OBJECT-ORIENTED MODELLING: FROM ANALYSIS TO LOGICAL DESIGN

by

DARRELL JUNG

B.Com., The University of British Columbia, 1994

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF

THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE IN BUSINESS ADMINISTRATION

in

THE FACULTY OF GRADUATE STUDIES

(Faculty of Commerce and Business Administration)

We accept this thesis as conforming

to the required standard

THE UNIVERSITY OF BRITISH COLUMBIA

August 1997

Faculty of Commerce and Business Administration
Department of Management Information Systems

The University of British Columbia
Vancouver, Canada

Date August 21, 1997

Abstract

Object-oriented concepts have recently been adapted for enterprise modelling. Because many of the existing object-oriented analysis and design methods were developed from software engineering roots, they lack fundamental rules and principles for modelling organizational activities. To remedy this situation, Object-Oriented Enterprise Modelling (OOEM), based on an ontological foundation, was developed to provide the rules and principles for enterprise analysis modelling. This thesis continues the work in the development of OOEM by providing extensions to the analysis modelling rules to encompass design concepts. The design principles provide a bridge for the transition from an enterprise model to an initial design of a logical information system model. The work proposes three types of concepts: decomposition, delegation and static objects, to provide a structured method for reasoning about design when one proceeds from an enterprise model.

Table of Contents

List Of Figures

List of Tables

1. Introduction

Object-oriented analysis and design are essentially modelling activities. Analysis examines existing systems, and design involves prescribing or planning for future systems (Olle et al. 1991, p. 2). Object-oriented modelling methods for these activities arose from software programming roots. Application of object-oriented methods to enterprise modelling is the latest focus that has joined the foray into object-oriented modelling. Zhao (1995) expanded on the object-oriented enterprise modelling (OOEM) rules, proposed by Wand and Woo (1993), based on ontological foundations. OOEM comprises a foundation of principles and rules that cover the analysis modelling of organizational activities. The next phase in modelling would encompass design activities. To continue from an OOEM model to a design model, one will require principles to guide the transition from analysis modelling to design modelling.

1.1 Background

The object-oriented paradigm originated from developments in software programming approaches. During the 1970's and 1980's the emphasis shifted from programming to design, and then finally to analysis issues. More recently, the *object-oriented* approach was further abstracted and adapted to information systems analysis. There is, however, a growing concentration in enterprise modelling. In enterprise modelling, the emphasis is on modelling the application domain before being concerned with specifying the requirements for implementation. The notion behind beginning with an enterprise model is that the global model (business overview) influences the enterprise model, which in turn influences the business plan, which impacts the analysis, design, and implementation of the individual software system (Yourdon 1992, p. 276). (NB: Although one may begin by constructing an enterprise model that is a reflection of a business plan, the enterprise model can be used as input for a new business plan.)

1.2 Problems with Current Object-Oriented Modelling Methods

There exist various proposed methods and techniques for object-oriented analysis and design; however, there is a fundamental weakness in current methods concerning how one transforms one model into another when one goes from *enterprise* analysis to design and finally to implementation. Proposed methods model the software domain. Software design activities remould analysis results to fit a specified implementation environment. Current methods do not provide basic principles to guide how one can think about or model the problem domain from the perspective of the enterprise or real world.

The roots for this problem lie in the software programming origins of the object-oriented approach. Analysis and design modelling evolved from object-oriented programming languages. Experience and constructs taken from the programming world were brought into the development of analysis and design methods. Because these methods were oriented and derived with the purpose of mapping out software development requirements, they provide no clear set of rules for using objects in modelling how an enterprise works. There is ambiguity as to how one would identify objects; determine which object properties to include; and evaluate the correctness of a model. It is common for formalized approaches to rely on non-object-oriented concepts (e.g. entities, functions, data flows) to provide constructs for conceptual modelling (Wand and Woo 1996, p. 4). In addition, Embley et al. (1995) state that "many so called analysis methods are really preliminary design methods.... [M]any of their features are more appropriate for design than analysis" (p. 19).

1.3 Objectives

This work proposes a method for transforming an object-oriented enterprise analysis model (OOEM) to a logical object-oriented design (OOD) model. The transformation will be made possible by applying a set of design rules. This work proposes a set of well grounded object-oriented design rules. These rules will have the effect of extending the original OOEM rules, which were used for object-oriented analysis modelling, to encompass object-oriented design modelling.

1.4 The Approach

This work takes the following general approach for proposing the rules for object-oriented design:

- suggest well-defined semantics for object decomposition, dynamics and statics
- introduce a notion of *delegation* in which certain activities of an object are delegated to another object (i.e. agent)
- examine and illustrate the design principles/rules with a case example

The rest of the work is presented as follows: chapter 2 discusses the object-oriented approach and some of the current formalized object-oriented analysis and design modelling approaches; chapter 3 provides a summary of object-oriented enterprise modelling (OOEM) and the underlying ontological foundation. OOEM serves as the basis from which a design model is to be built. Chapter 4 lays out the OOD modelling rules and principles. Chapter 5 presents a case study to illustrate the use of the OOD rules. Chapter 6 compares the proposed design rules to some of the current OOD methods. Chapter 7 outlines future research work and concludes the paper.

2. The Object-Oriented Approach: Analysis and Design

Object-oriented modelling (OOM) methods provide a specific language to document and communicate about information systems. These methods have traditionally been used in software development efforts. Systems analysis produces either a software requirements report or a conceptual model, and systems design produces a mapping for software code implementation. The traditional modelling methods provide a language for model representation, and heuristics to guide one's use of the language. This chapter provides an overview of the object-oriented approach in systems analysis and design.

