he is modelling the organization. For example, in a vehicle registration and title system, Coad and Yourdon [1991] included *Vehicle* as a class-&-object with subclasses *Truck-Vehicle, Motorcycle-Vehicle,* and *Trailer-Vehicle.* Clearly, this is an attempt to model the data that will be processed by some organizational unit, but not the organizational unit itself. In the OOEM, *Vehicle* should not be modelled as object unless it engages in meaningful interactions (via its services) with other identified components of the system. According to the OOEM's modelling rule, the organizational unit that processes vehicle information should be modelled as an object, and vehicle information should be modelled as attributes of the object.

In enterprise model, most entities that are defined in ER model may not qualify as object types because they do not provide meaningful services. It is difficult to argue, for example, that a *product item* (an entity) provides a service called "Being ordered by a customer". In fact it is the sales person who provides the service of accepting and fulfilling a customer's order. Although not included in the enterprise model, an ER model can be helpful in organizing the information possessed by the salesperson, the next phase in the development life cycle.

3.4. The Anatomy of Object

To completely understand an object in our system, we need to study the properties of the object, the services provided by the object, and the interactions between it and other objects. This section proposes a model of an object, the basic building block of an OOEM. The anatomy of an object is illustrated in Figure 3-1. Each modelling construct is discussed in the subsequent subsections.

Figure 3-1: Anatomy of an Object



Here is a brief description of Figure 3-1. An object can communicate with other objects via requests. Communication with other objects can take two forms. On the one hand, an object can submit a request to another object to demand a service to be performed by the latter. For example, the arrow originating from a service in the object to another object representing such a request. On the other hand, an object can respond to a request by providing a service to another object, e.g., the two arrows coming into the object represent requests from other objects. Requests are usually associated with information necessary for the respondent to perform the service. In our object model, information associated with a request is modelled as interface attributes because they can be directly modified by other objects.

A request triggers a service represented as rectangular inside the object. A service is a series of actions with the purpose of fulfilling the corresponding request. In its course of action, a service may access and/or modify some of the internal attributes of the object. It may also generate requests to other objects, which in effect, starts or continues the "request propagation" process.

3.4.1. Attributes

During the discussion of ontology, we defined attributes as models of *properties of a thing*. In OOEM, attributes take more specific forms to describe the state of the object. Figure 1 illustrates two types of attributes: internal attributes and interface attributes. Internal attributes are not known to other objects and can only be accessed or modified through the services of the object. This reflects the principle of *encapsulation* and *object independence*. Interface attributes can be accessed by other objects.

3.4.1.1.Internal Attributes

Internal attributes model the *intrinsic properties* of a thing. For example, in a university, student information such as name, address, student number, grades and tuition payment, is kept in the registrar's office. It represents the knowledge of the registrar's office about each student. Moreover, according to the encapsulation principle of object-orientation, such student information should be kept inside the registrar's office object, hidden from outside world, i.e., the internal attributes of the *registrar's office* object. The student information can only be changed through the operations of the registrar's office. When a student changes address, she/he sends a request to the registrar's office. Such a request changes the knowledge (the state) of the registrar's office. Therefore student information should be modelled as the attributes of the Registrar's office, not of the student. Knowledge of an object represented by the state of the object is, therefore, represented by the attributes of the object.

It should be noted that internal attributes may not necessarily be included in the object model, since it is sufficient for other objects to interact with an object by knowing or modifying only its interface attributes. However, since analysts usually know more than what other objects can know about the object, internal attributes can be included in the internal view of the object.

3.4.1.2.Interface Attributes

Interface attributes model *mutual properties* of things. They are shared by both communicating objects. The interface attributes provide a mechanism by which objects communicate with each other. By sending a request which modifies the interface attributes of the recipient, an object changes the *history* of the latter. It is this change in interface attributes (change of state) that triggers the service being activated (*state transformation*). This reflects the ontological notion of *coupling*.

An analogy can be drawn between object communication and a function call in programming language. Interface attributes correspond to function call parameters which enable one program to pass information to another. Similarly, when one object sends a request to another object, it sends necessary information to enable the service to be performed. A request changes the interface attributes related to the service being invoked in the respondent object.

An example of interface attributes can also be found in the registrar's office example. A student can send a registration request to the register's office. This request is accompanied by information about student identification, year and semester, courses and etc. In OOEM, such information is viewed as interface attributes associated with the registration request. In this view, the action of sending a request to the registrar's office changes the knowledge, therefore, the state of the registrar's office. Only when such information is provided can the registrar's office begin to process the request to register.

There are two types of interface attributes: "incoming" interface attributes and "returning" interface attributes. The former models the information that is sent from the requester to the receiving object. The latter models the information returned to the requester as a response to the initial request. An example of the latter is "confirmation of registration" as a response to *register courses* request from a student. Both types of interface attributes are shared by the communicating objects.

3.4.1.3. Attributes: Difference between OOEM and Data Modelling

In most system analysis methodologies literature, an attribute is used to describe data associated with the object or entity. The most common example is that a person (the object) has name, address, social insurance number, and phone number (the attributes of the object). Such definition of attributes is influenced by ER model. However, this definition is not sufficiently effective when one attempts to model a real world enterprise.

Our definition of attributes differs significantly from those used in traditional data modelling. As explained earlier, the aim of our enterprise modelling is to model the organization as a dynamic system interacting with its environment. We model the system components in terms of objects and the interactions among components and between the system components and the environment in terms of requests.

An example will make the difference clearer. In the student registration system, data modelling will treat the student information (i.e., student name, id number, address, year and semester registered and any other data belongs to the student) as attributes of student. Similarly, courses information (i.e., course name, section, time slot, room, number of seats, etc.) will be the attributes of a course. And the fact that some students are enrolled in courses is modelled in the student-enroll-course relationship. In OOEM, student, course, and enrollment information are kept *inside* the registrar's office, and can only be changed by the operations of the registrar's office. Therefore, they are the internal attributes of the registrar's office object. How the information is kept and what database technology should be employed are out of our domain of analysis[3].

Such a definition of attributes manifests the principle of information hiding. Each interface attribute is like a mail box slot; other objects can put in some information in the mail box associated with some requests. The change in the mail box signifies the change of state of the object (to an unstable state) which triggers the invocation of the corresponding service. How the object will respond to the request, e.g. how the student information will be stored, retrieved and process is not known to the outside world (encapsulation). Furthermore, the results of the

[3] Clearly, information modelling is included in the scope of later phases of the system development life cycle.

request, e.g. whether the student is registered successfully, should also be included in the interface attributes since it is known to the outside world.

3.4.2. Request

A request models the interactions between two objects. Ontology defines *interaction* to be the change of history of one thing as a result of the existence of another thing [Bunge, 1977]. When one object interacts with another, it sends a request to the latter. A request changes the interface attributes of the receiver which triggers corresponding service of the receiver being activated.

A request can demand the receiver to perform some actions, e.g., a customer orders some product from a salesperson. Alternatively a request can elicit some information from the receiver, e.g. a customer requests product and price information from a salesperson. There are potentially three consequences of a request: (1) the state of recipient changes; (2) the state of the sender of the request changes; (3) the states of both the recipient and the sender change.

First, a request is usually accompanied by some information which is necessary for the receiver to respond the request. We model this by a change in the interface attributes of the recipient. For example, when customer orders product, name of the product, quantity, agreed price, delivery details, payment details and other relevant information are sent along with the request. Such information further specifies how the service is to be performed. As a result of the request, the interface attributes of the order entry object change to reflect the new information available to the object. There is a certain degree of analogy between this aspect of request and the parameter passing of a function call in many computer languages. In both cases, the sender (the calling function or requesting object) sends information which will determine how the receiver (the called function or the respondent) will behave in respond to the request.

Secondly, a request usually implies a response. In case of a request for information, the response is the needed information being sent back to the requester, e.g. registrar's office sends a request to the financial service department to check if a student has paid tuition. In this case the sender changes state as a result of the response to the original request.

Finally, when one object requests another to perform some service, e.g. a student sends a request to the registrar's office to register a course, confirmation may be sent back to the originator. Here, states of both sender and receiver change as a result of the request. For example the registrar's office records that the student is registered in certain courses. The student also knows that s/he is registered in these courses and is responsible of complete them successfully.

However, not all requests necessitate information flow back to the sender from the immediate receiver. OOEM does not enforce the constraint that each request must have an immediate reply. When a customer orders some product, the order processing department may not response directly to the customer. Instead, the order processing department sends requests to several objects inside the company and eventually some objects in the enterprise (shipping and accounting departments) send the product and invoice to the customer. A more detailed discussion of the "Request Propagation" process inside a system is provided in the next section.

3.4.3. Service and Request Propagation

As discussed earlier, in order to qualify for an object in our model, a thing must be active, that is it must respond to at least one request from either an external object (client) or an internal object.

When a request is sent to an object, it invokes a service in the object. 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.

A service, in its course of action, may generate or spawn one or more requests to other objects. The actions associated with a service are not known to other objects (encapsulation). During system analysis, however, analysts need to know these actions to a certain extent so that requests spawned by the service are identified. Again in the registrar's office example, analysts need to know how *Register-a-course* (the service) is performed in order to determine the interactions between registrar's office and other objects in the system. If, for example, the registrar's office needs to check whether particular student has paid his/her tuition, a request is sent to the financial services department which provides *Check-Tuition-Payment* service. Hence, the request from a student (the client) triggers a series of requests inside the system. This process is called "Request Propagation". By tracing the request propagation process in the system, analysts can identify all objects and associated attributes and services these objects must possess.

3.5. Mapping OOEM Constructs to Ontology Constructs

After discussing modelling constructs used by both OOEM and Ontology, we summarize the relationship between them by mapping the former onto the latter in Table 3-1. The connections between the ontological constructs and OOEM constructs were discussed when we introduced each of the OOEM constructs. Here a summary is presented in Table 3-1. In general, these ontological constructs are more abstract in nature than those of OOEM since their purpose is to model a much wider domain than OOEM. OOEM's constructs can be viewed as special cases that operationalize of the ontological constructs for the purpose of enterprise modelling.

Table 3-1: Mapping OOEM Constructs to Bunge's Ontology

OOEM Constructs	Bunge's Ontological Constructs
Objects	Things
Attributes • Internal attributes • Interface attributes	Properties • Intrinsic properties • mutual properties
Services	Lawful transformation
Request • External Request • Internal Request	Event • External events • Internal events
System of interacting objects	System
External objects	System Environment

3.6. Modelling Rules

As a necessary part of OOEM method, I include in this section the modelling rules proposed by Wand and Woo [1993]. These rules provides fundamental guidelines for the model construction method discussed in next section.

3.6.1. Rule #1 The scope identification rule

This rule identifies the scope of the enterprise model. It draws the distinction between the enterprise and its environment. Essentially, such distinction reflects the analyst's *view* of the reality with the purpose of understanding a portion of it which is of interest to him/her. This portion of the reality might be an organization, a division of an organization, or in some cases, multiple organizations (e.g., to understand an EDI system). This rule states that the analyst should first identify the purpose of the model, i.e., the domain or the subject matters which s/he wishes to understand. This domain forms the scope of the enterprise model. Things inside the scope will be modelled to the level of details relevant to the purpose (see Rule #3 and Rule #4); whereas things outside the scope will be modelled only to the extent that their interactions with things inside the scope are understood.

The interactions between external things and the system are modelled as “external requests” to the system (e.g., customer orders a product). We term these the external objects or *clients* of the system. An external request is sent to a thing internal to the system (e.g., order entry clerk). As a result of the request, certain actions (e.g., taking order) are invoked on that object which in turn may invoke actions on some other objects within the system(e.g., inventory management is asked to ship the product).

3.6.2. Rule #2 The object identification rule

An object should be included in the model if and only if it provides at least a service to or requests at least a service from the system. These requests or services should be part of the system activities as described in the scope of the system (rule #1). An internal object is an object which is considered part of the system and provides at least one service. An external object is an object that interacts with the system but is not considered part of it. An external object can either generate a request to or provide services to the system.

3.6.3. Rule #3: The service inclusion rule

A service will be included in an object if and only if it is invoked by at least one request included in the view (as defined in rule #1) or the system. Note that such a request may be generated directly by an external object or indirectly by an internal object as part of the system’s response to an external request.

3.6.4. Rule #4: The attributes inclusion rule

To be necessarily included in the object model, an interface attribute must be (1) used or affected by at least one service of the object and (2) "known" to at least one other object. An internal attribute must be affected by at least one services of the object and "unknown" to other objects.

3.6.5. Rule #5: The attribute ownership rule

An important ontological principle is that properties always belong to things. Therefore in the enterprise model, there are no attributes in the system that do not belong to a specific object. In our model, every attribute can be modified by the service of one object only. The object that can modify an attribute will be called the "custodian" of the attribute. The only way for other object to obtain or to modify the value of an attribute is by an action of the custodian object.

3.6.6. Rule # 6: Aggregation and Decomposition rule

This rule governs when the composite object is shown and when the component of the object is shown in the OOEM. It states that a composite should be included only if it provides service not provided by any of its components. Here the term 'composite' and 'composite object' are used to refer to aggregate objects that can be decomposed into component objects in the problem domain. When modelling the properties of the composite object, only properties that are not properties of components should be included in the model of a composite. This rule reflects the ontological principle that a composite thing must have emergent properties - properties which are not processed by any of its components.

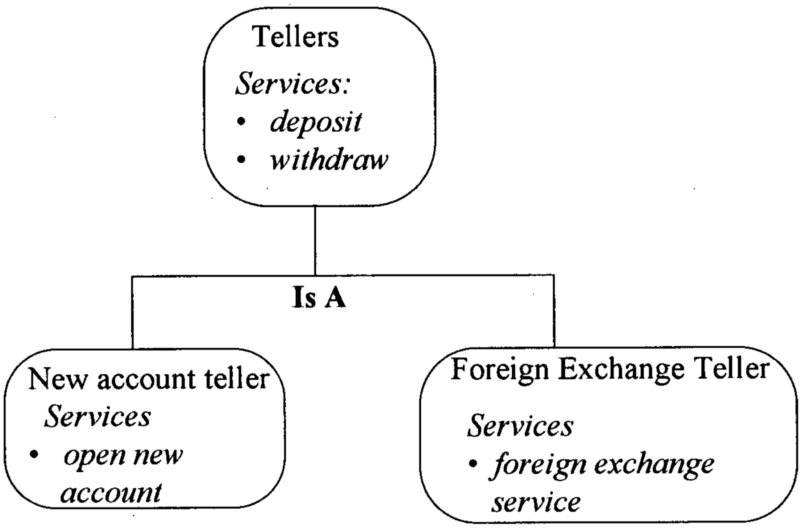
The concept of aggregation is concerned with the structural aspect of the system. It aids the analyst to group and organize cooperating objects into composite object with the purpose to effectively manage complexity. This concept is orthogonal to the concept of request which is used to model the dynamic aspect of the system. Since composition is a structural concept, there should be no request between the composite objects and its component.

3.6.7. Rule # 7: Generalization and Specialization

This rule states that if two or more object classes provide one or more common services, we can create a general object class. This general object class provide the common services. This general object class is called super-class of the original classes. The original object classes are called sub-classes of the general object class. Each of the sub-classes only possesses the unique services. All services provided by the super-class should be eliminated from the sub-classes. This rule is based on the ontological concept of natural kind which is a set of things that share the same laws. Laws are manifested in the OOEM as services, hence objects that provide the same services are of the same natural kind.

An example can be found in a bank. Assuming there are three types of tellers: 1) generic tellers who can do deposit and withdrawal; 2) new account teller who can open new accounts for customer; and 3) foreign exchange tellers who can provide foreign exchange services to customers. The last two kinds of teller can also do deposit and withdrawal. The generalization-specialization structure can be shown in Figure 3-2.

Figure 3-2: Example of Generalization-Specialization Structure



Similar to the concept of aggregation, generalization and specialization is a structural concept that is orthogonal to the dynamic concept of request. For the same reason, there should be no request between objects in the sub-classes with objects in the super-classes.

3.7. Chapter Summary

This chapter discussed the fundamental modelling constructs and modelling rules of OOEM. These constructs include objects, services, attributes and requests. The seven modelling rules dictate how these constructs can be applied in modelling reality. These constructs and rules are derived from ontological principles. A summary of Bunge's theory can be found in Appendix I. With the theoretical foundation laid, the next chapter presents the method of constructing an OOEM.

4. OOEM Method

The previous chapter presented the theoretical foundation of OOEM. This chapter attempts to put the theory into practice by giving procedural guides to construct an OOEM for an enterprise.

The key feature of this method is to focus on the behavioural aspects of objects, i.e., interactions among the objects. Interactions among objects are modelled in OOEM as requests from one object to another. What other objects in the system are interested in an object is what it can do and what service it can provide. The focus on interactions among objects also reflects the ontological principle that a thing is a component of a system only if it is coupled with other components of the system. To study an object, therefore, we need to study its behaviour.

The central theme of the OOEM method, therefore, is the "Request Propagation" concept discussed conceptually in the previous chapter. This chapter is more practical in nature. It shows that, by studying services and tracing the Request Propagation process, analysts can build a complete model of the organization which specifies all necessary objects and their behaviour. First, we provide a representation of a single object that captures its essential characteristics. Next we present algorithms to trace the *"Request Propagation"* process. By tracing this process, we can identify all objects along with their associated services, interface attributes, internal attributes, and request connections with other objects. Based on our experience with analyzing case studies, we also discuss practical guidelines on how to accomplish each of the steps in the algorithm.

4.1. OOEM Assumptions and Assertions

Before we discuss the OOEM model creation process, we first summarise the basic assumptions of OOEM.

1. There exist organizational activity descriptions based on which the OOEM can be created. Alternatively, analysts can pose question to problem domain experts regarding the organizational activities. The quality of the descriptions and the answers from domain experts ultimately determines the quality of the final OOEM.
2. Analysts have sufficient information to determine the scope of the organisation they are modelling.
3. By tracing the request propagation in an enterprise, analysts can identify all objects, their attributes and services.
4. Analysts can understand not only the terms used by various group of domain experts but also the underlying meanings of this terms. This assumption is necessary to avoid duplicate models of the same thing when different domain experts use different terms to refer to the same thing.
5. Analysts can choose and consistently work at a certain level of granularity of the objects.

The last three assumptions are concerned with the relationship between the analyst's ability and the quality of the final model. The OOEM method is no more than a set of tools and techniques that can be applied by analysts. A less capable analyst can use a superior method to create a bad model. This is true for all analysis methods. The greater the extent to which these assumptions are true, the higher quality the final model has. It should be noted, however, that if these assumptions about analysts hold to a lesser degree, the method will still result in an enterprise model. In this case, the problems with duplication and model granularity might be corrected by further reviews.

OOEM makes the following assumptions about the modelling of an enterprise.

- *System:* An enterprise can be modelled by a system of interacting objects
- *Scope:* The system has a well-defined scope. Objects inside the scope are internal objects. Objects outside the scope are external objects.
- *Interaction:* Interactions among objects are modelled as requests (or 'message passing').
- *Object:* An object is included in the model **iff** it interacts with at least another object.
- *Service:* A request invoke a service of the responding object.
- *Interface attribute:* A request is accompanied by information shared by the communicating objects. This information are modelled by interface attributes.
- *Internal attribute:* An object may encapsulate certain information which is necessary for it to provide the services and it is not included in the interface attribute. This information is modelled as internal attributes of the object.
- *Request propagation:* In the process of performing a service, an object generates zero to many requests to other objects. This is called request propagation process. This request propagation process starts with a request from an external object and terminates on external objects or objects which do not propagate request.

4.2. Object Representation

The goal of the OOEM is to identify objects, their services, attributes, and the interactions between objects. For each object, we need to know its services, attributes, and requests to other objects spawned by each service as part of the activities to deliver that service. We also need to understand the message connections between objects, i.e., who is the originator of the request, who is the respondent and what service the request invokes. As discussed previously, there are two types of objects in OOEM: internal objects and external objects. Internal objects are considered to be part of the system. External objects are objects interacting with the system but are not included in the scope of the system. Representations of these two types of objects are discussed separately.

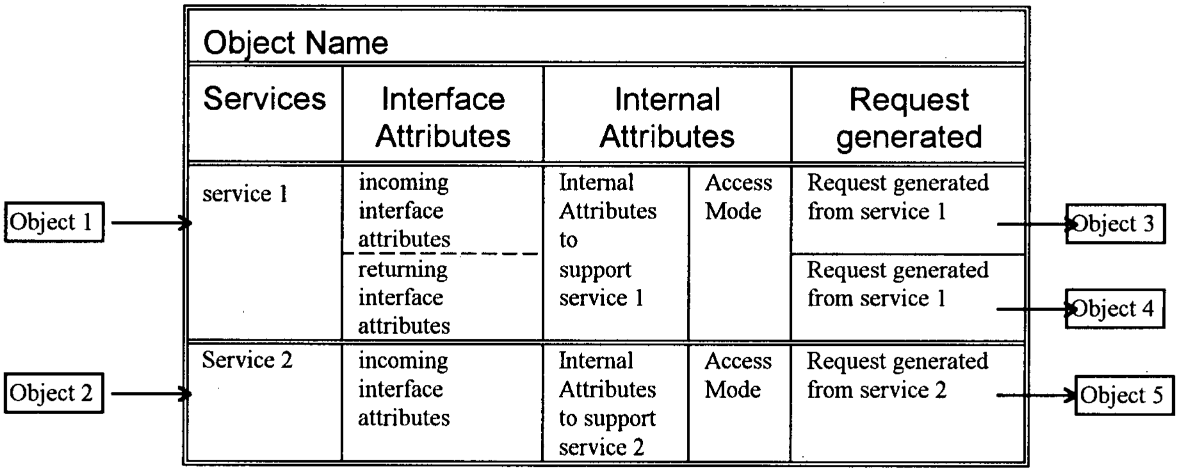
4.2.1. Internal Object Template

An object model that can be used to capture object characteristics is shown in Table 4-1. An example of an internal object in a student registration system is presented in Table 4-2. The object is identified by the object name. A set of services that the object needs to perform is listed in the *Services* column. For each service, we identify a set of information that must be passed along with the request that invokes the service. Such information is listed under the *Interface Attributes* column. In the registration system example (Table 4-2), a student can submit request to the registrar's office to enroll in a course. This request should be accompanied by information such as student id number, course number, and course section. They are listed under the *Interface Attributes* column above the dotted line as "incoming" interface attributes. When the service is accomplished, the object may return certain information to the requester. This information is listed below the dotted line to indicate that they are "returning" interface attributes. In the Registrar's office example, confirmation of registration is returned to the student.

In addition to the interface attributes, we also identify internal attributes that support the service under *Internal Attributes* column. In the case of the student registration system, information including student name, address, student number, registration status, major, academic standing, course restrictions and section availability are accessed when the registrar's office processes *register-courses* request. It may be useful to specify the access mode of the service to each group of internal attributes. In the Object Detail Table, "U" means use; "M", modify.

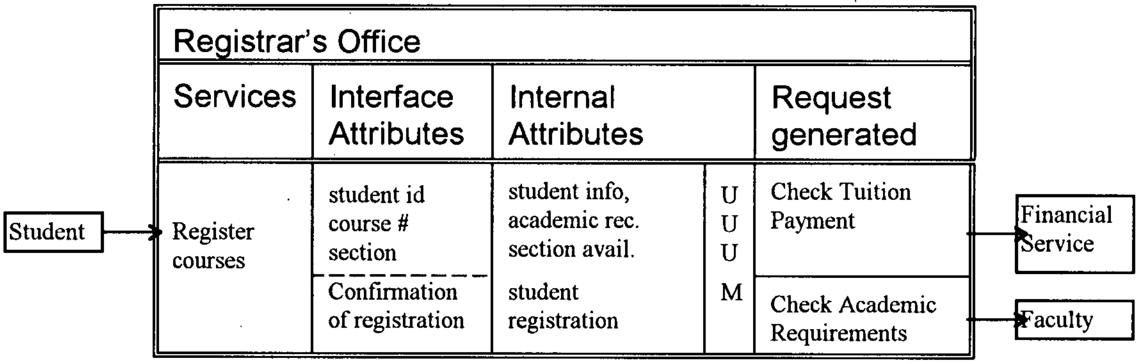
Next we analyze that actions the object takes to perform the service in order to determine the requests generated by the service. Some of these actions require the help (service) of other objects in the system. An object obtains assistance from other objects by sending requests to them. In other words, a service of an object may, in the course of the process, generate one or more requests to other objects. To process the *register courses* request, for example, assuming *registrar's office* object needs to check with the Financial Service Dept. to see if the student has paid certain amount of tuition fee. This is represented as a request, *Check_Tuition_Payment* generated by the *register courses* for a student service and sent to the *Financial Service* object (Table 4-2). Since the purpose of analyzing the service is to determine the request propagated from it, the analyst only need to study it to the level of detail at which requests to other objects are identified.

Table 4-1: Object Representation: Internal Object Template (IOT)



Object Name				
Services	Interface Attributes	Internal Attributes		Request generated
service 1	incoming interface attributes returning interface attributes	Internal Attributes to support service 1	Access Mode	Request generated from service 1 Request generated from service 1
Service 2	incoming interface attributes	Internal Attributes to support service 2	Access Mode	Request generated from service 2

Table 4-2: Example of the Registrar's Office Object



Registrar's Office				
Services	Interface Attributes	Internal Attributes		Request generated
Register courses	student id course # section Confirmation of registration	student info, academic rec. section avail. student registration	U U U M	Check Tuition Payment Check Academic Requirements

A service may generate zero to many requests to other objects in the system. Therefore, we may need more than one line in the *Request Generated* column for a single service.

4.2.2. External Object Template

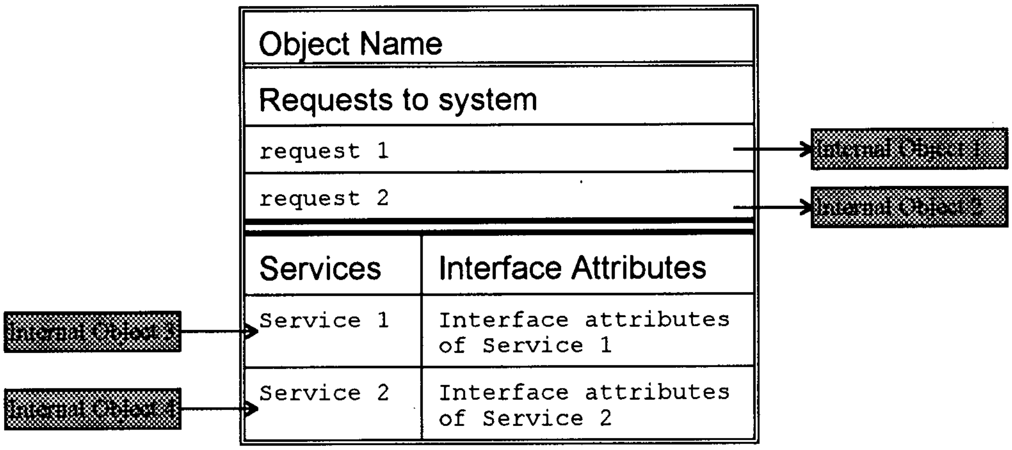
An external object is an object outside the system that interacts with the system. An external object can assume two roles: it can submit a request to the system, and hence become a *client*; or it can provide service to the system. These two roles are not mutually exclusive. In other words, an external object can submit requests to the system and provide services to the system. For example, a customer may order a product (therefore become a *client* of the system) and subsequently receive a bill for the project (to provide a service which can be named as "*pay-bill*").

For an external object, we are only concerned with its interactions with objects inside the system. Since an external object can serve as both initiator and terminator of request propagation process. we need to model two aspects of it: 1) the requests sent by the external object and, 2)the services it provides. In addition, we also want to record, for each service, the interface attributes necessary for the external object to perform the service. Same as those for internal objects, these interface attributes may include both 'incoming' and 'returning' attributes. An external object can be represented with a template presented in Table 4-3.

An external object is modelled differently than an internal object because

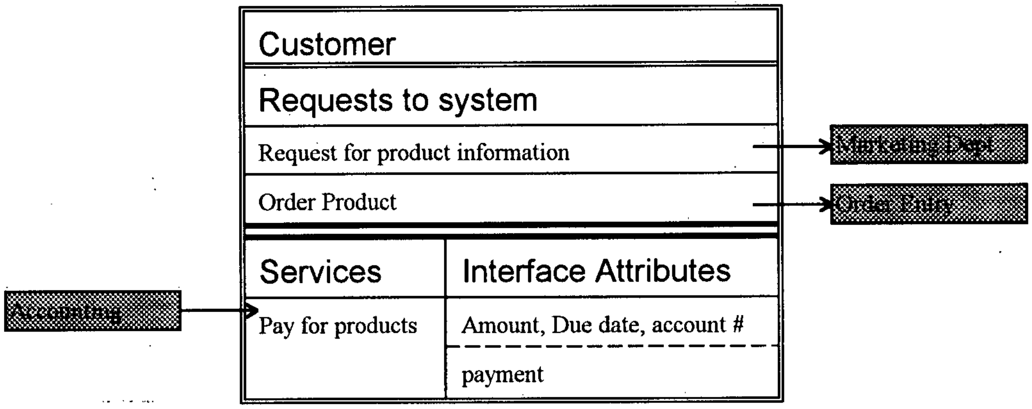
1. an external request is a starting point of the *request propagation* process; and
2. Analysts do not need to know any additional information about it except its interactions with the system.

Table 4-3: Object Representation: External Object Template (EOT)



An example of the external object (see Table 4-4) is a customer of a company mentioned in the previous paragraphs.

Table 4-4: External Object Example: a Customer

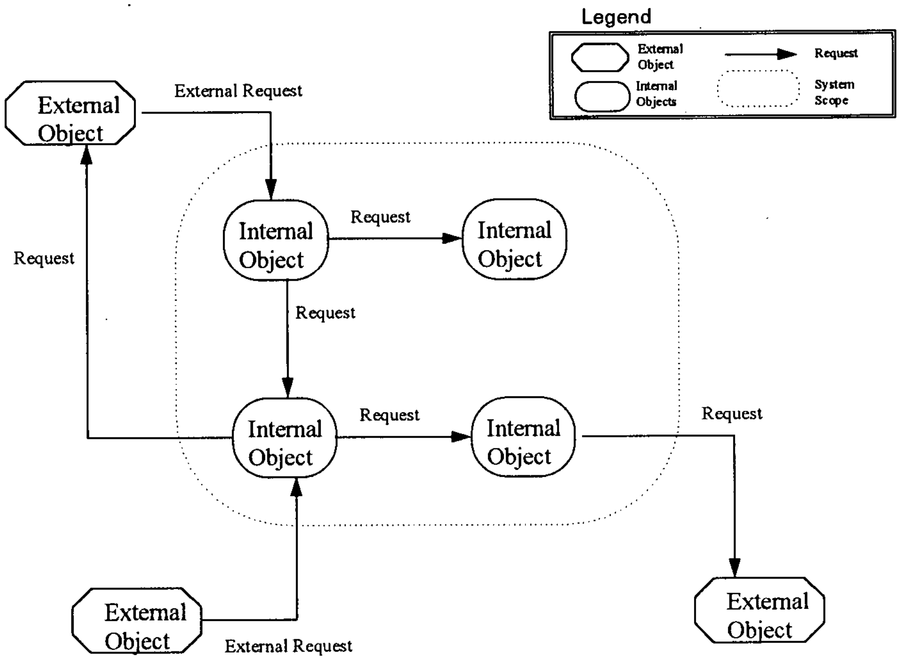


Customer	
Requests to system	
Request for product information	
Order Product	
Services	**Interface Attributes**
Pay for products	Amount, Due date, account #
	payment

4.3. System Representation: Object Communication Diagram

The entire enterprise can be modelled in an Object Communication Diagram (see Figure 4-1). This diagram concentrates on depicting the interactions among objects in the system. This diagram employs simple notation to represent objects. For each object, a separate Object Detail Tables can be constructed. The entire system model is the Object Communication Diagram along with all attached Object Detail Tables. The separation and connection between the Object Communication Diagram and the Object Detail Tables provide a powerful technique to effectively manage the complexity inherited in large organizations.

Figure 4-1: OOEM Object Communication Diagram



4.4. Tracing Request Propagation -- the Algorithm

This section presents the proposed algorithm for the OOEM method. The algorithm can be followed manually by a team of system analysts. We believe that it can also be incorporated into a high level CASE tool that could interview domain experts and create an enterprise model semi-automatically. Research in this direction is current underway at the Faculty of Commerce, UBC.

The purpose of the algorithm is to identify the following sets of objects. These sets are created and used by both algorithm.

- A set of *external objects* of the system (include *Clients* in *client-set*)
- A set of *objects* in the system (in *object-set)*
- A set of *services* (*service-set)* for each object,

- A set of *interface attributes* for each service (in *service-set*)
- A set of *internal attributes* for each object
- A set of *requests* and corresponding *respondents* generated by each service.

The proposed algorithm for identifying these sets consists of two parts:

- the main algorithm which iterates over all requests for all *clients* of the system (see Figure 4-2);
- the recursive algorithm which traces a single request (see Figure 4-3).

The main algorithm calls the recursive algorithm to trace a particular request from a *client*. It is more efficient to separate the algorithm into two parts since the number of times the recursive algorithm calls itself is unknown. In both figures, the bold typeface signifies that the steps are discussed in detail in the following subsections.

In these two figures, EOT refers to the External Object Template. IOT refers to the Internal Object Template.

Figure 4-2: Main Algorithm

```
Beginning of Main Algorithm
     Initialize all sets to empty
     Identify external clients of the system and put in
               client-set [Create EOT]
     do while client-set is not empty
          remove a client from client-set
          Identify all requests to the system from the
               removed client [Add to Requests in the EOT]
          For each of the request identified above, do
               Trace the request propagation process for the
                    request(Figure 4-3)
          end do
     end do
End of Main Algorithm
```

Figure 4-3: Recursive Algorithm for Tracing a Request

```
/* In Parameter: in-request: the request to be traced    */

Beginning of Recursive Algorithm
     Identify the object that will respond to the
          "in-request"(Step 2.1)
/* stop if the service is already identified by previous */
/* recursions                                             */
     If (the object is in the Object-set) and
        (service-set of the object contains a service
          responding to the in-request )
          Return
     end if

/* Stop if the object is an external object               */
     If the object is outside system scope
          If object not in External-object-set
               create new EOT
          end if
          If service-set of the external object does not
               contain in-request
               Add service to external-object
          end if
          Return
     end if

     If the object is not in Object-set
          Add the object to object-set [Create New IOT]
     end if

     Add service to the service-set of the object to
       respond to the "in-request"  [Add to Column 1 in IOT]
     Identify interface attributes associated with the
       request (Step 2.2) [Add to Column 2 in IOT]

     Identify internal attributes necessary to support or
       affected by the service (Step 2.3)
          [Add to Column 3 in IOT and specify use mode]
     Identify all requests generated by the service and put
       them in generated_request_set (Step 2.4)
          [Add to Column 4 in IOT]
     For each request in the generated_request_set do
          Trace request propagation process for the request
     end do
End of Recursive Algorithm
```

4.4.1. Important Steps in Main Procedure

Important steps in the algorithms are steps that require the analyst to find information from domain expert and make modelling decisions. In Figure 4-2 and Figure 4-3, they are represented by the statements in bold typeface.