2.1 Modelling in General

We use models to capture particular aspects of a problem domain for the purpose of analysis and/or design. A model is defined as "an abstract representation of reality that excludes much of the world's infinite detail. The purpose of a model is to reduce the complexity of understanding or interacting with a phenomena by eliminating the detail that does not influence its relevant behavior. Therefore a model reveals what its creator believes is important in understanding or predicting the phenomena modeled" (Curtis 1992). Models are also known as scripts (Wand and Weber 1995). Models or scripts are generated through grammars which employ a set of symbols for representation. The models we work with provide a semantic mapping from modelling constructs to the problem domain. Wand and Weber (1995) term this manifestation of meaning the deep structure of an information system.

Zhao (1995, p. 5) defines an enterprise model as "a symbolic system which captures the essential and abstract relations and properties, both structure and dynamics, of an organization, usually as a whole." Enterprise modelling produces an abstract representation of a business that is used to reason about the function of an organization. For our purposes, the model may be used to represent either a complete organization or a part of it.

2.2 Modelling and Systems Analysis

Analysis is the understanding of a system's behaviour. More formally, the Houghton Mifflin Dictionary (1980) states: "[Analysis is] the separation of an intellectual or substantial whole into constituents for individual study." In terms of enterprise modelling: "[A]nalysis calls for examination and study of the existing state of affairs in a given business area of the enterprise" (Olle et al. 1991, p. 15). Operationally, "[s]ystems analysis is the study of a system for the purpose of understanding and documenting its essential characteristics" (Embley et al. 1995, p. 19).

Traditional systems analysis methods focused on modelling information content and system components for software engineering purposes. The emphasis was on soliciting the requirements for a software design. The analysis process results in a provision of a list of software deliverables. This focus is evident in traditional structured analysis techniques (DeMarco 1979, Gane and Sarson 1986, Yourdon 1989) which specifically states the software development intentions of these techniques. This software engineering focus has continued in the development of object-oriented analysis methods (Booch 1993, Coad and Yourdon 1991a, Coleman et al. 1994, Rumbaugh et al. 1991).

OOEM models organizational activities. Objects in an OOEM model represent things in an organization, and do not represent software components. Parsons and Wand (1993) argue that "the key to a commonly accepted and natural object paradigm of systems analysis should be a view of objects as representation constructs, independent of implementation considerations" (p. 1). Furthermore, Rumbaugh et. al. state, "Object-oriented development is a conceptual process independent of a programming language until the final stages. [It] ... is fundamentally a new way of thinking and not a programming technique." In accordance, a software system may be represented in an OOEM as an object.

2.3 Modelling and Systems Design

Design involves the creation of a plan for a system that must have a certain specified behaviour. Traditional systems design involves generating a detailed plan for the construction of a software artifact. The results from systems analysis provides the input for creating a model which consists of modules or object classes that guided a programmer's implementation of the software code.

Design, from this paper's perspective, is not restricted to software alone. The design process begins from the enterprise model, and may eventually works its way down to software construction. At the end, the process produces a design model or specification.

2.4 Object-Oriented Analysis and Design

Object-oriented modelling is an approach to modelling a problem domain using object-oriented constructs. The approach is centred on the concept of an object. With the proliferation of proposed methods, the term object has been defined and interpreted in many ways, and "there is some dispute about exactly what characteristics are required by an object-oriented approach" (Rumbaugh et al. 1991, p. 1). However, there is an implicit basic agreement that all objects embody an abstraction (Synder 1993). An abstraction is defined as "a simplified description, or specification, of a system that emphasizes some of the system's details or properties while suppressing others. A good abstraction is one that emphasizes details that are significant to the reader or user and suppresses details that are, at least for the moment, immaterial or diversionary" (Shaw 1984).

Traditional methods model software requirements and implementation. Object-oriented analysis (OOA) is used to gather the software requirements to plan a software implementation plan. OOD produces a model built from OOA results that encompass implementation considerations. OOEM approaches the problem domain from a different angle. Instead of modelling from the perspective of software development, OOEM is used to develop models at the organizational level.

In the OOEM approach, the object construct represents an organizational unit, function, or role. A role is relative to what you ask the object to do. For example, if we had objects called clerk and receptionist, these two roles may be played by the same person. We would view these as two objects that represent two organizational roles.

In our model, information technology can serve two roles:

1) to process knowledge	- resources which reflects knowledge
	- services that operate on these resources
2) to provide communication	- information carried or requests
	- implies shared resources.

The following sections provide a summary of three well known object-oriented modelling approaches. These approaches include Coad and Yourdons's OOA/OOD, Coleman et al.'s Fusion, and Jacobson et al.'s use case driven approach. These three were chosen to provide a brief survey of general trends in system modelling. Both Coad and Yourdon's OOA/OOD, and the Fusion approach, model on the basis of creating a conceptual model of the problem domain for the software environment. This approach is similar to modelling in the database domain where one creates a semantic entity-relationship model of a problem domain (typically conceptual analysis model). This model is then mapped into a data (typically relational) design model. The approach is centred on software design and implementation. Jacobson's use cases approach provides one of the few methods that emphasizes the modelling of organizational activities.

However, as with most approaches, these three lack an ontological foundation of principles and rules that can be used to guide the construction of models. Because traditional methods lack this basis for analysis modelling, they do not provide methods to proceed from an organizational analysis model towards design.