4.4.1.1.Identify external clients

External clients are objects that submit requests to the system but are not part of the system. In OOEM, we only model the communication aspect of the external client, i.e., the requests they submit to the system. To model information processing details of an external client is out of the OOEM scope. After the scope of the system is established, identifying clients of the organization is usually achieved by asking domain experts questions such as:

- What objects outside the system scope interact with the system?
- Whom does the organization need to serve?

4.4.1.2.Identify requests from a client

This step requires that we focus on an individual *client.* We ask the question:

What are the requests that the client might submit to the system?

There are two types of request from a *client*:

- to obtain information, e.g., customer asks for price information and product descriptions;
- to trigger the system to perform certain actions, e.g., customer order products.

By accomplishing this step, we identify, for each *client*, a set of requests submitted to the system[1]. Next subsection discusses the recursive algorithm that traces the request propagation process triggered by each of the external requests.

4.4.2. Major Steps in Recursive Algorithm

This algorithm takes a request as a parameter, "*in-request*". To ensure a complete model of the propagation process, this algorithm is recursive, i.e., it may call itself with a different request. Since the recursive algorithm stops only when no more requests are generated or a request is sent to an external object, it covers the entire request propagation process. The parameter "*in-request*" can be an external request or a request from an object within the system. In the former case, the recursive algorithm is called from the main algorithm. In the latter case, it is called by itself.

There are two conditions in which the algorithm terminates and returns control to the calling algorithm:

1. The first condition is satisfied when the object is already identified and the service that answers the *in-request* is already included in the model. This means that any further propagation of this *in-request* is already modelled by previous recursions. Tracing it any further can only result in duplication.

2. The second condition is when the request is sent to an external object. We model an external object only in terms of its interactions with the system, i.e., sending or

[1] The term "system" refers to the organization not the computer systems that might be used to support the organization.. The term is used synonymously with "enterprise" or "organization" in this context.

receiving requests. When the receiver of the *in-request* is an external object, only the interface attributes should be captured. Therefore, a model of an external object only includes requests, services, and associated interface attributes.

The following subsections discuss how an analyst can achieve each important step in this algorithm. These steps are identified by the bold typeface in Figure 4-3.

4.4.2.1.Step 21 Identify the object responds to the "in-request"

An analyst can ask domain expert "Who is responsible to the *in-request*?". This question identifies the receiver of the *in-request.* The answer is usually easy to find. However, sometimes additional thought is necessary for accurate identification. In particular, correct answers depend on the level of granularity at which the analyst wants to model the system. For example, when a customer sends an order to a company, should the analyst treat the order-entry clerk as a receiving object or the order entry department as a whole the receiver? Ultimately this is the decision of the analyst. If the analyst is interested in the entire organization before going into details of a single component, then it is preferred to model the order entry department as the responding object to the request. Otherwise, if the detailed operations of the order entry department are under study, then it is probably necessary to model the order-entry clerk as an object.

4.4.2.2.Step 2.2 Identify interface attributes for request

In the process of identifying these interface attributes, analysts any ask domain experts questions such as:

- What does the object need to know from the requesting object in order to fulfill its responsibility? Answers to this question determines the 'incoming' interface attributes.
- What information does the object need to send back to the requesting object? Answers to this question determines the 'returning' attributes.

4.4.2.3.Step 23: Identify internal attributes for service

As discussed earlier, internal attributes represent its state of the object. For example, information about courses that student can register is the knowledge of *Registrar's Office* and is necessary for it to provide *register-courses* service. Therefore, when analysts study the effect of *register-courses* request, a *course-information* could be added to the internal attributes of the *Registrar's Office* object, if it is not already included. To determine the internal attributes associated with the service in question, the analyst can ask question such as:

- Besides the interface attributes determined in Step 22, what does the object need to know in order to perform the requested service?

Sometimes analysts are faced with the dilemma of deciding to which object a piece of information should belong, since this can be potentially modelled in two ways:

1. Information is internal attribute of the object under analysis, or alternatively,
2. Information is internal attribute of another object which provides services of accessing the information.

For example, assuming tuition payment status is checked before registrar's office register courses for a student. Tuition payment information can be modelled as

1. internal attribute of the *registrar's office*, or
2. internal attribute of the *Financial Service* object which provides access to this information to *registrar's office* object.

Rule # 5, the attribute inclusion rule provides guideline for solving this problem. In practice, analyst can ask the following questions:

- Who has the most logical ownership to this information?

In the example, *Financial Services* object with the function of administering most of the financial transactions in a university is the logical owner of the *tuition payment* information.

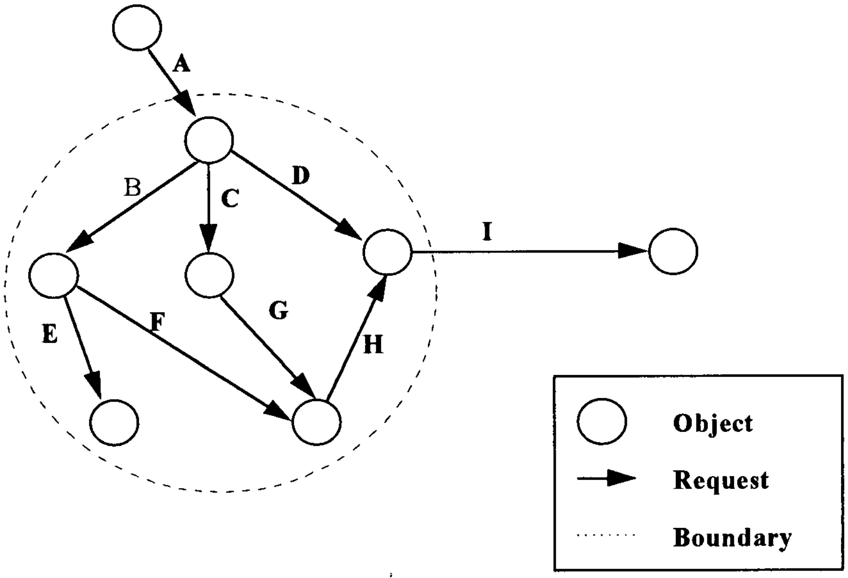


Figure 4-4: Request Propagation

4.4.2.4.Step 24: Identify requests generated by a service

In order to identify requests spawned from a service, we need to look into the details of processing activities performed in a service. There are two ways to trace the request propagation process for an external request:

1. identify all requests generated by an "*in-request*" and the services that are triggered by it, before investigating each request. This can be called "breath-first" approach, since it works with the same "level" of the request propagation stage before moving to the next. In Figure 4-4, this approach traces requests in the sequence of **ABCDEFGHI**.
2. trace an individual request first before investigating other requests generated by the service. This can be called "depth-first" approach. In Figure 4-4, the tracing sequence according to the this approach is **ABEFHICGD.**

The latter approach requests analysts to switch focus from one object to the next. This may cause difficulties, especially when they analyze large, complex organizations. We therefore recommend the first approach in the recursive algorithm. This algorithm helps analysts to gain a fuller picture of an object service before move to the next.

To perform this step, analyst may ask the domain experts the following questions:

- How can the service be fulfilled? Or What steps are necessary to accomplish the service?
- Which of these steps are the functions of other objects and, therefore, should be delegated via requests to them?

- Besides the interface attributes associated with the request and the internal attributes of the object, what additional information must be obtained from other objects? Who has the information?

These questions identify requests for information and the respondent.

4.5. Apply Rule #6 and #7

After applying the request propagation algorithm, analysts create a model of the enterprise which captures its dynamic interactions with its environment and among its components. This model is represented in the Object Communication Diagram and associated Object Detail Tables for each object. Now, analyst can apply Rule #6 and Rule #7 to organize the objects and object types in the model into whole-part and/or generalization and specialization structure.

In applying Rule #6, analyst investigate the connections between objects and ask the question:

- *Is one object actually a part of another object even thought the case description and domain experts refer to them separately.*
- *If this is true, does the composite object provide services not provided by the total of its parts?*

If the answer to the latter question is true, then the composite object should be included in the model. Otherwise, it should be eliminated. In addition, analyst should also ask"

- *Is there any of the component objects that only provides service(s) provided by the composite object?*

If this is true, those component objects whose services is redundant should be eliminated from the model. It should be noted that this consolidation of component objects into composite objects may changes the models of some other objects that interact with the component objects. For example, if object A interacts with two objects (B and C) which are the components of one composite object (D). When the object B and C are consolidated into object D, the interaction between object D and object A maybe changed. As a result the Object Template for A maybe changed.

When apply Rule #7, an analyst looks for the objects with common services. If this situation arises, a new and more general object class which provides the common services should be added to the model. Common services in the super-class should be eliminated from each sub-classes.

4.6. The Algorithms and the Modelling Rules

The algorithms provide a specific technique or strategy to apply the OOEM modelling rules introduced in the previous chapter.

Rule #1. Analysts identify the scope of the enterprise by separating the system and its environment. More specifically, they identify all external objects and analyze the external requests they submit to the system.

Rule #2. object identification rule, is satisfied since the recursive algorithm adds a new object in the model only when a request is sent to it. This insures that each object in the model provides at least one service.

Rule #3. The recursive algorithm adds a service to an object only when the object receives a request that cannot be accomplished by existing services. In other words, a service is included in an object iff it is invoked directly by at least one request.

Rule #4. An internal attribute is identified by when it is needed to support a service of the object. Hence, all internal attributes must be used or affected by at least one service of the object. An interface attribute is shared by both communicating objects and therefore 'known' to both objects. In addition, the 'incoming' interface attribute is always used by the receiver to perform the service. The 'returning' interface attributes is always affected by the result of the service. Both types of interface attributes are represented in the Object Detail Template of the receiving object. Therefore all interface attributes and internal attributes satisfy Rule #4, the attribute inclusion rule.

Rule #5. Since all attributes of an object are identified and included to support the services provided by the object, every one of the attributes in the system belongs to an object. To ensure that each attribute belongs to one object only, the analyst can consider the question of *"who is the logical owner of the attribute"*. Other objects can access the information of this attribute via the services provided by the custodian of the attributes.

Rule #6 & 7. These rules are used to organize the objects identified by the algorithm into whole-part and generalization-specialization structures. Since these two rules are concerned with the structural aspects of the system, while the first five rules deal with the dynamic aspect, they are applied after the algorithm has been performed.

4.7. Chapter Summary

This chapter presented the OOEM method. It included the representation of both internal and external objects and the algorithm for constructing an OOEM model of an organization. Essential information about an internal object, including services, interface attributes, internal attributes and request generated from each services is captured in the Internal Object Template. The External Object Template describes the communication aspect of an external object.

The process of constructing an OOEM model is formalized in the Request Propagation Algorithm. It has two parts. The main algorithm iterates over all external requests. The recursive part traces all requests triggered by a single external request in a "level-first" fashion. To assist an analyst to perform important steps, possible questions to domain experts are also discussed for each of the steps.

5. A Case Study

Previous chapters presented the theoretical foundations and the practical techniques of OOEM. Although examples are given to illustrate OOEM concepts and method, no systematic demonstration of the entire method was presented. To remedy this lacking of practical illustration, this chapter presents a reasonably realistic case, the ABC company Case. This case is adapted from [Thierauf, 1986 p51-57]. In order to keep the length within reasonable limit, detailed information are abstracted and omitted.

The ABC company, a hypothetical manufacture company, is the target company for our enterprise modelling. This section describe the functions of major components of the company. This description leads to a top level Object-Oriented Enterprise Model of the ABC company, which is presented in the next section.

5.1. ABC company case

In a manufacturing environment, at least two basic sub-systems are considered as vital to the operation of the organization: marketing and manufacturing. Moreover, these systems must be supplemented by supportive systems which include R & D, engineering, purchasing, accounting and finance, physical distribution, and personnel. General functions of each system are described below.

5.1.1. The Marketing system

The primary job of the marketing system is to contact potential customers and sell products through salespeople, distributors, advertising, and special promotions. When a customer orders a product, the order section of the marketing department creates an order. The department first checks with the warehouses to see if the required quantity of the product is available. If so, the order is then sent to the nearest field warehouse who is going to arrange the shipping of the product to the customer. If the product is not available in stock, a production order is issued. In some cases, the marketing department may need to check the credit status of certain customer by requesting the accounting department to approve the purchase order before deciding to ship the product.

5.1.2. The Manufacturing system.

The manufacturing of the finished products for the ABC Company involves many steps. Not only must plant, equipment, and tools be provided, but also appropriate personnel be hired and trained to utilised the manufacturing facilities. Row materials and goods-in-process must be available as needed. Production must be planned, scheduled, routed, and controlled for output that meets certain standards. The company's products can be produced in anticipation of demand or on receipt of customer orders. If goods are being produced to order, a sales order may be the production ordering. The usual arrangement is to have the production planning and control department initiate actions on factory orders.

Manufacturing materials can be obtained from inside or outside the company. If a purchase requisition is produced, it is forwarded to purchasing for vendor purchase. If, on the other hand, materials are available from the stock control department, a materials requisition is prepared.

5.1.3. The Research and Development, and Engineering Systems.

Often, the marketing system receives inquiries regarding a new product from certain customer. It will initiate a research and development order (R&D order) for its development if marketing prospects look promising. Similarly, the marketing research section may have its own ideas on what products should be developed. This too, can result in an R&D order. Before manufacturing operations can commence, it is necessary to design the products as agreed upon by the marketing section in conjunction with research and development. This means designing the product from scratch. The engineering blueprints (manufacturing specifications) are forwarded to the manufacturing system.

5.1.4. Purchasing and Inventory Systems

The purchasing function is concerned with procuring row materials, equipment, supplies, and utilities, as well as other products and services required to meet the ABC Company's needs. The procurement process begins with the completion of the purchase requisition. Based upon the purchase requisition form, a buyer locates the appropriate contract and orders from the suppliers. If a contract has not been signed for the desired goods or services, the buyers sends a request for quotation to prospective vendors. Once the outside supplier has been determined, based on an analysis of total delivered cost, quality, and service, a purchase order is sent to the vendor. It contains the items to be shipped, price and specifications, terms, and shipping conditions. The original is forwarded to the vendor while duplicate copies are distributed to purchasing, receiving, stock control, preparing departments, and accounts payable.

As soon as goods are received from suppliers, they are checked and verified against the copy of the original purchase order by the inventory or receiving department. Once the receiving clerk is satisfied that the goods correspond to those on the purchase order, the individual prepares a receiving report, noting any discrepancies between the order and actual material received.

Goods are delivered by the receiving department to the stock control department. In the stock control department, the transfer-in of materials from an outside vendor is documented by the receiving report. Stock control has the added function of replacing stock when it reaches a minimum level, often referred to as a reorder point. The stock clerk prepares a purchase requisition for the specific materials and forwards it to the purchasing department.

5.1.5. Physical Distribution System

Once the customer order has been manufactured and finished goods have become available in the field warehouse, they are ready for shipment. The finished products must be packed, labelled, and transported to the customer. The order and shipment processing form which authorises shipment is delivered in advance of the goods. If pick-up is made by the customer, receipt of goods will be acknowledged by signing a copy of shipping order which is then filed in the ABC Company's shipping office.

Shipments that are made via public carriers must be accompanied by a bill of lading which is actually a contract between the consignor and the carrier. It serves as a description of the shipment and its contents.

5.1.6. Accounting and Finance Systems

After the required business operations have been performed by the shipment of finished goods, the accounting department must prepare customer invoices. The first copy is sent to the customer while remaining copies are distributed to accounting department's accounts receivable file. Payments received from customers are deposited in the ABC Company's bank account. These payment are recorded in the case receipt journal as documented evidence of their receipt.

5.2. An OOEM of ABC Company

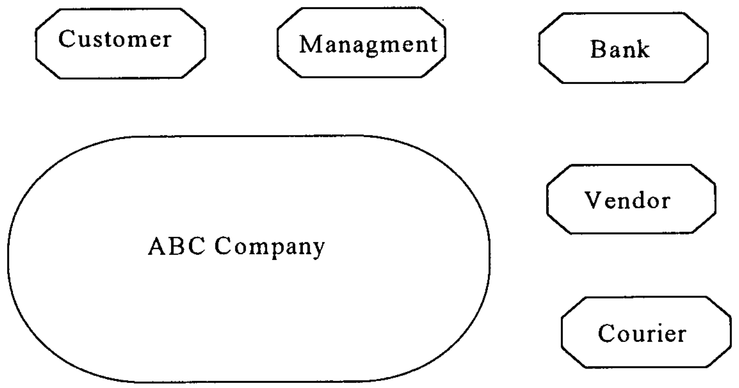
This section describes in detail the procedure of building an OOEM for the ABC company. We follow the algorithm presented in the previous chapter. The main purpose of this section is to illustrate and clarify by example how our method can be used in modelling reality.

5.2.1. Identify System Clients and External requests:

To begin modelling, we first identify the external objects of the system. An external object is an object outside the scope of the system but interacts with the system. In the case description, it is mentioned that customers, vendors, bank, and public carrier interact directly with the system. In addition, although not mentioned directly in the case description, management plays a very important role in setting directions and objectives for many departments. In this analysis, we choose to treat senior management as an external object to the production system which is the scope of the model[1]. By now, we can produce a system level object diagram in which the entire system is modelled as a single object (see Figure 5-1). In Figure 5-1, an oval box represents the system scope and polygon box represent external objects.

[1] As defined in Rule # 1, the portion of reality we wants to model is the production system, not the entire organization, which includes senior management. If we choose to model the entire enterprise per se, then the management is an internal object and shareholders represented by the board are external objects. Obviously, the scope decision must be made by the domain experts.

Figure 5-1: Identify the System Scope



5.2.2. Identify External Requests

For each of the external clients, we identify the requests they submit to the system. Although interactions with the system can take two forms; submitting a request to the system or responding to a request from the system, we are only concerned with the former type of interactions. For example, "customer orders a product from the company" is an external request, whereas "field warehouse requests the public carrier to ship product to a customer" is not an external request, though it is a request to an external object. The reason we are only concerned with the former is that such requests are the start point of the request propagation process. By tracing the request propagation process, we can build a complete model of all relevant components of the entire system.

In practice, it is not necessary to identify all the external requests first. In fact the analyst can start with tracing the most important requests, i.e., the major tasks that the system must offer to its clients. By doing so, she/he can avoid the potentially confusing situation of switching attention from one object to another, especially in large systems which may contain many objects. This is one of the advantage of OOEM. Users can start with one external request and understand the details of that perspective of the system without being distracted by other views of the system.

5.2.3. Identify Internal Objects

For illustration purpose, let's start with the request from the customer who orders product from the company. This is the primary function of the company. We ask the question:

Which object in the system responds to the request?

The case description indicates that the request goes to the *Marketing department.* According to the object identification rule, *Marketing department.* is the first object we identified in the system, and the *Order product* is the first service of the *Marketing department*, according to the Service inclusion Rule. Now we can create an Internal Object Table based on internal object template and fill the service column with *order product.*

5.2.4. Identify Interface Attributes

Next let's look at what interface attributes must accompany the request *Order Product.* In other words,

What information must the object must get from the requester in order to provide the service?

This question will prompt the analyst to pose detailed questions to the problem domain experts. Such information may include customer name, address, account number, product specifications, sales terms, desired ship date, payment method, quantity, and price. This product order information becomes the interface attributes of the *Marketing department.* object associated with the service of *Order Product.* Access modes for each of the information items are also determined. In addition, we also identify that information that must be returned to the customer as part of the interface attributes. In this case, "confirmation of sales and expected delivery dates" are returned to the customer and therefore included in the interface attributes associated with *order product* service of the *Marketing department.*. These attributes are added to the "Interface Attributes" column in the form.

5.2.5. Identify Internal Attributes

Next in order to identify the internal attributes that are used to support the *order product* service, we ask the question:

What does the object need to know and remember in order to perform the service?

To successfully carry out the order product action, the *Marketing department.* must remember the customer order information, and the decisions and actions taken for each customer order. Such information can only be accessed by the Marketing department. So they are modelled as the internal attributes of this object.

5.2.6. Identify Request Generated and Request Propagation

Next, we look at the process of providing this service in greater details and ask the question of

- *Which parts of the activities of this service should be delegated to other objects.*
- *What requests to other objects are triggered by this service?*

In the effort to provide the order product service, *Marketing* sends four types of requests to other objects in the system.

First, for some customers, the marketing department needs to check their credit history before anything else can be done. If the credit history is not acceptable, the request is denied. To check the credit status of a customer, the marketing department sends a request to the accounting department where customer's payment history is kept. Credit history information is the internal attributes of the accounting department according to the attribute ownership rule. By now we have witnessed the "request propagation" process by which one request to an object, *Order Product*, triggers a request (*check customer credit history*) to another object *Accounting department*. We have just identified another object *Accounting department* and the responding service, *Check Customer Credit History*. In our analysis of the *Marketing* department., we record that this object sends a request of *Check Customer Credit History* to the *Accounting* department. Associated with the request are the necessary information from the *Marketing* department to the *Accounting* department to carry out the request. These include customer's name, account number, dollar value of the order etc. They are recorded under the interface attributes attached to *Check-customer-credit* request of the *Accounting* object.

Second, if the customer's credit is approved, the *Marketing* department should *check product availability* with the warehouses.

Thirdly, if the products are available, *Marketing* acknowledges the customer and request the *Warehouse* to *ship products* to the customer. In order to decide which warehouse to send the shipping request, marketing department. need to access the *customer's nearest warehouse* information. This information is also included in, and therefore the internal attribute of the *Marketing* department.

Finally, if the product ordered by the customer is not available on stock, a production order is issued by the *Marketing* object. This is represented by a request *Produce Products* sent from *Marketing* object to the *Production* object.

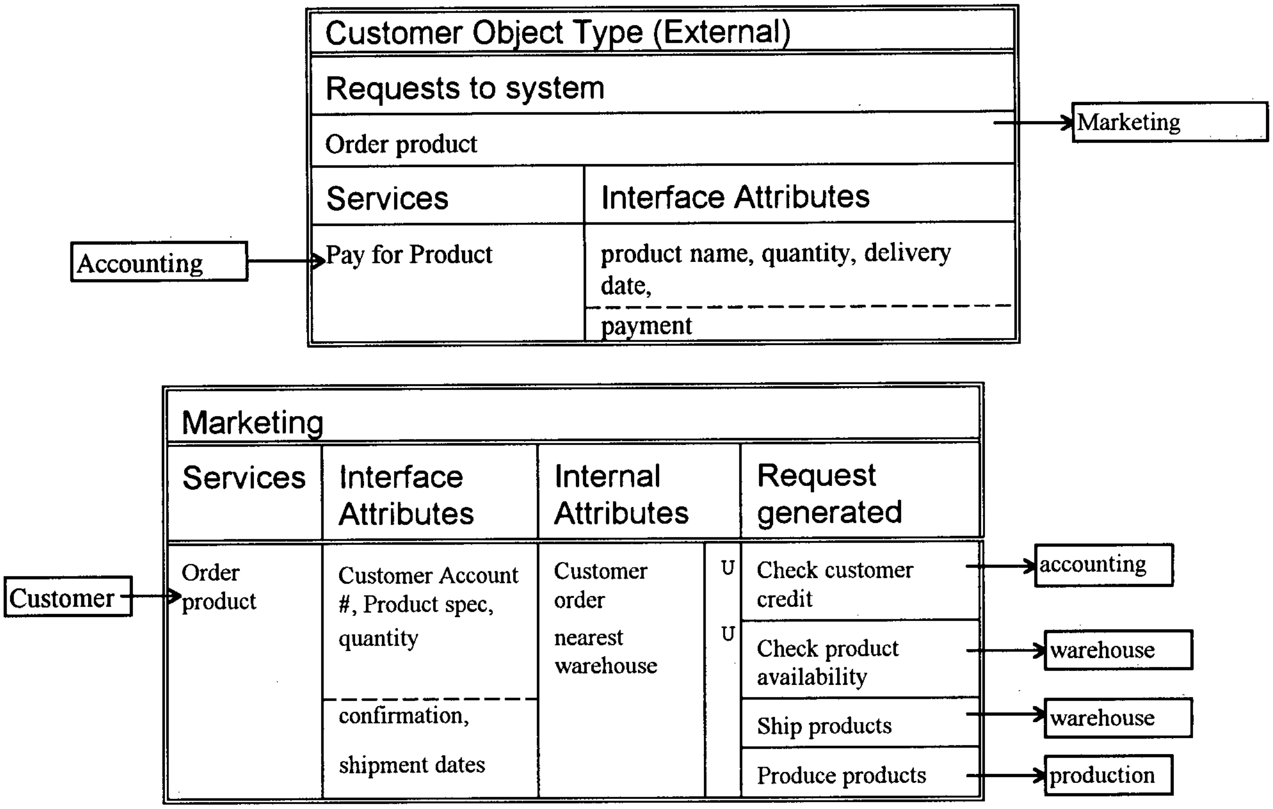
We have just witnessed the request propagation process by which a single request (*Order product)* to an object *(Marketing)* generates four requests to the rest of the system. In turn, we can trace these four requests using the recursive algorithm.

The tracing stops only when an object can provide the service all by itself, without any assistance from other object. An example of this is the *check customer credit* service in the *Accounting* object.

Object models for *Customer* and *Marketing* after the external request *order product* is traced are presented in Table 5-1. An object communication diagram after the *order product* request is completely traced is shown in Figure 5-2. A complete model after all external requests are traced can be found in Appendix II.

5.2.7. ABC Case: A Customer-Order-Product Perspective

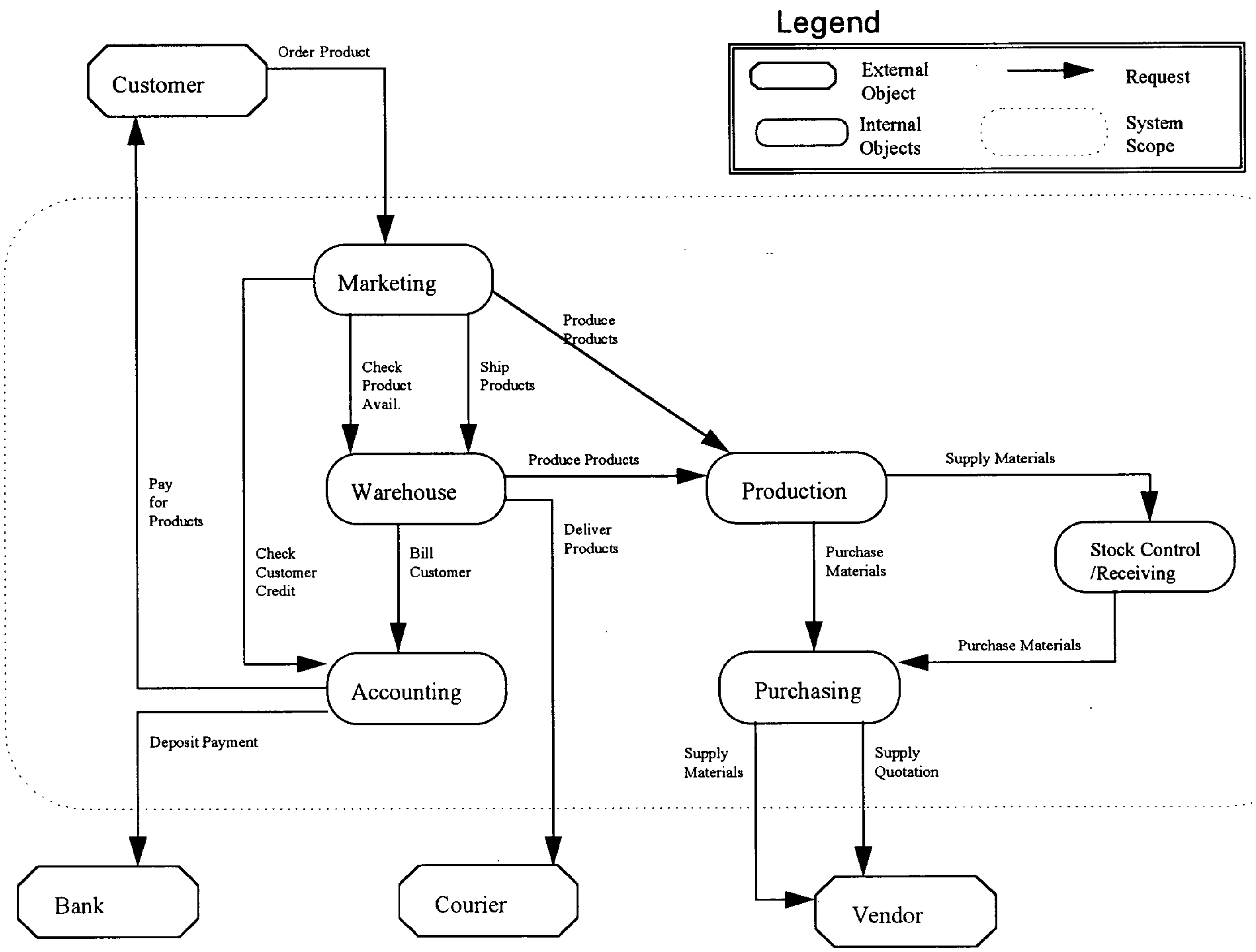
Table 5-1: Object Models after Request Propagation Analysis of *'Order Product'*



Customer Object Type (External)	
Requests to system	
Order product	
Services	Interface Attributes
Pay for Product	product name, quantity, delivery date, payment

Marketing				
Services	Interface Attributes	Internal Attributes		Request generated
Order product	Customer Account #, Product spec, quantity	Customer order	U	Check customer credit
		nearest warehouse	U	Check product availability
	confirmation, shipment dates			Ship products
				Produce products

Figure 5-2: Object Communication Diagram: Request Propagation Analysis of *'Order Product'*



6. Comparison of OOEM with Other OOA Methods

The purpose of this chapter is to examine the contributions of OOEM by comparing it with other OOA methods. There exist many object-oriented analysis methods. The 1994 survey by the Object Management Group (OMG) included 22 object analysis and design methods [Hutt, 1994]. Here, we do not intend to provide a comprehensive comparison with all OOA methods, which is impractical in the thesis. Instead, we selected three methods to represent general trends in this field. These three methods are among the best known OOA methods. They include Coad and Yourdon OOA method, Jacobson Objectory method for business modelling, and Rumbaugh's OMT method. For each method, we provide a summary of the basic concepts and modelling procedures. Then we discuss the similarities and differences between the method and OOEM by comparing analysis results on a specific case.

6.1. Comparison with Coad and Yourdon OOA Method

Coad and Yourdon OOA Method is presented in a series of books by the two authors [Coad. et. al., 1990, 1991, 1995]. After a brief summary of the method, we compare it with OOEM by studying a case found in [Coad et. al. 1995].

6.1.1. Summary of the Coad and Yourdon Methods

Coad and Yourdon's modelling constructs include Class & Object, Attributes, Services, and class relationship. They define object as "a person, place, or thing" and class as "a description of what object knows and does." They also distinguish four types of object-class: Problem Domain Components, Human Interface Components, Data Management Components and Task Management Components for real-time systems. An attribute is something that an object knows and a service is something that an object does [Coad et. al. 1995 p491,495]. Relationships between classes are classified into association, Gen-Spec and Whole-Part relationship. Association is used to "tie together certain things that happen at same point in time or under similar circumstances" [Coad et. al. 1991 p 15].

Their method suggests the following activities for building a multi-layered object model for the problem domain.

1. Finding Classes & Objects
2. Identifying structure
3. Identifying subjects
4. Defining attributes
5. Defining services

For each of these activities, Coad and Yourdon recommend a set of modelling heuristics which they termed Patterns and Strategies. These strategies include "Where to look for", "What to look for", and "What to consider and challenge." Their newest book [Coad and Yourdon 1995] contains 148 strategies and 31 patterns to be used in a variety of business modelling situations. The following subsection uses a case to compare the two methods.

6.1.2. Order centre case study: Comparison of methods

This case is adapted from Coad [1995 p. 151] and modified for the purpose of comparison.

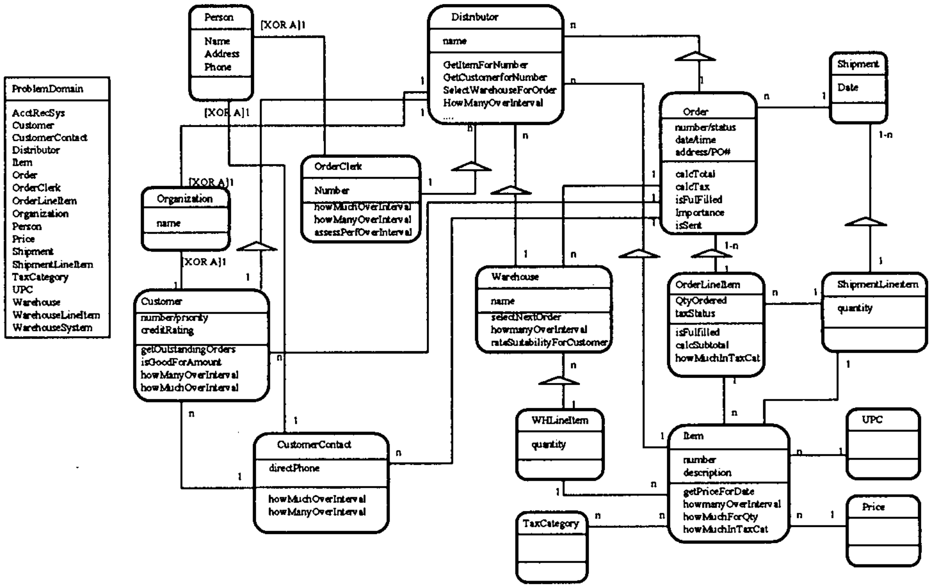
The order centre takes orders from customers and processes them. The responsibilities of the order centre include:

1. *Take an order over the phone and write it down on an order form.*
2. *Total the order and tell the customer how much the order will cost.*

3. *Get a purchase order (PO) number from the customer and write it down on the order form. PO number is needed to make sure that there is some corporate commitment.*
4. *Decide to which warehouse to send the order. The warehouse is selected to minimize the shipping costs. Usually the warehouse closest to the customer is selected. The warehouse is selected when the centre takes the first order from the customer. Warehouse uses public carriers to deliver the products.*
5. *Fax the order to the warehouse for shipping. For each item shipped from the warehouse, get confirmation via fax.*
6. *Send the order form with the amount shipped to the accounts receivable department. They key it into an account receivable system and collect the amount from customer.*
7. *Answer customer's inquiry about their order status.*

Analysis result using Coad and Yourdon method is presented in Figure 6-1. A rounded box denotes an object-class with name, attributes and services in three sections of the box. Lines connecting object-classes represent associations. Lines with a small triangle stand for whole-part relationships between object-classes. Cardinality of association is also specified by the numbers on both ends of the lines. OOEM model for this portion of the organization is presented in and Table 6-1 contains an example of object detail table. Next subsections compare the two approaches.

Figure 6-1: Coad and Yourdon Object Model of Order Centre



Source: [Coad 1995 page 192].