2.4.1 Coad and Yourdon's OOA/OOD

Coad and Yourdon's (1991a) OOA method uses four basic modelling constructs: class & object, attribute, service, and class relationship. The method consists of five primary activities: finding classes & objects, identifying structures, identifying subjects, defining attributes, and defining services. These activities provide a structure for building an OOA model which consists of five layers: subject layer, class & object layer, structure layer, attribute layer, and service layer. The subject layers serve as a partitioning mechanism; the class & object layer captures classes and objects; the structure layer captures inheritance and whole-part structural relationships; the attribute layer captures attributes and instance connections between classes & objects; and the service layer captures methods/services and message connections between class & objects.

Coad and Yourdon's (1991b) OOD model is an extension of the OOA model. OOD uses four additional components which include the problem domain component, human interaction component, task management component, and data management component. The problem domain component houses the OOA results, and it is used to improve and add to the OOA results during the OOD phase. The human interaction component captures how a human will command the system and how the system will present information to the user. The task management component is used to design multitasking requirements for the system. The data management component provides the basis for storage and retrieval of objects from a data management system.

2.4.2 Jacobson's Object-Oriented Business Engineering Based on Use Cases

Jacobson (1992) proposes a modelling technique called *object-oriented business engineering* based on the notion of use cases. The technique consists of two interrelated models: the use case model and the object model. Use cases identify, at a high level, how a customer uses the company through business processes. Each way of using a business is represented as a use case. The object model clarifies the inner workings of a company's processes, products, services, resources, and their relationships.

Objects are classified into three types: interface object, controls object, and entity object. An interface object represents the way in which the business system communicates with the external environment. They portray a set of operations which have direct contact with the outside world. A control object represents a set of tasks which must be performed during a flow of events. They represent a set of operations, like the interface object, but do not have direct responsibilities for contact with the business environment. An entity object represents occurrences of such things as products, deliverables, documents, and other things handled in the business that are used/consumed during a flow of events (p. 340).

When constructing a business model, one starts by generating a use case model. This model describes what the business does. Then for each identified use case, an object model is created. The object model describes the inner most core of the business. One identifies the business' use cases through the environment (i.e. actors). Every stimulus from the actor to the business implies a triggering of a use case; and every stimulus sent to a use case implies a triggering of an object. The process can be summarized as such:

1. Build a Use Case Model
    - find actors (e.g. customer, business partner, supplier, government, subsidiary, etc.)

- find use cases
- prioritize use cases
- describe use cases
- select metrics and review

2. Build an Object Model
   - find subsystems
   - describe use cases in relation to subsystems
   - find objects

Jacobson's approach proceeds to a logical design by forming use cases for the software development process. A systematic algorithm is presented for developing an information system model by using the business object model as the basis. One starts identifying information system use cases by starting with the interface and control objects in the business system object model. The next step is to isolate business interface and control objects that will use the information system. If the business system object has responsibilities that involves the use of the information system, one would identify an information system use case for it. These use cases will form the basis for the information system object model (p. 273).

2.4.3 Object-Oriented Modelling with Fusion

Fusion (Coleman et al. 1994) is an integration and extension of existing object-oriented approaches, and is based on a set of diagramming techniques (notations) for describing analysis and design decisions. Fusion consists of three phases: analysis, design and implementation. The steps in each phase provide a sequence for decisions. The deliverables of each step are inputs for the next step. For each step, Fusion provides rules for checking the consistency and completeness of a model as it develops.

The analysis phase defines the intended behaviour of the system by using a customer supplied initial requirements document for input. Models in this phase describe classes of objects; relationships between classes; operations that can be performed on the system; and allowable sequences of those operations. The analysis is intended to provide a specification document.

The design phase produces models that show how system operations are implemented by interacting objects, references between classes, inheritance relationships, attributes of classes and operations on classes. The result of the design phase is an architecture document.

The analysis phase ends with operation models and design begins with making object interaction graphs and is finished by making inheritance graphs. Descriptions of the modelling techniques are summarized below:

Analysis Models

1. Object models - define the static structure of the information in the system. These represent the concepts in the problem domain and their relationships. The object model notation is based on the extended entity relationship notation.
2. System object model - a subset of an object model that corresponds to the system in development. It is formed by excluding all the classes and relationships that belong to the environment.
3. Interface models - define the communication of the system with the environment. The description is in terms of events, and the change of state that the events cause. The interface of a system is composed of the set of system operations which it can receive and the set of events that it can transmit. Different aspects of behaviour are described in two models:
   a) Life-cycle models characterise the allowable sequencing of system operations and events. A life-cycle expression defines a pattern of communication by showing the allowable sequences of interactions that a system may participate in over its lifetime.

b) Operation models describe, through the use of preconditions and post conditions, the effect of each system operation in terms of the state change it causes and the output events it sends.

Design Models

1. Object interaction graphs - constructed for each system operation. They define the sequences of messages that occur between objects to realize a particular operation.
2. Visibility graphs - show the reference structure of classes in the system. It shows the communication paths that are required for objects to have access to other objects (i.e. message sending). The following is identified for each class: objects that class instances need to reference and appropriate kinds of reference to those objects.
3. Class descriptions - one class description is made for each class. It describes both the class methods and attributes. These descriptions serve as foundations for implementation.
4. Inheritance graphs - show generalization-specialization relations between classes.

3. Object-Oriented Enterprise Modelling

Object-oriented enterprise modelling (OOEM), based on Wand and Woo (1993) and Zhao (1995), presents a method for constructing object-oriented models of organizations grounded on ontological principles. OOEM provides a formalized set of object-oriented analysis (OOA) rules, a request propagation algorithm, and a model representation technique. Because discussion of design will expand on only the OOA rules and propagation rules, only these two topics will be summarized here. A sample of the model representation technique is provided in Appendix II (i.e. with external and internal object templates). A discussion of modelling principles, ontology, and modelling constructs will provide the foundation for principles for object-oriented design. Modelling principles specify what type of model we are trying to build. Ontology provides a basis for what there is to model in the real world. Modelling constructs provide us with a set of constructs that we can use to represent reality. Ontological constructs can be mapped to their object-oriented counterparts, and thus lay out the foundation for object-oriented modelling. The OOA rules provide the properties or characteristics for a correct OOA model, and instruction for composing a model.

3.1 Modelling Principles

The following modelling principles lay out the basic assumptions for the type of model that the OOEM method is intended to build (Wand and Woo 1993). The first two principles are general statements about the relationship among information systems, organizations, and models. Principle 3 and 4 are construct independent statements about models. Principle 5 ties the object construct to modelling; and principle 6 sets the boundary or *scope* of a model.

General

1. Information processing activities are part of the regular activities of the organization. The OOEM method models both information-processing and non-information processing activities.

2. An information system is a representation of another system (the real world system). An information system is an abstraction or model of some part of the real world as perceived (existing or imagined). A model, such as an OOEM model, uses constructs that directly map to the problem domain. Such a model emphasizes the *meaning* of the real world system that is captured.

Basic Constructs

3. The modelled system is viewed as made of clearly defined (discrete) components. This principle ensures that a model does not contain components that have ambiguous meaning.

4. All actions in the system result from external stimuli. Thus, the purpose of a system is defined by the external stimuli to which it responds. Internal system events are triggered by external stimuli.

5. Every organizational component and every thing interacting with the system is modelled as an object. The object construct is the basic component for building models.

Minimality

6. Only aspects directly relevant to the purpose of modelling the system should be modelled. Relevance is determined by the external stimuli to which the system has to respond. The model boundary is limited by the requests from external objects.

3.2 Ontological Foundation

Ontology, a branch of philosophy (metaphysics), is the study or theory of being (i.e. basic traits of all reality). Bunge states, "Metaphysics studies the generic (nonspecific) traits of every mode of being and becoming as well as the peculiar features of the major genera of existents (1977, p. 5). Metaphysics can render service by analyzing fashionable but obscure notions such as those of a system, hierarchy, structure, event, information..." (1977, p. 24). Bunge's ontological approach (1977, 1979), which has been adapted to information systems (Wand and Weber 1995), is used as the foundation for OOEM (Wand 1989). Such an approach provides a base of reference on how to model the world. Because an enterprise model captures information about organizational things such as customers, products, employees, and equipment, a theory of the nature of things can function as the foundation for enterprise modelling. Ontology provides the concepts for how we can reason about the world, and thus we will use ontology as the basis to model and talk about organizational activities. The following is a summary of the ontological principles which serve as a basis for modelling (Wand and Woo 1993):

- The world is composed of things that possess properties
- Attributes are characteristics humans assign to things
- Every property can be modelled as an attribute
- Everything abides by laws which are invariant relations among properties of the thing. These laws limit the possible states and state transitions.
- Systems are interacting things
- Everything changes and every change is a change of states of things

3.3 Modelling Constructs

Modelling constructs provide a way to represent something in the world. Ontology provides constructs for describing the real world. These constructs can be mapped onto object-oriented constructs, and allow us to represent and reason about the problem domain using object-oriented concepts and constructs (Wand 1989).

3.3.1 Ontological Modelling Constructs

Wand and Weber (1990) adapted Bunge's ontology to information systems. The ontological constructs (Wand 1989) for IS were categorized into four categories as described by Zhao (1995). The constructs are listed below and are further described in Appendix I:

Static model of a thing - thing, property, state, composite thing, law, class

Dynamic model of a thing - events, transformation, history

Static model of a system - coupling, system, composition, environment, structure, subsystem

Dynamic model of a system - stable state, unstable state, external event, internal event, well-defined event, poorly defined event

3.3.2 Object-Oriented Modelling Constructs

There are various object-oriented analysis and design methods that present their own basic constituents of object-oriented constructs. Although there is no agreed upon standard, there are constructs that are consistently used throughout most methods. Constructs such as object, attribute, operation/function, hierarchy, and message are used in many of the methods.

Although these formal methods may not present a uniform set of concepts, the idea of encapsulation and objects are at the center of most methods. Informally, objects are packages of both attributes/state variables and functions/operations. The object construct entails the encapsulation of attributes (state variables) and operations, by forcing access to these variables to go through its publicly accessible mechanisms. Objects communicate with each other by sending messages to one another. OOEM uses the concept of an object as the starting point to build a set of object-oriented constructs from the set of ontological constructs.

3.3.3 Mapping Ontological Constructs to Object-Oriented Constructs

Mapping ontological constructs to object-oriented constructs provides us with the object constructs we can use to reason about a problem domain. Ontology provides a means of justifying or reasoning for why certain object-oriented concepts should be included in an enterprise model, and a meaning for each of these concepts.