Figure 6-2: OOEM Object Communication Model for Order Centre

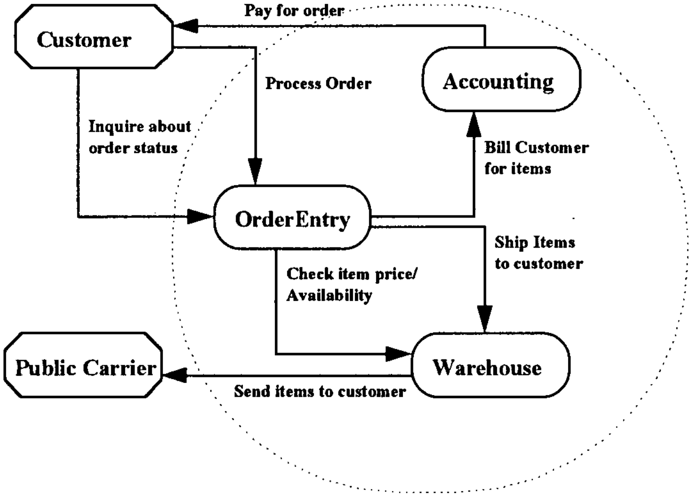
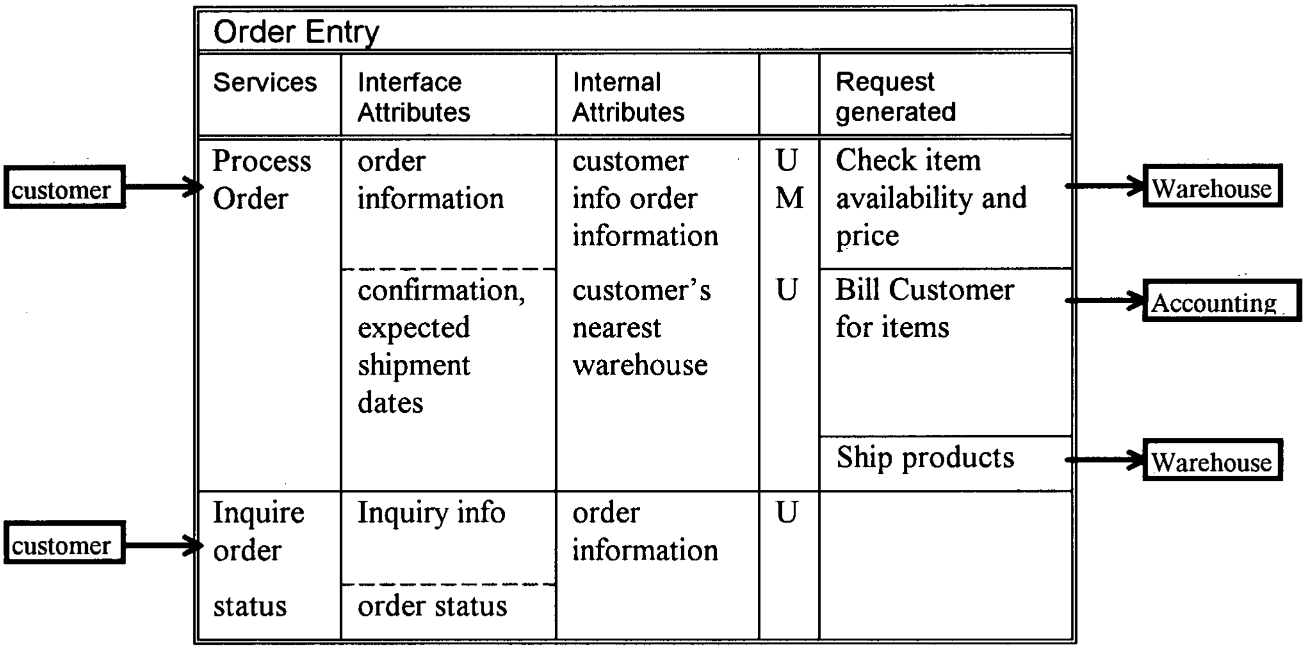


Table 6-1: OOEM Object Models for Order Entry



Order Entry				
Services	Interface Attributes	Internal Attributes		Request generated
Process Order	order information	customer info order information	U M	Check item availability and price
	confirmation, expected shipment dates	customer's nearest warehouse	U	Bill Customer for items
				Ship products
Inquire order status	Inquiry info order status	order information	U	

6.1.3. Mapping Analysis Results

One important difference between the two methods is reflected in the number of objects included in the analysis results. There are 16 object-classes in the Coad and Yourdon's analysis result and only 5 in the OOEM model. The distinction will be more apparent if we map the 16 objects onto the 5 object in the OOEM model (see Table 6-2).

Moreover, the Coad and Yourdon's model does not express some important aspects of the business process. First, it does not reveal the external object which triggers and/or terminates a business process. From studying Coad and Yourdon's diagram, readers cannot understand readily the scope of the organizational activities. Second, is does not show the dynamic interactions among objects within an organization. Most objects in their model are given some services. However, the purpose of the services and who uses these services are not clear.

The rest of this subsection discusses the difference between Coad and Yourdon's model and OOEM from more specific aspects. These include the modelling constructs, the nature of the models, the domains of the models, and what to be included in the models. A brief summary of the subsection is presented at the end.

Table 6-2: Mapping Objects in Coad and Yourdon's model onto OOEM: Order Entry Case

Objects in Coad and Yourdon's model	Objects in OOEM model
Person, Organization, Customer (partial) CustomerContact Order OrderLineItem Shipment ShipmentLineItem OrderClerk (partial)	Order Entry • *Order Entry must remember information about Person, Organization, Customer and CustomerContact in order to perform its services. Therefore customer information are encapsulated in Order Entry.* • *Part of the information about Customer, e.g., credit limit and payment method is encapsulated in the Accounting object.* • *Order and OrderLineItems are also encapsulated in Order Entry.* • *Shipment information describes the status of an order and therefore is treated as part of order information.* • *Here, the OrderClerk refers to information about the order clerk. Clerk Number may become part of the Order Information.*
WHlineItem Item, UPC Price TaxCategroy Distributor Warehouse	Warehouse • *WHLineItem, Item, UPC, Price and TaxCategory are information about inventory in a warehouse and therefore encapsulated in the Warehouse (Inventory Information).* • *Information about distributors are knowledge of the Warehouse object who needs to provide the service of shipping product to customer.*
Customer (partial)	Accounting • *Customer financial information e.g., credit limit, and payment method, are encapsulated in the Accounting object.* • *This object also includes customer payment status which is not available in the Coad and Yourdon's model*
OrderClerk (partial)	<Not included in OOEM> • *The assessment of OrderClerks performance is not part of the business process triggered by a customer order. Therefore, HowMuchOverInterval, HowManyOverInterval, and assessPerfOverInterval are irrelevant to the purpose of this case and not included in the OOEM model.*

6.1.4. Comparing Modelling Constructs

Having compared the analysis models of the two methods for the order entry case, we can compare the modelling constructs in Coad and Yourdon's method onto those used in OOEM. Such a mapping is shown in Table 6-3.

Table 6-3: Comparing Modelling Constructs: Coad and Yourdon vs. OOEM

Coad and Yourdon Modelling Constructs	OOEM Modelling Constructs
Object-Class (Problem Domain Components)	Object sometimes Internal attributes of object
Attribute	Internal attribute
Attribute for transaction object (e.g., order)	interface attributes
Service	Service (Sometime not included in the OOEM)
Generalization-Specialization structure	Generalization-Specialization Structure
Whole-Part structure	Composite Object

As discussed previously, not all objects identified by the Coad and Yourdon's method qualify for objects in OOEM. In the Coad and Yourdon's model for the Order Entry case, objects such as *Person* and *WarehouseLineItem* are referring to the entities whose information must be kept somewhere in the system. Since these entities do not participate in business activities, and hence do not provide any service, they are not qualify for object in OOEM. Instead, in OOEM, information about them is modelled as Internal Attributes of some objects, i.e., information about *person* is managed by *Order Entry* object and information about warehouse items is the internal attribute of *Warehouse*. OOEM rules clearly exclude objects without services or attributes. Since Coad and Yourdon's method does not provide rules to identify objects, their method is very easily confused with data modelling.

Attributes in the Coad and Yourdon's method can be mapped onto attributes in OOEM. However, attributes in OOEM encompass a wider range of domain. For example, as discussed above, some objects in the Coad and Yourdon's model are modelled as Internal attributes in OOEM. In addition, the OOEM Interface Attribute, which provides much detailed semantics for request, is not available in the Coad and Yourdon's method.

The constructs related to services, Gen-Spec, and Whole-Part structures are quite similar between the two methods. To manage complexity, Coad and Yourdon use the "Subject" construct while OOEM recognizes that an object can be a composite one consisting of several other objects at lower level of granularity.

Coad and Yourdon provide modelling guidelines comprised of a sequence of activities. These guidelines are enriched by their consulting and modelling practices based on which they summarized a set of strategies and patterns [Coad, et. al., 1995]. In contrast, OOEM is based on a set of well-defined modelling rules derived from ontological model of the world. This set of modelling rules enables the development of the algorithm and object detail table (template) as presented earlier.

6.1.5. Nature of the Model: Dynamic vs. Structure.

The Coad and Yourdon object model emphasizes on the structural connections between objects. Associations are classified into "Gen-Spec", denoted by a small semicircle on the line, "Whole-Part", denoted by a small triangle on the line, and unspecified association, which means that relationship between the two object-class exists but not specified and is denoted by a line. Cardinality of association is also specified, presumably for integrity checking.

OOEM depicts dynamic interactions between domain objects using the modelling constructs of a request. Message connections are the most important relationships between objects in OOEM. Coad and Yourdon approach put much greater emphasis on structural aspect of the model.

Although Coad and Yourdon's method also shows the message connections between object-classes, it does no express explicitly the linkage among the requests, the services triggered by these requests, the associated interface and internal attributes and the requests propagated from theses requests.

6.1.6. Domains of the Models: Enterprise vs. Information Systems

Study of the analysis results of the two methods in this case shows that some of the objects in Coad and Yourdon model do not reflect business activities. All of the four kinds of objects: Problem Domain Component, Human Interface Component, Data Management Component, and Task Management Component, are models of information system entities that can be designed, and implemented. Although Coad and Yourdon attempt to separate OOA and OOD, their OOA constructs and method is certainly influenced by implementation concerns.

The goal of OOEM, on the other hand, is to model essential business activities in an organization. An OOEM illustrates how a business process is initiated, performed by various organizational units, and eventually terminated. It also specifies the services that must be provided by each object. However, it does not specify how the services should be provided, leaving room for each object to search for the best way of performing those tasks. Object Detail Tables further specify the information associated with each request, information encapsulated in each object, and the request propagation process. Each object represents an organizational entity whose functions are necessary for the entire system. Some of the objects may be automated; some activities can be performed by humans. Which objects should be automated or supported to what degree is design consideration and subject to many practical constraints. Yet the essential activities identified in OOEM must be performed, by human or by machine, in order to keep the enterprise running. In an environment in which technology is changing rapidly, it becomes even more important to extract and model the essential business functions and hide their implementation details.

In summary, while Coad and Yourdon's method is attempting to devise a possible information system solution, OOEM is set out to capture the essence of the business.

6.1.7. What to Include: Necessary vs. Possibly Useful

Another important difference between OOEM and Coad and Yourdon method is the way objects and services are identified. Coad and Yourdon method requires analysts to list all possibly useful objects by applying 37 different strategies for selecting objects [Coad, et. al. 1995 p385 - 394]. For example, one strategy for selecting objects is to "Select and Reuse an Analogous Class" which asks the analyst to:

- *Look for a class that might apply.*
- *Consider synonyms,*
- *Consider a more general name, using 'is a kind of'*
- *Consider metaphors (corresponding objects) within analogous systems (a system that has a analogous purpose)*

[Coad, et. al., 1995, p 169]. Then analysts can apply another set of 49 strategies to establish responsibilities (services and attributes for those objects). These strategies can identify services that are potentially useful. For example, for each object, the model suggests to consider *HowManyOverInterval* and *HowMuchOverInterval* services. In the Order Entry case, customer object is given these services as a result of applying this strategy.

OOEM uses a *request propagation* method to identify objects and establish their necessary services. According to Rule # 2, objects are included in the model **only if** they provide some services or submit requests. Objects identified by this method are always necessary for the enterprise to perform its tasks.

6.1.8. Summary

This subsection compares the Coad and Yourdon's method and the OOEM. Major differences between the two methods include the following:

- The fundamental difference arises from the definitions of the modelling constructs. While OOEM defines object as a model of thing which is active in the problem domain and hence requires each object to provide at least one service, Coad and Yourdon give a loose definition of object which include entities about which information must be kept by a information system. This difference results in modelling constructs that have different semantics.
- OOEM puts greater emphasis on the dynamic interactions among objects that participate in related business process. The Coad and Yourdon's method stresses on associations among objects.
- OOEM sets its domain of modelling to the business enterprise while many of the objects Coad and Yourdon's model do not model business activities in the organization.
- OOEM includes objects and services in the model only if they are necessary for the understanding of the business activities to be modelled. Coad and Yourdon suggest analysts to apply heuristic strategies and patterns that result in potentially useful objects and services being included in the model.

6.2. *Comparison with Jacobson's Method*

Jacobson's method - use case-driven approach - is an approach to object-oriented analysis and design that centered on understanding the ways in which a system is actually used [Jacobson, et. al., 1992]. The method was originally developed for information systems analysis and design. Recently Jacobson applied his concepts to enterprise modelling [Jacobson, et. al., 1995]. This section summarizes the Jacobson's business modelling method and compares it with OOEM.

6.2.1. Summary of Jacobson's Modelling Concepts

Modelling constructs of Jacobson's method also include object-classes, attributes, services, and messages. In addition, Jacobson introduced some new concepts. In particular *Use Case* and *Actor*. He defines a *Use Case* of a business system as "...a sequence of transactions in a system whose task is to yield a result of measurable value to an individual actor of the business system". An *Actor* is defined as "...one or a set of roles that someone in the environment [of business system] can play in relation to the business." [Jacobson, et. al., 1995. p 103].

In Jacobson method, objects are classified into three types: *control objects, interface objects,* and, *entity objects.* A *control object* "...represents a set of tasks in a business. These tasks could be performed by one resource instance which typically is a specialist or a routine worker, not dealing directly with the customer". An *interface object* represents "a set of operations in the business each of which should be performed by one and the same resource. Its tasks involve communicating with the environment of the business". An *entity object* represents "occurrences such as products, deliverables, documents, and other things that are handled in the business." [Jacobson et. al. p340]

The steps suggested by this method to model an existing business can be summarized as the following:

1. build a use case model
   - find actors. An actor can be a customer, business partner, supplier, government, or subsidiary.
   - find use case. Begin with an actor (usually the customer) and identify a complete course of events that result in measurable value for the actor.
   - prioritize use cases
   - describe use cases
   - select matrices and review
2. Build an object model
   - find subsystems
   - describe use case in relation to subsystems
   - find objects

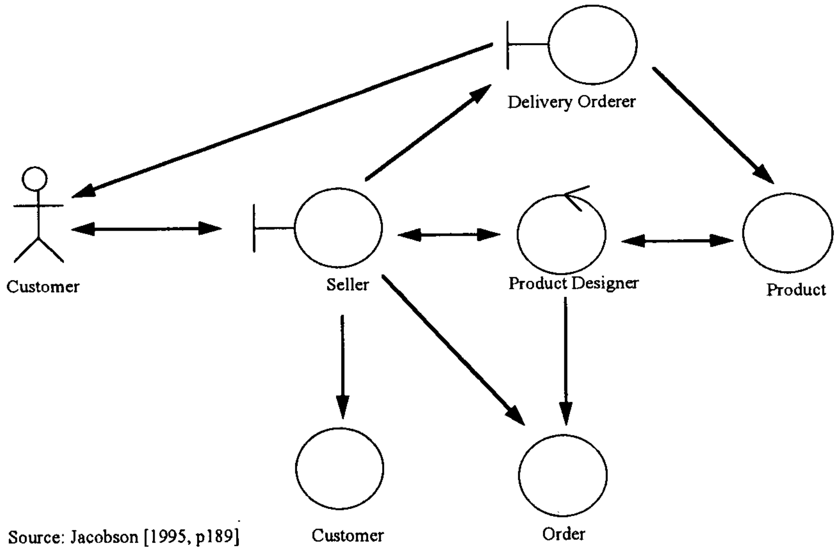
6.2.2. Case study: Customizes sales process

This case of customized sales process, also called customized sales, is adapted from [Jacobson et. al. 1995, p 186]. The sales process can be described as following.

> *To order a customized product, a customer contacts a sales person and gives him/her product specifications. The salesperson forwards the product specifications to product designer who designs and manufactures the product. When the product is ready, product designer informs the sales person who would contact delivery orderer to send the product to the customer.*

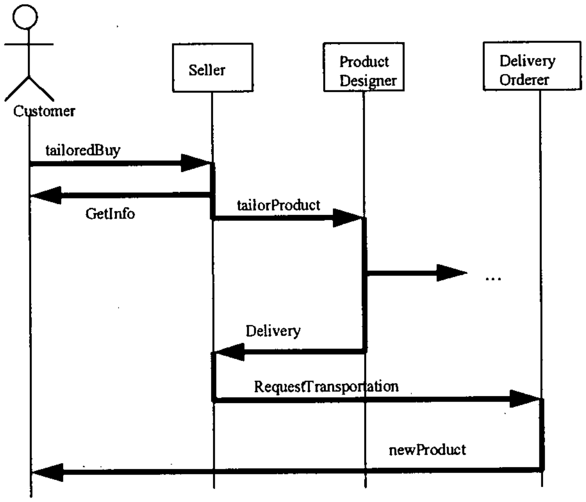
The object model using Jacobson's method is shown Figure 6-3. An interaction diagram which is used by Jacobson to depict interactions among objects is shown in Figure 6-3. In this figure, the 🧍 denotes an actor, the ├○ an interface object, the ⥀ a control object, and the ○ an entity object. The Figure 6-4 depicts the flow of events in the system. Each arrow represents a request. A thick vertical line represent the period during which the request is being processed by the corresponding object. OOEM Object Request Model and Object Detail Model are presented in and respectively.

Figure 6-3: Jacobson's Object Model for Customized Selling



Source: Jacobson [1995, p189]

Figure 6-4: Jacobson's Interaction Diagram for Customized Selling



Source: Jacobson [1995, p190]

Figure 6-5: OOEM Object Communication Model for Customized Selling Case

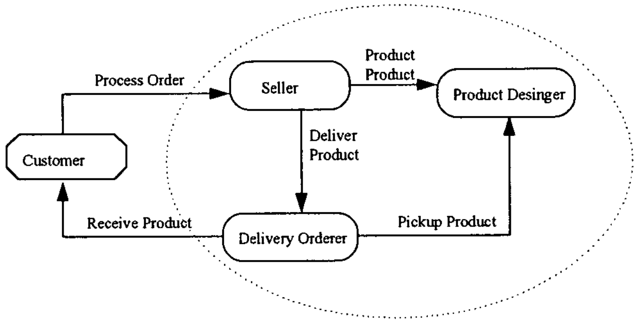


Table 6-4: OOEM Object Detail Models for Seller

	Seller					
	Services	Interface Attributes	Internal Attributes		Request generated	
Customer	Process Order	Product spec order info customer info	customer info order info	M, M	Design and produce product	Product designer
		confirmation, expected shipment dates	order status Product specs.	M M	Deliver product	Delivery Orderer

6.2.3. Mapping Analysis Results

This subsection discusses the mapping from Jacobson's objects to the objects in the OOEM model of the Customized Selling case. The mapping of objects is shown in Table 6-5.

Table 6-5: Mapping of Objects: the Customized Selling Case

Objects and Actors in Jacobson's Model	**Objects in OOEM model**
Customer (Actor)	Customer (external object)
Seller Customer (entity object) Order	Seller • Information about customer is encapsulated as internal attributes of the seller. • Order information is also encapsulated as internal attributes of the seller
Product Designer Product	Product Designer • Product design details are encapsulated in the product designer who creates such information.
Delivery Orderer	Delivery Orderer

Similarities between the two models are quite evident. In this case, all interface objects and control objects can be mapped to OOEM internal objects. Actors are mapped to client in OOEM. Although request names and associated information are not presented in the Jacobson's model, message connections among actors, interface objects, and control objects resembles those in OOEM.

However, differences remain. The rest of this section discusses these differences by concentrating on specific aspects of the models.

6.2.4. Comparison of Modelling Constructs

We can map Jacobson's constructs onto OOEM's constructs in the following manner.

Table 6-6: Mapping Modelling Constructs: Jacobson and OOEM

Jacobson's Modelling Constructs	**OOEM Modelling Constructs**
Actor	External Object
Use-Case	External Request and its propagation
Interface Object Control Object	Internal Object *In Jacobson, the distinction of* interface objects *and* control object *depends on the sytem scope.*
Entity Object	*Not Object* *In OOEM information about the Entity objects is encapsulated in* Internal Attributes *of objects. The concept of* Entity Object *which serves as data store violates the OOEM Rule # 5.*
Message Connection • among actor, interface object and control object • with an entity object	Request *In OOEM, access of internal attributes not modelled explicitly*

6.2.5. Nature of Models: Use case and external request

The OOEM concept of *external request* is similar to the Jacobson's concept of *use case*. Both require the analyst to identify clients (actors) of the enterprise and the requests (use case) these clients may submit to the organization. Both describe the sequence of events that the external request triggers in the system. In other words, use case in Jacobson's method and external requests in OOEM are used to model essentially the same thing.

6.2.6. What to include: On Object Identification

When Jacobson classifies objects into three types of objects, he does not offer the fundamental reasons of doing so. Both control objects and interface objects provide some services. The only difference between them is that the latter interfaces with external objects. When the analyst changes the scope of the analysis, a control object may become an interface object or vise versa. The entity objects provide a mechanism to store information which can be accessed by other objects. When two or more objects access and modify an entity object, the encapsulation principle of object-orientation is violated.

In addition Jacobson does not provide rules to identify objects. OOEM not only provides a set of modelling rules, but also well-defined algorithms which implement these rules.

In the customized selling case, the OOEM algorithm clearly excludes all entity objects in Jacobson's model from the model. An interesting observation on Figure 6-3 is that there are two models for customers, *customer* actor, and *customer* entity object. *Customer* actor denotes the class of people who interact with the business, whereas *customer* entity object refers to the management of information about the customers. By management of information we mean that the retention, access, and manipulation of information. In this case, it refers to adding, updating, retrieving, and deleting customer contact information including name, address, phone number, etc. In OOEM, objects represent things themselves, not merely information about things. A customer contact information represents the fact that *seller* ***knows*** about the customer. It is the *seller* who uses this information to perform its services. Therefore customer information and the management of it should not be represented as separate objects in the enterprise model. It should be encapsulated in the *seller* object in order to reflect the essence of what is happening in the organization.

6.2.7. On Model Semantics

The semantics of Jacobson's Object Diagram are not necessarily clear and consistent. For example, the arrow from the actor *customer* to object *seller* seems to indicate a request, whereas the arrow from *designer* to *order* seems to represent information access. This overloading of diagrammatic notation is further confounded by lack of labels on these arrows. On the other hand, the semantics of an OOEM diagram is more formalized. Arrows denoting requests have well-defined semantics. The semantics of requests, i.e., the associated information to be sent to the receiver and possible response, is specified in the Object Detail Tables. Services and internal attributes of objects are tabulated. The sequence of modelling activities is also formally specified by the request propagation algorithm.

6.2.8. Summary

The emphasis on *Actor* and *Use-case* is similar to the Request Propagation approach which starts analysis with the External Objects and External requests. In the case study, all interface objects and control objects match internal objects in OOEM.

Jacobson's classification of objects into three types seems rather implementation related. The entity objects which provide mechanism to store information accessed by other objects do not model real world active business components. When entity objects are accessed and modified by multiple objects, the encapsulation principle is violated. In addition, no systematic modelling strategy is offered in Jacobson's method to identify objects, services, and attributes. The semantics of connections between control objects and interface objects are inconsistent with those between control object and entity objects.

6.3. Comparison with Rumbaugh's OMT

The important modelling constructs employed by Rumbaugh include *Object, attributes, operations* (services), *link and associations, aggregation, and generalization and specialization.* A *link* is a "physical or conceptual connections between object instances" and *associations* are "group of links with common structure and common semantics"[Rumbaugh et. al. 1991, p 27]. An *aggregation* is "the 'part-whole' or 'a-part-of' relationship in which objects representing the *components* of something are associated with an object representing the entire *assembly*[Rumbaugh et. al. 1991, p36].

The Rumbaugh's Object Modelling Technique (OMT) method includes 4 distinct stages: analysis, system design, object design, and implementation. Although OMT is primarily concerned with information system development, the analysis stage has a significant overlap with the domain of OOEM. This subsection will concentrate on comparing OOEM with this stage of OMT. The description of OMT is based on [Rumbaugh, et. al. 1991].

6.3.1. Rumbaugh's OMT Summary

The OMT uses three kinds of models to describe a system: the *Object Model,* the *Dynamic Model,* and the *Functional Model.* The *Object Model,* represented in an Object Diagram, describes the objects in the system and their relationship. The *Dynamic model,* represented in a State Diagram, describes the interactions among objects in the system. *Functional model,* represented in a Data Flow Diagram, describes the data transformations of the system.

The goal of analysis is to develop a model of what the system will do. The analysis process suggested by OMT consists of the following activities:

1. Write or obtain an initial description of the problem domain

2. Build an Object Model:

   - identify object classes. Objects include "physical entities, such as houses, employees, and machines, as well as concepts, such as trajectories, seating assignments, and payment schedules".
   - begin a data dictionary containing descriptions of classes, attributes, and associations
   - add associations between classes
   - add attributes and links(association between objects)
   - organize and simplify object classes using inheritance.
   - test access paths using scenarios and iterate the above steps
   - group classes into modules, based on close coupling and related functions

3. Develop a Dynamic Model

   - prepare scenarios of typical interaction sequences
   - identify events between objects and prepare an event trace for each scenario
   - prepare an Event Flow Diagram for the system
   - develop a state diagram for each class that has important dynamic behaviour

4. Construct a Function Model

   - identify input and output values
   - use data flow diagrams as needed to show functional dependencies
   - describe what each function does
   - identify constraints
   - specify optimization criteria

5. Verify, iterate and refine the three models.

6.3.2. Case Study: Automated Teller Machine (ATM)

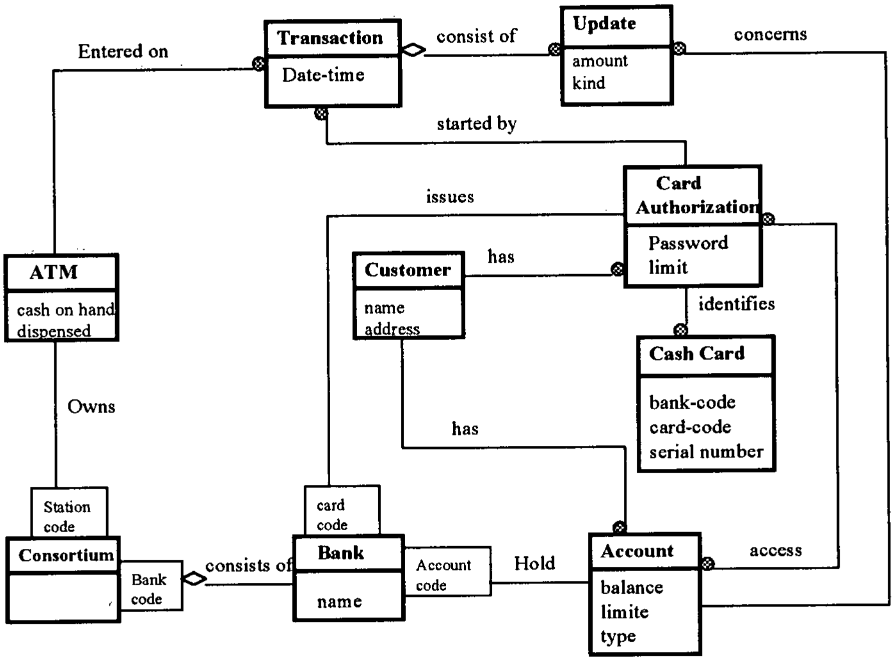
In order to provide a concrete comparison, this section includes a case example and the solution by the two approaches. This case is adopted from [Rumbaugh et. al. 1991, p.151].

> *ATMs allow customers to access their account directly and 24 hours a day. The ATM network is shared by a consortium of banks, each of which maintains its own accounts and processes transactions against them. ATMs communicate with a central computer in the consortium that clears transactions with the appropriate banks. A customer of one of the member bank of the consortium can log-on one of the ATM's and issue a series of transactions. The ATM communicates with the central computer to carry out the transaction, dispenses cash, and prints receipts.*

The Object model, an Event Flow Diagram, a State Diagram and a DFD of Rumbaugh's model for this case are presented in Figure 6-6, Figure 6-7, Figure 6-8, and Figure 6-9, respectively. In the object model, a box represents an *object-class.* Lines connecting boxes are *associations* between objects-classes. A small diamond box on a line stands for whole-part relationships. The box connected with the diamond is the aggregate. In the State Diagram, an oval box stands for a state. The arrows stand for control flows. In the DFD, two horizontal bars represent a data store. An oval denotes a process. An arrow stands for data flow. A hollow arrow stands for data flow that results in a data store. A dash arrow stands for a control flow.

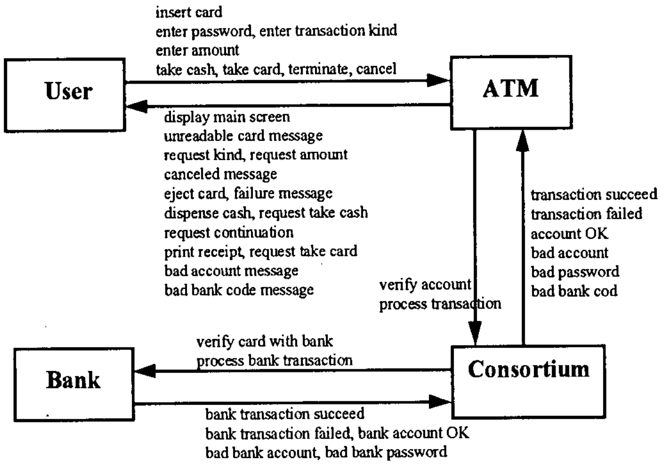
An OOEM Object communication diagram and Object detail table are show in Figure 6-10 and Table 6-6 respectively. The next subsection discusses the differences between the two methods.

Figure 6-6: OMT Object Model for ATM case



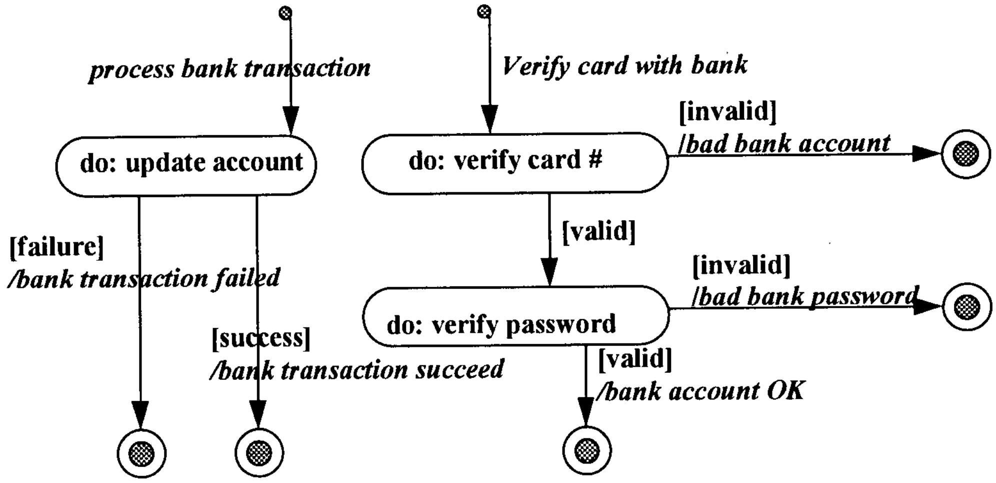
Source: Adaped from Rumbaugh et. al. [1991, p168]

Figure 6-7: OMT Event Flow Diagram for ATM Case



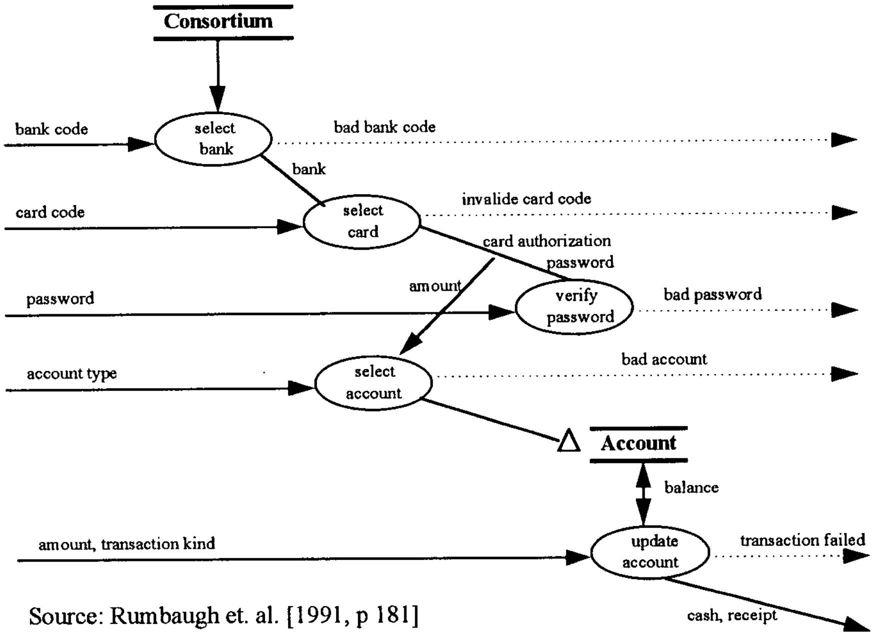
Source: Rumbaugh et. al. [1991, p175]

Figure 6-8: OMT State Diagram for Bank



Source: Rumbaugh et. al. 1991 p177

Figure 6-9: OMT Data Flow Diagram for ATM perform transaction



Source: Rumbaugh et. al. [1991, p 181]

Figure 6-10: OOEM Object Request Model for ATM Case

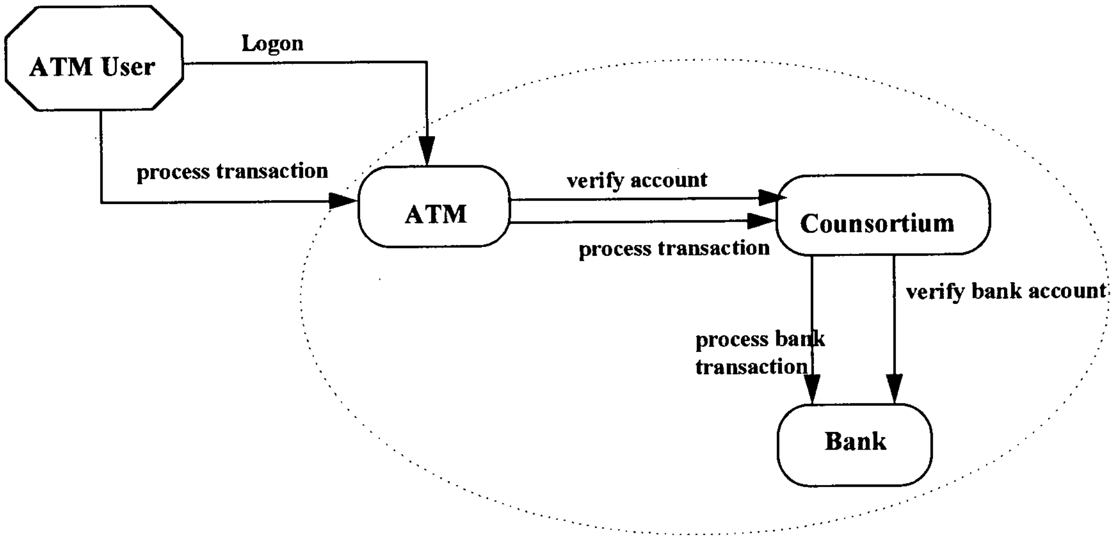


Table 6-7: ATM Case: OOEM Object Detail Table for Consortium

	Consortium					
	Services	Interface Attributes	Internal Attributes		Request generated	
ATM →	Verify account access	card-code, bank code account #, password	bank code connection	U M	Verify account access	→ Bank
		access authorized, or access failure				
ATM →	process transaction	transaction type, amount	bank connection	U	Process transaction	→ Bank
		transaction OK or transaction failed				

6.3.3. Mapping Analysis Results

To compare the two methods, let us first map the objects in OMT onto those in OOEM (see Table 6-8). The OOEM is a simpler model since it only uses 4 objects to represent the problem domain while OMT uses 9 objects. The OOEM model captures all information represented by the OMT. In OOEM, information about account, customer, cash card, card authorization, transaction, and update are included as internal attributes of appropriate objects. They are not modelled as objects because they do not provide meaningful services.