3.3.3.1 Objects and Things

A thing is considered the basic unit of existence in our ontological model. In accordance, OOEM asserts that the world is made of objects. The object is the unit of analysis and its basis is rooted in the ontological principle that states that the world is composed of things. An object is defined as a model of a substantial thing that interacts with other things (objects) in the problem domain (Zhao 1995, p.12) (See also Wand 1989). Object types or classes represent roles that are played in an organization in an OOEM. A role represents a functional schema or view, and thus an object can be modelled as a functional schema. A functional schema is a set of attribute functions that represent intrinsic and mutual (shared) properties in objects. The model of a thing (its functional schema) reflects the purpose of the modelled thing. Objects and their dynamics are described by their states (attributes), state changes (services/methods/operations), and object interactions (external events/requests for service).

Ontologically, a class is a set of objects that have common properties. The class corresponds to the concept of a natural kind in ontology. A natural kind is a group of things that share the same laws and properties. In OOEM, the class construct is used to refer to a group of objects that share the same attributes and services and hence play the same role. Individual objects are commonly called instances. OOEM uses the class construct in its model representation even if only one object is in the model.

There are two general types of objects: internal objects and external objects. Internal objects are considered part of the system, and must provide at least one service in response to an external request. External objects are objects that interact with the system but are not considered part of it. They can either generate requests or be requested by the system to perform a certain action.

3.3.3.2 Attributes and Properties

Attributes model properties of things. An attribute is a characteristic that one assigns to an object. Such an attribute would reflect a property of the object. An object property is comprised of state and state transformation laws (functional schema). Attributes describe the state of objects. Attributes represent both the state of the object itself and its knowledge of the problem domain (which is part of the state). OOEM divides attributes into two type: internal attributes and interface attributes. Internal attributes are not publicly known to other objects. They can only be accessed or changed through the services of the owner object. Interface attributes are publicly accessible by other objects and are the means for communication.

Internal attributes model the intrinsic properties of a thing. Internal attributes need not be included in an object model since it is sufficient for other objects to interact with an object through the interface attributes. Internal attributes are usually included in the internal view of the model to show the additional knowledge of the analyst.

Interface attributes model mutual properties of things. They are shared by all communicating objects. Interface objects are the means by which objects communicate with one another. An object communicates with another by sending a request to another object. The request modifies the interface attributes of the recipient, and thus changes the state history of the recipient. This change in the interface attributes triggers a service that results in further state transformations. This type of object interaction reflects the ontological concept of coupling.

Interface attributes are divided into two types: incoming interface attributes and returning interface attribute. Incoming interface attributes model messages that sent from a requester to the recipient. Returning interface attributes model the message returned to the requester in response to the initial request.

3.3.3.3 Services and Lawful Transformations

Services model the state transition law of a thing. A service is a series of well-defined actions taken by an object to fulfill a request. A service may access or modify the objects own attributes, or generate requests to other objects.

3.3.3.4 Requests and Events

Requests model the interactions among objects. Interaction is the change of history of a thing as a result of the existence of another thing. An object interacts with other objects by sending requests. A request changes the interface attributes of the recipient object, and consequently triggers the corresponding service. The outcome of a request can change the recipient's state; the state of the sender; the state of both recipient and sender, or neither the recipient's nor sender's state.

Requests are categorized into two types: external requests and internal requests. Requests can be external or internal to either the system or object.

External system requests are those events that originate from external objects which are outside (environment) the modelled system. External object requests are those events that an objects receives due to its interaction with other objects.

Internal system requests model internal events of the system. Internal system events reflect the interaction of objects within the modelled system. Internal events to objects are those events that arise within an object by virtue of lawful state transformations. Such events are triggered, directly or indirectly, by external requests into the system. As a result of external system requests, the system becomes unstable, and may trigger a number of internal system requests to reach a stable state.

3.3.3.5 System of Interacting Objects and System

A group of interacting objects form an aggregate that is known as a system. A system of interacting objects models a real world system (existing or imagined). Object interaction models the coupling between things. Object interaction are modelled as either submitting requests or providing services. Objects in OOEM actively participate in organization's activities, and are not just subjects about which information is kept. OOEM's emphasis is on modelling aspects relevant to a model's purpose. External system requests define the services, and thus the purpose, of the system. Ontologically, such a request from an external object would cause a state change in an internal object through the modification of a mutual attribute, and cause both the system and affected object to go into an unstable state. As the object proceeds with its internal operations to satisfy the request, other mutual attributes may change (within the bounds of state laws). This change will be detected by other object which share these attributes and may make them go into an unstable state. This course of state transformation propagates throughout the system until the system reaches a stable state. This process is referred to as request propagation.

3.3.3.6 External Objects and System Environment

External objects model the system environment. External objects provide the requests that define the purpose of the modelled system or enterprise.

The following table summarizes the mapping of ontological constructs to object-oriented modelling constructs:

Object Constructs	**Ontological Constructs**
Objects	Things
Classes	Natural Kinds
Attributes	Properties
Internal Attributes	Intrinsic Properties
Interface Attributes	Mutual Properties
Services	Lawful Transformation
Request	Event
External Request to Objects	External Events to Things
External Requests to the System	External Events to the System
Internal Requests to Objects	Internal Events to Objects
Internal Request to the System	Internal Events to the System
System of Interacting Objects	System
External Objects	System Environment

Table 1 Mapping of Object Constructs to Ontological Constructs

3.4 Object-Oriented Analysis Rules

This section provides a summary of the OOEM modelling rules. There are a total of seven ontologically based rules that define the course of modelling a domain (Wand and Woo 1996).