Table 6-8: Mapping Objects in OMT onto those in OOEM: ATM Case

Objects in OMT Model	Objects in OOEM
ATM	ATM
Consortium	Consortium
Bank	Bank Bank *object in OOEM has more attributes than the one in OMT.*
Account	*Account information is encapsulated as internal attributes of* Bank *object*
Customer	*Name and address of a customer are internal attributes of a* Bank *object*
Cash Card	*Information on the Cash Card is passed as interface attributes from* Customer *to* ATM *object in the OOEM model*
Card Authorization	*Password and limit are interface attributes among* ATM, Consortium *and* Bank.
Transaction	*Access of Account is recorded as internal attributes of the* Bank *object.*
Update	*The association between a transaction and an account is recorded in the* Bank *object.*

The similarity between OOEM and OMT event flow diagram is quite apparent; both contain the same objects, which are connected in the similar pattern. One important difference is in the semantics of the arrows. In OOEM, the meaning of an arrow is defined as a request. Interface attributes associated with each request are specified in the object detail tables. In OMT, an arrow may mean a request, e.g., *verify account*, or a data flow, e.g., *bad account message.*

The fact that only 4 out of 9 objects in the object diagram are included in the event flow diagram indicates that the objects in the object diagram are not of the same importance. This issue is discussed in detail in the later subsection on "what to include".

6.3.4. Comparing Modelling Constructs

Based on the comparison of the ATM case analysis results, we can compare OMT's modelling constructs with those of OOEM.

Table 6-9: Mapping of OMT constructs onto OOEM constructs

OMT Modelling Constructs	OOEM Modelling Constructs
Object	Object *Sometimes* Objects *in OMT are modelled as* attributes *since they do not provide meaningful services in the problem domain.*
Attribute	Attribute *Sometimes* attributes *in OMT are not included in the OOEM.*
Operation	Service
Generalization	Generalization
Aggregation	Whole-part

The two methods generally agree on the concepts of *operation/service*, *generalization*, and *aggregation*. Again important differences appears in the concepts of *Object* and *Attributes*.

The OOEM modelling rule of object identification dictates that an object must provide service(s) meaningful in the analysis domain. This criteria of object identification excludes many entities in the OMT object model. These entities represent things about which information is processed, but which do not provide services themselves. For example, a transaction does not provide meaningful service in the analysis domain. Therefore, information about transaction is modelled as attribute of some object, in this case, the *Bank*.

There are also fine distinctions between OMT's concept of attributes and that of OOEM. In OMT, attribute is simply defined as "data value held by the objects in a class"[Rumbaugh et. al. 1991 p23]. For instance, the name of a bank is modelled as an attribute of the bank. In OOEM, attributes of an object are the information remembered and possessed by the object. This information is necessary for the object to perform its services. For example, the bank must remember the password for each account. The aggregate of the passwords and their associated accounts are attributes of the *Bank* object. Please notice that based on this criteria, the name of the bank should not be included in the model unless a request uses it.

6.3.5. Nature of the Model: Ternary versus Unary Approach

Graham [1994] identifies three dimensions of information systems engineering: data, process, and dynamics. OMT uses object model, functional model, and dynamic model to describe these essential aspects of information systems respectively. Yet the linkage among the three models is not always apparent. For example considering the object-oriented principle of encapsulation which requires that all process in a model be encapsulated in some objects. In the ATM case, from reading the diagrams presented in Figure 6-6 and Figure 6-9: OMT, readers cannot easily decide which object owns the process "select bank". To obtain an overall picture, readers have to mentally connect one component in one model with some other components in another. Model integration may be particularly difficult when OMT is used to model large complex organization where the sheer volume of information may overload readers quickly.

OOEM employs a single diagram to model the components and the interactions among them. To keep this model simple, all detailed information about an object is presented in object detailed tables. This unified approach enables the reader to comprehend the entire model.

However, OOEM may have difficulties to express certain temporal sequence of events. For example, the fact that user must first log-on the ATM and then issues transactions is not readily expressed in the OOEM. Service pre-condition and post-condition may be specified for services such as the "process transaction" in the detailed description of ATM object. In the latter case, additional column should be added to the object detail table.

ATM (modified)					
Services	Interface Attributes	Internal Attributes			Request Generated
			Pre-cond.		
Log-on	card-code, bank code account #, password Log-on OK or log-on failure	current card info, log-on status, Cash on hand		U M U	Verify account access
process transaction	transaction type, amount transaction OK or transaction failed	log-on status	**log-on OK**	U	process transaction

6.3.6. Domains of the Models: Information System vs. Enterprise Modelling

OMT is a system development method with the purpose of building working software systems that support part of business activities. OMT's analysis method is concerned with "devising a precise, concise, understandable, and correct model of the real-world" [Rumbaugh, 1991, p. 148] for the IS developer. To serve this goal, OMT includes a rich set of notations designed to model details of the system environment. To produce a robust software system, we need to know a great amount of details about the organizational environment that the software is designed to support. OMT's diagrams provide means to capture such detailed information from data, dynamic and functional aspects.

OOEM, on the other hand, is set out to model the essential business activities. It only specifies the fundamental components of the business and interactions among them. The detailed procedures of managing and processing the information is deliberately excluded from the model. This is in contrast to the OMT's attempts to model the processing details within an object. With higher level of abstraction, OOEM diagram should be more understandable to management and business analysts who are more concerned with business process rather than information processing details.

6.3.7. What to include: "First-Class" and "Second-Class" Objects

In theory, each object in the object model encapsulates both data and behaviour. A state diagram can be drown for each object to represent its behaviour. However, the state diagram is only relevant for a few active objects in the OMT Object Diagram. In the ATM case, these objects are *ATM*, *consortium*, and *bank*. Other objects, such as *cash card*, *account*, *updates* are rather "passive objects" for which state diagrams are irrelevant. Therefore, there are two types of objects in the OMT object diagram: "first-class" objects that encapsulate both data and behaviour and are called "Actors" in OMT, and "second-class" objects, which do not provide meaningful services in the analysis domain. The latter only serve as containers for some information, like entities in ER diagram.

According to the object identification rule, OOEM should only includes the "first-class" objects, never the "second-class" objects. This observation is supported by the ATM case. Actors in the OMT model is included in the OOEM model. Information about some of the "second-class" objects in OMT, e.g., account, is encapsulated in OOEM objects, e.g., bank.

6.3.8. Summary

- OMT uses three separate models to capture process, data, and dynamic aspects of an information system. Model integration may be problematic for large, complex systems. OOEM employs a unified object model which is supplemented by object detail tables.

- The OMT object model includes not only objects that provide services, i.e., "first-class objects", but also entities about which information is processed: "second-class objects". In contrast, OOEM only allow objects that provide service(s) to be included in the model.
- As an integral part of information system development method, OMT analysis method intents to model information processing details. These detailed information form a base for system design implementation. OOEM, on the other hand, is used to understand the essential business activities. In order to manage complexity, it deliberately excludes low level details from the model.

6.4. Chapter Summary

This chapter compares OOEM with three other object-oriented methods: the Coad and Yourdon method, the Jacobson's method, and the Rumbaugh's OMT. The major finding are the following.

- Although object-oriented concepts such as inheritance, Generalization-Specialization, and whole-part relationship are generally agreed upon, differences remain on the fundamental definition of an object. All the methods compared here include not only objects appeared in OOEM models, but also some other entities. Information about these entities must be retained and processed in the information system. But these entities do not provide service(s) meaningful in the business domain. The object identification rule in OOEM clearly excludes these entities from the object model. Instead, information about these entities may be modelled as attributes of objects.
- All the methods compared here are influenced to various degree by the goal of developing an information system. OOEM, which derive its modelling concepts from ontology, aims at providing means to describe and understand business enterprise. In order to manage

complexity in the real world business environment, it deliberately excludes certain low level details.

The major contribution of OOEM is its systematic way of creating a model. Other methods uses heuristic strategies and steps but none provides a formalized method for finding objects, their attributes and services. The Request Propagation Algorithm along with the Object Detail Templates provide a unequivocal process and tool to construct a business model. This algorithm is derived from a set of well-formulated ontological concepts and principles.

7. Conclusions and Future Research

7.1. Thesis Summary

This thesis presented a new method for business analysis named Object-Oriented Enterprise Modelling (OOEM). It first offered a definition to the term "enterprise modelling" based on research in cognitive psychology and systems analysis. The purpose of enterprise modelling is to assist analysts to understand, capture and communicate both structural and dynamic aspects of business activities in an organization.

Based on the research by Wand and Woo in the enterprise modelling area, this thesis derived a formal model of object and object-oriented system from ontological principles. It proposed a definition of object which emphasis both structural and behaviour aspects of an object. The essential modelling constructs in OOEM are *object, attributes, services,* and *requests.* The thesis also discussed modelling rules developed originally by Wand and Woo. These rules ensure the OOEM adheres to the ontological principles. They define the fundamental characteristics of the model in areas of the scope, object identification, attribute and service inclusion, attribute ownership, aggregation, and classification.

The OOEM method consists of techniques and activities that can be employed to build an enterprise model. The Object Communication Model is used to capture communications among objects both inside and outside the enterprise scope. Detailed information about each object is presented in a separate Object Details Template. Services, interface attributes, internal attributes, request propagation, and their links for each object are captured in the Object Detail Table. The separation of the Object Communication Model and the Object Detail Tables is particularly useful for managing complexity when OOEM is used to model large organizations.

The process of creating an OOEM model consists of a sequence of activities formally defined by the Request Propagation Algorithm. The algorithm has two parts: the main algorithm which iterates over all external requests of the enterprise; and the recursive algorithm which traces all requests propagated from a single request. For each important step in the algorithm, a set of questions is included in the method. These questions can direct analysts to focus their attention on specific aspects of the business. The answers to these questions are captured by the Object Request Model and the Object Detailed Template. The algorithm ensures the OOEM modelling rules are followed.

To illustrate how the method can be applied to a real world situation, the analysis process for a reasonably realistic case is presented. It is demonstrated that by following the Request Propagation Algorithm and asking the proposed questions, an analyst can systematically understand all business objects, identify services and attributes for them, and model the message connections between them.

Next, the thesis compares OOEM with Coad and Yourdon OOA, Jacobson's Objectory and Rumbaugh's OMT. All of the compared methods are influenced, to various degree, by the goal of developing an information system. OOEM which derive its modelling concepts from ontology clearly defines its modelling to be the real world business enterprise itself, not the information system. In addition, OOEM offers a set of well-defined modelling rules, a structured representation of object and a systematic method to create the model for an enterprise. All the methods compared only provides heuristic strategies to find objects, attributes and services.

7.2. *Contributions*

This thesis defines all OOEM modelling constructs based on ontology, a formal model of the real world. This gives OOEM constructs specific and well-defined semantics when they are used to model real world business enterprises. The OOEM object and system representations contain, in our opinion, all necessary information for an analyst to understand the organizational activities. A significant contribution of the thesis is the Request Propagation Algorithm which operationalize the OOEM rules by providing a systematic approach for modelling an enterprise. The strengths of this specific technique of model construction is that it is derived from its formal ontological foundation rather than based on traditional IS design heuristic or programming concepts. This prevents design and implementation features, which are likely to be detrimental to systems analysis [Embley, et. al. 1995], from "creeping into" the analysis. In addition, the thesis showed the mappings between the constructs used in OOEM and those used in three other OOA methods. Such mappings form the base for future research on how to apply OOEM modelling rules to other OOA methods.

It should be noted that the work presented in this thesis is based on previous research efforts in this direction conducted in Faculty of Commerce at UBC.

7.3. Limitations and Future Research

The foundation of the OOEM method was established by previous research. This thesis makes several additional contributions related to the practical application of it. However, OOEM is a rather new method. The number of cases where this method has been applied and tested is still limited. The application of Rule #6 and Rule #7 should be made more systematic. The assumptions, especially the one about the analyst's ability to understand the meanings of different terms used by domain experts, may not hold in all cases. Rigorous test of the method in analyzing a greater number of real world business enterprises will help to verify and enhance the method.

Future research may focus on the following areas:

- Systematic approach for applying Rule #6 and #7 to organize objects identified by the Request Propagation Algorithm can be developed.
- CASE tools that integrate the Object Template and Object Communication Diagram and automate certain parts of the Request Propagation Algorithm will assist wider usage of the method.
- Empirical research can be conducted to test the ease-of-use and model quality of OOEM in comparison with other methods.
- Research can address the question of how to adopt the OOEM modelling constructs, rules, and algorithm to the design of information systems.

Bibliography

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

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

Chen, P. P. (1976) The Entity-Relationship Model - Toward a Unified View of Data. *ACM Transactions of Database Systems,* 1, 9-36.

Coad, P and Yourdon, E. (1995) *Object Models - Strategies, Patterns, and Applications.* Englewood Cliffs, NJ: Yourdon Press/Prentice Hall.

Coad, P. and Yourdon, E. (1991) *Object-Oriented Analysis,* 2nd edn. Englewood Cliffs, NJ: Yourdon Press/Prentice Hall.

Cox B.J. (1986). *Object-Oriented Programming - An Evolutionary Approach.* Reading, MA: Addison-Wesley.

Cox B.J. and Novobilski A. (1991). *Object-Oriented Programming - An Evolutionary Approach* 2nd edn. Reading, MA: Addison-Wesley.

Craik, K. *The Nature of Explanation.* Cambridge University Express, Cambridge, UK 1943.

Dahl O.J., Myrhaug B. and Nygaard K. (1968). *SIMULA 67 Common Base Language.* Norwegian Computing Centre, Oslo. 1968.

Dedene, F. A practical approach to consistent object oriented business modelling. In *Proceedings of 5th international conference: Putting into Practice Methods and Tools from Information System Design.* Nantes, France. 1992.

Embley, D. W., Jackson, R.B., and Woodfield, S. N. (1995). *OO Systems Analysis: Is It or Isn't It?.* IEEE Software, July 1995, Vol 12, #4.

Embley, D. W., Kurtz B.D. and Woodfield S.N. (1992). *Object-Oriented Systems Analysis: A Model Driven Approach.* Englewood Cliffs, NJ: Yourdon Press.

Fichman, R. G and Kemerer, C.(1992) *Object-Oriented Analysis and Design Methodologies: Comparison and Critique* IEEE Computer.

Glykas, M.; Wilhelmij, P.; Holden, T. (1993) *Object oriented in enterprise modelling and information system design,* IEE Colloquium on 'Object Oriented Development' No.007 p. 8/1-19 London, UK

Gorman, Michael. *Enterprise Database in a Client/Server Environment.* John Willey & Son, Inc. 1994.

Graham, Ian. (1994) *Object-Oriented Methods* Addison-Wesley Publishers Ltd.

Hutt, A. T. F. edt. (1994) *Object Analysis and Desing - Description of Methods.* John Wiley & Sons, Inc.,

Jacobson, I. (1992) *Object-Oriented Software Engineering - a Use Case Driven Approach* ACM Press, Addison-Wesley.

Jacobson, I., Ericsson, M., Jacobson, A. (1995) *The Object Advantage - Business Process Re-engineering with Object Technology* ACM Press. Addison-Wesley Publishing Company.

Johnson-Laird, P.N. (1989). *Mental Models. In Foundations of Cognitive Science.* Cambridge, MA: The MIT Press, 469-499

Kay, A., and Goldberg A. (1977) Personal dynamic Media. *IEEE Computer* 1977 - originally a 1976 Xerox technical report.

Meyer B. (1988). *Object-Oriented Software Construction.* Englewood Cliffs, NJ: Prentice-Hall.

Parsons, J. and Wand, Y., (1993) *Object-Oriented Systems Analysis: A Representation View.* Working Paper 93-MIS-001, Faculty of Commerce and Business Administration, The University of British Columbia.

Rumbaugh J., Blaha M., Premerlini W. et al. (1991). *Object-Oriented Modelling and Design.* Englewood Cliffs, NJ: Prentice-Hall

Stroustrup B. (1988). What is object-oriented programming? *IEEE Software,* May 1988, 10-20.