3.4.1 Rule #1: Scope Identification

The rule states that the analyst should define the purpose of the modelled system (i.e. the problem domain) in terms of organizational activities to be supported. This rule defines the boundary of the enterprise model. It distinguishes between the environment and the enterprise. The environment is represented with external objects that interact with the modelled system by sending external requests to the system. These requests define the scope of the modelled system. The problem domain becomes the scope of the model, and determines which aspects of the domain should be modelled.

3.4.2 Rule #2: Object Identification

The rule states that an object is included in the model if and only if it provides at least one service, or makes at least one request to the system. This rule determines the things (organizational units or actors) that should be modelled as objects. The rule ensures that every change is tied to a change of state of things and everything changes. An internal object is an object that is part of the system and provides at least one service. An external object is part of the environment that interacts with the system.

3.4.3 Rule #3: Service Inclusion

The rule states that a service is included in the system if and only if it is invoked by at least one request in the system. This rule determines which object dynamics should be included in the model. Such a request may come from either external objects or internal objects. Services reflect internal transformations (state changes) of things.

3.4.4 Rule #4: Attribute Inclusion

The rule states that an interface attribute must be used or affected by at least one service, and known to at least one other object. This rule determines which object attributes should be included in a model. An internal attribute must be affected by at least one service of the object and is unknown to other objects.

3.4.5 Rule #5: Attribute Ownership Rule

The rule states that there are no attributes that do not belong to any object. This rule determines which object is responsible for an attribute value. The value of every attribute can only be modified or obtained by the services of one object. This object is known as the *custodian* of the attribute. Other objects can only access or modify the value of an attribute through the actions of the custodian object. Properties always belong to things.

3.4.6 Rule #6: Aggregation and Decomposition

The rule states that a composite object is included in the model if and only if it provides services that are not provided by any of its components. This rule determines when to include composite (aggregation) or component (decomposition) objects in the model. When modelling the properties of a composite, include those properties not modelled in a component. A composite object must have emergent properties.

3.4.7 Rule #7: Generalization and Specialization

The rule states that if two or more object classes provide one or more common services, one can generate a general object class. This rule determines when to form super-classes (generalization) or sub-classes (specialization). A class (natural kind - See Appendix I) is defined by a set of shared properties (same laws). The general object class provides the common services, and is known as the super-class of the original classes. The original classes are known as sub-classes of the general class. All services provided by the super-class should be eliminated from the sub-classes. Alternatively, one can form a sub-class only if it includes new services and attributes with respect to the super-class. In accordance with Rule #1 - object identification, the super-classes and sub-classes must receive requests from other objects.

3.4.8 Service and Request Propagation

OOEM's service and request propagation algorithm provides a systematic method for applying the OOA rules to find and determine the objects to include in an enterprise model. Tracing the request propagation process identifies all the objects with their associated services, interface attributes, internal attributes, and request connections. In the process of performing a service, an object may generate requests to other objects. Request propagation begins with a request from an external object, and ends on external objects or on objects which do not need to propagate requests. The following provides a simplified summary of the recursive process:

1. Identify external clients - consider all things submitting requests to the system.
2. Identify requests from clients
3. Identify the object that receives the request - consider all things mentioned in the case that provide some useful services or request some service.
4. Identify the interface attribute - record for a change of state, request for state information.
5. Identify services - consider all actions taken by an object as a result of all requests submitted to it. Include only actions that are triggered directly.
6. Identify internal attributes used by services
7. Identify requests generated by a service
8. Recurs to step 3
9. Apply Rule #6 and #7

4. Object-Oriented Design

Whereas systems analysis encompasses both the study of current systems and the requirements analysis for a new system; systems design deals with laying out plans for systems development. Decomposition is a principal activity of systems analysis and design (Wand and Weber 1991). In analysis, decomposition is used to further break down a system into its components for further analysis. In design, decomposition is used to plan the construction of an aggregate from components. Thus, a major aspect of the design of an organization or information system will be based on the process of decomposition. Wand and Weber (1991) state, "Good decomposition is considered a major requirement and frequently the essence of good design" (p. 101). Furthermore, Booch (1993) says, "Object-oriented design is a method of design encompassing the process of object-oriented decomposition…" (p.39).

More formally, the Houghton Mifflin Dictionary states, "[Design is] the invention and disposition of the forms, parts, or details of something according to plan. [It is] a drawing or sketch" (1980). In term of information systems, design involves the creation of a plan for a system that must have a certain specified behaviour (Wand and Weber 1991, p. 102). From these two definitions, we see that design involves a model of a thing, the decomposition of a thing, and specified bounded behaviour of a thing. Once the foundation of decomposition is established, we can use the objected-oriented constructs to express a design model.

4.1 Foundations for Design Modelling

The model from the requirements analysis feeds the process of design. Once the overall aggregate behaviour of a system is defined, the system is decomposed into components. Each component posses behaviour and interacts with other components to provide the aggregate behaviour of the system. Each component is itself a subsystem that can be decomposed into finer components. Decomposition will be based on a set of common formal foundations, and the process of finding the parts of a system is guided by the process of request propagation. The fundamental rules and principles for analysis modelling will continue to be applied in the design modelling process.

4.1.1 Modelling Principles for Design

The modelling principles used to lay out the basic modelling assumptions for object-oriented enterprise modelling will provide the starting point from which we will form modelling principles for design. From those analysis modelling principles, we can distil principles that are more specific to design. These principles specify the type of model we are trying to build as we progress from the analysis model to the design model. We focus on the following list of principles:

Basic Constructs

1. The modelled object is viewed as made of clearly defined components. This indicates that self-contained objects are systems that can be decomposed and analyzed in further detail by breaking them down into finer components.
2. All actions in the object result from external stimulus or requests. The purpose of the object is defined by the external stimuli to which is has to respond. This principle indicates that components of objects can be discovered by following the request propagation since the purpose of the component objects are defined by external requests for services.
3. Every system component and everything interacting with the system is modelled as an object. This suggests that objects are composed of finer objects. To examine an object without decomposing it, will require the examination of the object's attributes and services.

Minimality

4. Only aspects directly relevant to the purpose of modelling the object should be modelled. The purpose of the modelled object is defined by external stimuli to which the object has to respond. This principle implies that only component objects that are relevant to the scope of analysis should be included in the model. This is the principle of minimality.

The presented modelling principles set out the basic assumptions from which we can proceed with design through decomposition.

4.1.2 The Fundamentals of Decomposition

> [C]omplex systems, [such as information systems], might be expected to be constructed in a hierarchy of levels. The basic idea is that several components in any complex system will perform particular subfunctions that contribute to the over-all function.... To design such a complex structure, one powerful technique is to discover viable ways of decomposing it into semi-independent components corresponding to many functional parts. The design of each component can then be carried out with some degree of independence of the design of others, since each will affect the others largely through its function and independently of the details of the mechanisms that accomplish the function" (Simon 1981, p. 148).

Decomposition fits the ontological concept that things can be composed of other things (i.e. interacting things form systems and aggregates). In other words, things can associate to form composite things. Thus, decomposition provides a method for analyzing a thing in finer detail. As a thing is decomposed, its state and behaviour are also decomposed. As a result, the state and behaviour of the aggregate thing are divided among its components. One may infer from ontology that the decomposition of things is a possible route for systems design since one would be defining the interacting components things to form the aggregate system. The fundamentals of design decomposition do not prescribe a method for producing the ideal design model. In fact "there is no reason to expect that the decomposition of the complete design into functional components will be unique. In important instances there may exist alternative feasible decompositions of radically different kinds" (Simon 1981, p. 149).

More formally, the approach that will be taken is summarized in these following points (Wand and Weber, 1991):

1. A thing or system is viewed as:
    - A whole thing - an object
    - A composite made of other things - an object composed of other objects
2. A good decomposition reflects the dynamics of the system - component objects, as a whole, preserve the services of the aggregate object
3. The decomposition of an information system should reflect a decomposed view of the represented system.

General principles (Wand and Weber, 1991):

- A good decomposition is defined with respect to behaviour under a given set of external events
- A good decomposition is related to things that are "well-behaved" and "well carved out" of their environment. A well-behaved thing responds to each external event in a predictable way. A well carved out thing has some state variables that are determined by the thing only - encapsulation of internal state.
- A good decomposition is an approximation.
- Same principles for decomposing a system and for viewing it as separated from its environment.

The necessary conditions for a good decomposition for a given set of external events (Wand and Weber, 1995):

1. Determinism: For a given set of external events at the system level, a decomposition is good only if for every subsystem (at every level in the level structure of the system) an event is either (a) a specified external event, or (b) a well-defined internal event.
2. Minimality: A decomposition is good only if for every subsystem (at every level in the level structure of the system) there are no redundant state variables describing the subsystem.
3. Losslessness: A decomposition is good only if every emergent variable in the system is a function of properties of at least one subsystem in the decomposition. This means that one should be able to deduce the behaviour of a whole systems from the interactions and events of the components.

Given that we have established what is decomposition of an object (what), we may now proceed to the process of decomposition (how).

4.1.3 The Process of Decomposition

Object-oriented design involves the breakdown of a large aggregate object into its component objects. Such a process can be executed recursively for each component object. The decomposition begins with finding objects that process the external requests. By tracing through one of these requests, the scenario description will provide the details on how the component object processes the particular request. Just as for analysis modelling, one goes through the request propagation process for design. If the initial object needs services it alone cannot provide, it will "spawn" a request to another object for the required service. At the end of the request propagation trace, we will have a model of all the objects, with their attributes and service, that are needed to process the specific request. When this tracing is completed for all requests, we will have a complete decomposed model of the aggregate object. In analysis, decomposition is used to reveal the inner workings of objects. In design, decomposition is used to create a plan for how objects would carry out their inner operations.

4.2 Design Modelling Constructs

Design requires some further elaboration for some additional modelling constructs for the decomposition process. The decomposition process makes use of two types of objects: the delegate object and the static object. These two categories of objects still correspond to the ontological concept of a thing, but have been classified for the types of roles they play.

A delegate object is an object which has been created to act as a representative or a service provider for the delegator object. A delegate allows one to transfer responsibilities for attributes and services from one object to another. The delegate object provides the primary means for the transition from an analysis model of an organization to a logical design model of a software system. When we delegate responsibilities to an automated system, we relieve the delegator from having to perform former services and knowledge keeping. The delegate object is used to capture this essential meaning.

A static object is an object with only state (interface attributes) and no services. Its state is changed by other objects by altering interface attributes. It only undergoes external events. In comparison, dynamic objects undergo both external and internal events. During a decomposition of an object, one may find that a dynamic object manipulates a static object in order for the dynamic object to provide its services. For example, we may find that a clerk (dynamic object) looks up and changes the contents of a filing cabinet (static object) in its normal course of operations. (N.B. Static objects will not appear in the analysis - OOEM.)