Takagaki, K., (1990) *A Formalism for Object-Based Information Systems Development* Unpublished Ph.D. Dissertation, Faculty of Commerce, The University of British Columbia.

Takagaki, K., Wand, Y. (1991) An Object-Oriented Information Systems Model Based on Ontology, *Object Oriented Approach in Information Sytems* Assche, F. V., Moulin, B. and Rolland, C. eds., 275-296.

Thierauf, R. (1986) *Systems Analysis and Design.* Englewood Cliffs, N.J. Prentice-Hall.

Wand, Y., (1989) A Proposal for an Formal Model of Objects. *Object-Oriented Concepts, Languages, Applications, and Databases* Kim, W. and F.H. Lochovsky. eds., 537-559, ACM Press, Addison-Wesley publishing Co., New York, New York, Wand, Y., Weber, R., (1990a) An Ontological Model of an Information System, In *IEEE Transactions on Software Engineering*. Vol. 16, No.11 p1282-1292.

Wand, Y., Weber, R., (1990b) *Mario Bunge's Ontology as a Formal Foundation for Information Systems Concepts.* Studies on Mario Bunge's Treatise Rodopi B.V., Amsterdam - Atlanta, GA, (p 123 -149)

Wand, Y., Woo, C. (1992) A Formal Model for Analyzing Organizational Computing Concepts. Working Paper 92-MIS-010, Vancouver, BC, Canada.

Wand, Y., Woo, C. (1993) *Object-Oriented Analysis - Is It Really That Simple?* Proceedings of the Workshop on Information Technologies and Systems, Orlando, Florida (186-195)

Appendix I Bunge's Ontological Constructs

The following sections are mainly an abstraction of Bunge's work [Bunge 1977, 1999]. Most of the wordings are direct quote from these books. The section regarding to the dynamic model of system is from Wand and Weber[1990a].

Static model of an substantial individual

- **Thing** A thing is defined as an entity or substantial individual endowed with all its properties. The world is made of things that have properties.

  Bunge distinguishes thing and constructs. Constructs are creations of the human mind. There are four basic kinds of constructs: concepts, propositions, contexts, and theories. Constructs do not have all the properties of things. For example, sets add and intersect but do not move around, have no energy and no causal efficacy, etc. Constructs, even those representing things or substantial properties, have a conceptual structure not a material one. In particular, predicates and propositions have semantic properties, such as meaning, which is non-physical property.

- **Properties, Attributes and Functional Schema** Properties of substantial individuals are called substantial properties. Properties of things can be intrinsic or mutual to several things, e.g. if a person is employed by a company, employment is a property of both the person and the company. A property is modelled via an attribute function that maps the thing into a set of values. Attributes are characteristics assigned to thing by human; therefore they reflect certain view point of the observer. An attribute can be represented as a function from a set of things and a set of observation points into a set of values. This is the base for defining a model of a thing as a functional schema: A functional schema is a set of attribute functions defined over a certain domain, usually time. Similar things can be modelled using the same functional schema.

- **Composite things** Composite things are things composed of other things. More precisely, an individual is composite iff it is composed of individuals other than itself and the null individual. Composite thing has *hereditary* properties and *emergent* properties. A property of a composite thing that belongs to a component thing is called a hereditary property. Otherwise it is called emergent property. A composite thing must have emergent property. The notion of emergent property is an important assumptions in Bunge's ontology. To him, every concrete system has assembled from, or with the help of, things in the same or lower order genera but possesses properties not available in the components of the system. The hierarchy of system genera can be characterised as: physical, chemical, biological, social, and technical.

- **State and Conceivable State Space** Every thing is - at a given time associated with a given reference frame - in some state or other. The vector of values for all attribute functions of a thing is the state of the thing.

  The set of all states that the thing might ever assume is the conceivable state space of the thing.

- **State Law** A state law restricts the values of the properties of a thing to a subset that is deemed lawful because of natural laws or human laws. A law is also considered a property of the thing.

- **Class, Kind, and Natural Kind** A class is a set of things that possess a common property.

  A kind is a set of things that possess two or more common properties.

  A natural kind is a set of things that share the same laws.

  Things come in natural kinds, i.e. classes of things possessing ("obeying") the same laws.

  A natural kind constitutes a natural grouping because it rests on a set of laws, but it is not a real thing: it is a construct.

Dynamic model of an substantial individual

- **Event** An event is a change of state of a thing.

  In order to keep track of the changes undergo by things, we need *the principle of nominal invariance* which states that a thing, if named, shall keep its name throughout its history as long as the latter does not include changes in natural kind - changes which call for changes of name.

- **Event Space** The event space of a thing is the set of all possible events that can occur in the thing. Let S(x) be a state space for a thing x. Any pair of points in this set will represent unambiguously a conceivable event in x.

- **Transformation and Lawful Transformation** A transformation is a mapping from a domain comprising states to a co-domain comprising states.

  Lawful transformation defines which events on a thing are lawful.

- **History** The chronologically ordered states that a thing traverses are the history of the thing.

Static Model of System

- **Coupling** A thing acts on another thing if its existence affects the history of the other thing. The two things are said to be coupled or interact.
- **System** A set of things form a system iff for any bipartition of the set, coupling exists among things in the two sets.
- **System Composition** A decomposition of a system is a set of subsystems such that every component in the system is either one of the subsystems in the decomposition or is included in the composition of one of the subsystems.
- **System environment** Things that are not in the system but interact with things in the system are called the environment of the system.
- **System structure** The set of couplings that exist among things in the system and among things in the system and things in the environment of the system is called the structure of the system.
- **Subsystem** A subsystem is a system whose components and structure are subset of the components and structure of another system.
- **Level structure** A level structure defines a partial order over the systems in a decomposition to show which subsystem are components of other subsystem or the system itself.

Dynamic Modelling of System

- **Stable State and Unstable State** A stable state is a state in which a thing, subsystem, or system will remain unless forced to change by virtue of the action of a thing in the environment (an external event).

An unstable state is a state that will be changed into another state by virtue of the action of transformation in the system (an internal event).

- **External Event** An external event is an event that arises in a thing, subsystem or system by virtue of the action of some thing in the environment on the thing, subsystem or system. The before-state of an external event is always stable. The after-state may be stable or unstable.
- **Internal Event** An internal event is an event that arises in a thing, subsystem or system by virtue of lawful transformations in the thing, subsystem or system. The before-state of an internal event is always unstable. The after-state may be stable or unstable.
- **Well Defined Event** A well-defined event is an event in which the subsequent state can always be predicted given that the prior state is known.
- **Poorly Defined Event** A poorly defined event is an event in which the subsequent state can not be predicted given that the prior state is known.

Appendix II Complete OOEM for ABC Case

This appendix presents a complete OOEM for the ABC case. The completeness means that all external requests are traced. In other words, the main algorithm has iterated over all external requests in this case.

Figure II-1 shows the complete object communication diagram for ABC case. Object Detail Template for each object is presented in Table II-1.

Figure II-1: Object Communication Diagram of ABC Case

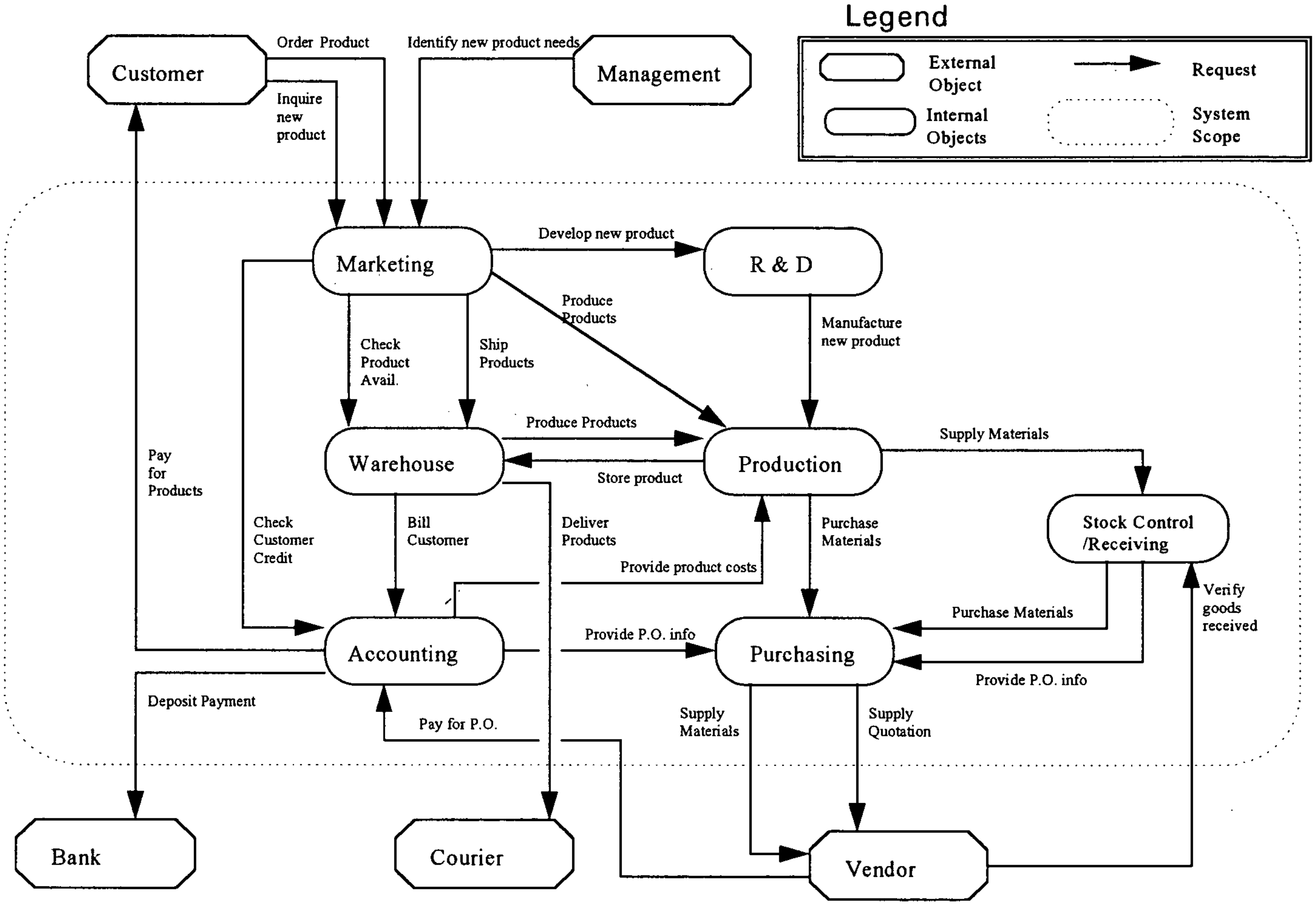


Table II-1: Object Detail Tables for ABC Case

Customer Object Type (External)		
Requests to system		
Order product		→ Marketing
Inquire new		→ Marketing
Services	Interface Attributes	
Accounting → Pay for Product	product name, quantity, delivery date,	
	payment	

Marketing					
Services	Interface Attributes	Internal Attributes		Request generated	Receiver
Order product	Customer Account #, Product spec, quantity,	Customer order file,	M	Check product availability	warehouse
				Ship products	warehouse
	confirmation, expected shipment dates	Customer's nearest warehouse	U	Produce products Check customer credit	production accounting
Inquire new product	product requirements	customer inquire history, marketing research results	U, M	develop new product	R & D
Identify new product needs	company strategy	Marketing research result	U, M	develop new products	R & D

R & D					
Services	Interface Attributes	Internal Attributes		Request generated	Receiver
Develop new product	product requirements	product design blueprint	U, M	manufacture new products	Production
	product developed				

Warehouse

Services	Interface Attributes	Internal Attributes		Request generated	Receiver
Check product availability	product #, quantity	Inventory file	U		
	product availability				
Ship products	Product #, quantity customer address	Inventory file	M	Deliver Products	Courier
	product shipped			Bill Customer	accounting
				Produce products	production
Store product	product #, quantity	inventory file	M		

Production

Services	Interface Attributes	Internal Attributes		Request generated	Receiver
Produce products	Product #, Quantity	production schedule, product blue prints	U	supply materials purchase materials	stock control purchasing
	product available	Finished goods file	M	Store product	warehouse
Manufacture new product	manufacture blueprint	production blueprint production schedule	M U,M	supply materials purchase materials	stock control purchasing
	product feasibility			Store product	warehouse
Provide product costs	product #,	production blue print production schedule	U U		
	date, bill of materials, labor costs				

Accounting

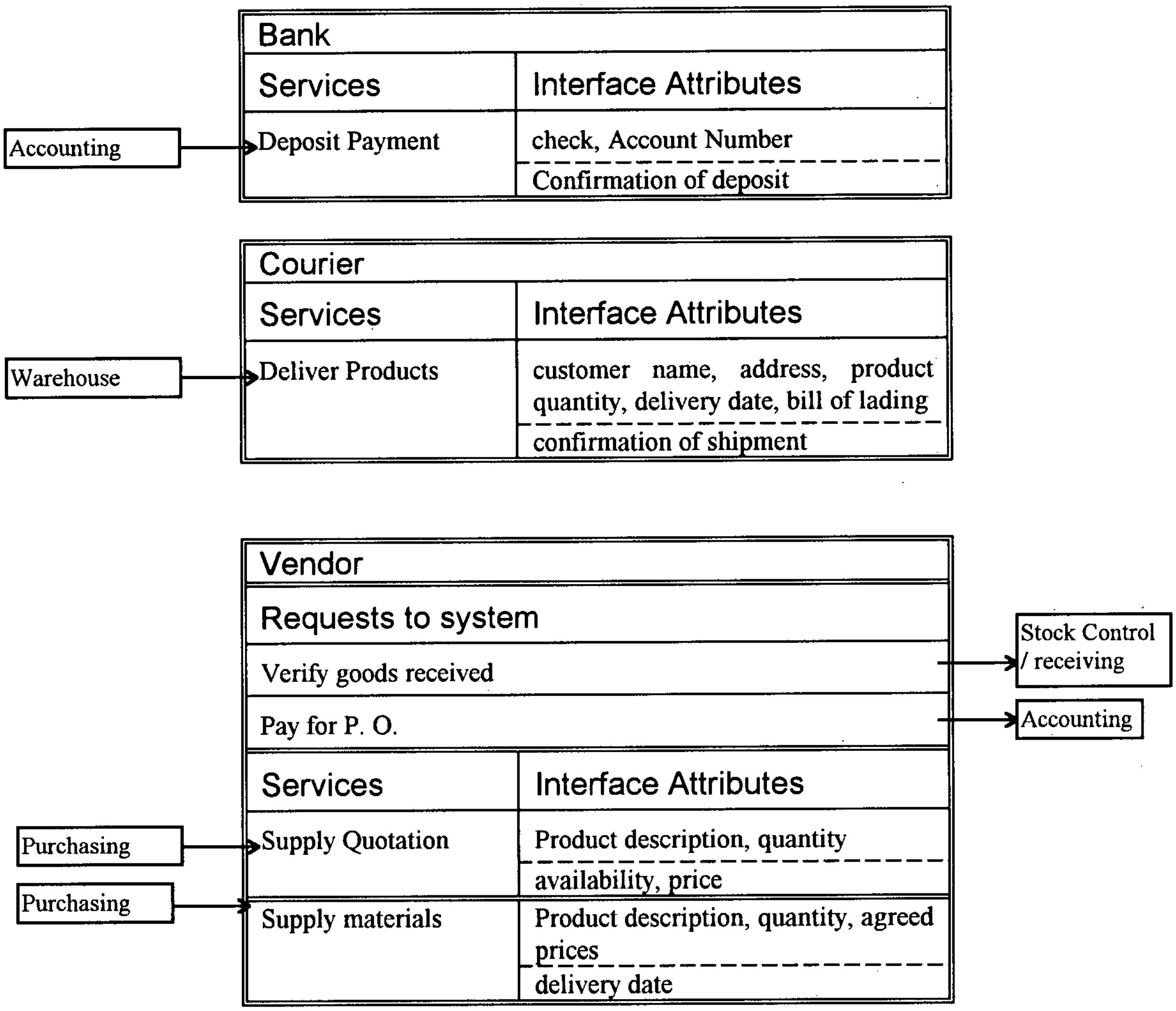
Services	Interface Attributes	Internal Attributes		Request generated	Receiver
Check Customer Credit	Customer name, # Product ordered, amount	Customer payment history	U		
	Credit availability				
Bill Customer	order #, product description, quantity, delivery date	Billing history Accounts Receivable Cash receipt journal	M M M	Pay for product	customer
	customer billed			Deposit Payment	Bank
Pay for P.O.	P.O.#, date, quality, total payable	material costs	M	provide P.O. info	Purchasing
	payment				

Purchasing

Services	Interface Attributes	Internal Attributes		Request generated	Receiver
Purchase Materials	materials specifications, quantity Material Ordered	Vendor quotation file Purchase Order file vendor contact file	U, M M U	Request for Quotation Supply Materials	Vendor Vendor
Provide P.O. info	P.O. # P.O. info	Purchase Order file	U		

Stock Control/Receiving

Services	Interface Attributes	Internal Attributes		Request generated	Receiver
Supply Materials	material specifications, quantity, Material availability	materials Inventory file	U, M	purchase materials	purchasing
Verity materials received	materials name, # quantity, P.O.# received OK	material inventory file receiving report	M M	provide P.O. info	purchasing



Bank

Services	Interface Attributes
Deposit Payment	check, Account Number Confirmation of deposit

Courier

Services	Interface Attributes
Deliver Products	customer name, address, product quantity, delivery date, bill of lading confirmation of shipment

Vendor

Requests to system
Verify goods received
Pay for P. O.

Services	Interface Attributes
Supply Quotation	Product description, quantity availability, price
Supply materials	Product description, quantity, agreed prices delivery date