4.3 Object-Oriented Design Rules

Given the fundamental principles and conditions for design and decomposition, we can derive rules for object-oriented design. These rules describe properties that a good object model should possess. Unlike rules of thumb, these rules are based on a formal foundation of design, and are properties that a good model should possess.

1 Object Decomposition Rule

Each request received by a composite object is processed by at least one of its component objects.

2 Delegation Rule

Each delegation object is modelled with a subset of the attributes and services of the client object.

3 Static Object Rule

Each static object either uses (has its state read) and/or is manipulated (has its state changed) by at least one dynamic object.

The decomposition rule provides the means towards design since we use decomposition to discover and define the components of a system. The delegation rule provides the transitional link from a model of an enterprise to a model of an information system. The static object rule provides the states and events that need to be reflected in the database design of an information system.

The general sequence for using the rules begins with the decomposition rule; followed by either the delegation rule or static object rule in any order in iteration. We start with the decomposition rule since the designer requires a more detailed makeup of a system before he/she can begin applying either the delegate or static object rule in design.

The following mini example, about a university's registrar's office, will be used to illustrate the design rules:

Students submit a request to attend a course to the registrar's office. The registrar's office checks if the student has the right to enrol (i.e. proper academic standing, no default on tuition) and then submits the approved requests to the faculties. If the request is rejected, the student is notified.

In the faculty, students are assigned to course sections - pending availability. If space is limited, priority is given based on program requirements and credit accumulated so far. The registrar's office is then notified as to the status of requests and section allocation. Student requests not approved due to space limitation are placed on a waiting list.

The registrar's office notifies students whether their requests have been approved or not by the faculty and on their fee payable based on the student's status and courses approved.

The notation that will be used to illustrate the mini-example consists of the following:

- object class - rounded rectangle
- object class name - top section of the rounded rectangle
- attributes - middle section of the rounded rectangle
- services - bottom section of the rounded rectangle
- requests - dotted arrows

The label at the tail end of an arrow represents a request, and the label at the arrow head represents the response to the request. In shorthand, a request for information (i.e. value from an attribute) is only represented with an arrow and request label. The returned response is implied in the display of the attribute that contains the requested information. No specific request for service attribute is necessary.

Figure 1 shows the initial OOA model for the registrar's office mini-case example. The external object - STUDENT - initiates the request propagation. The propagation trace provides the objects REGISTRAR'S OFFICE and FACULTY.

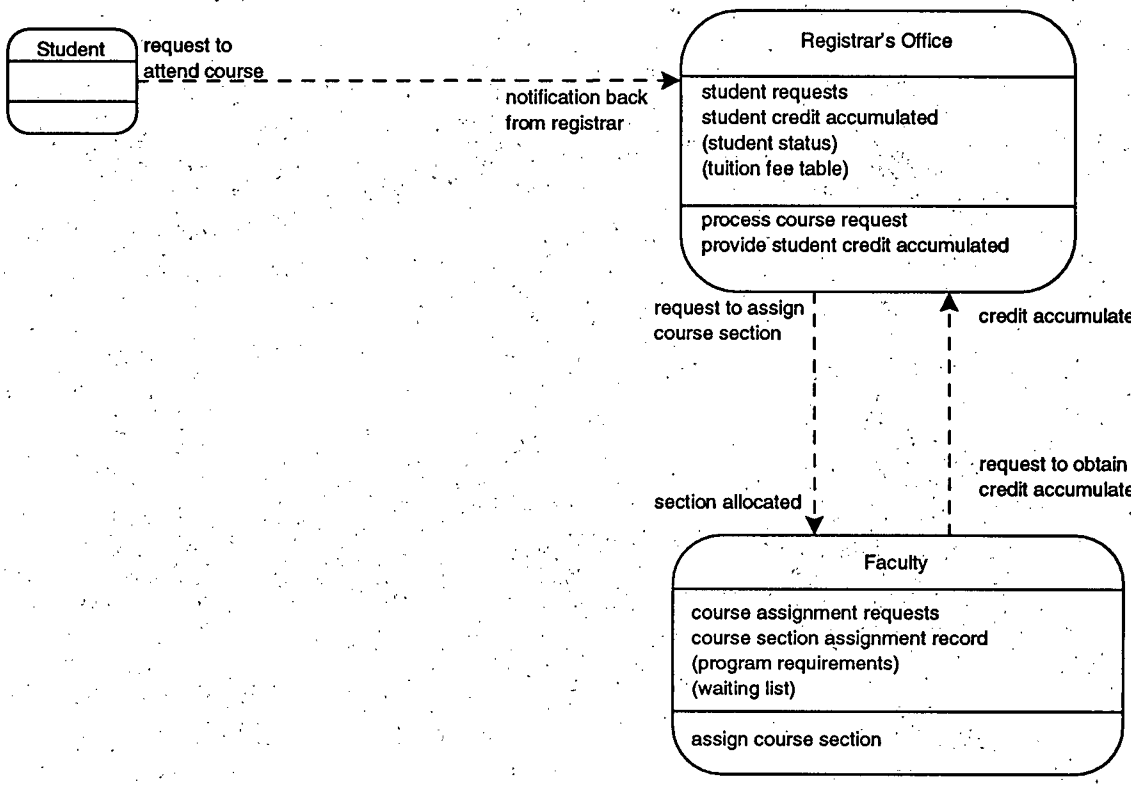

Figure 1 Initial OOA of the Registrar Case

4.3.1 Object Decomposition Rules

The composition characteristic of objects states that objects can be combined to form composite objects. The force that binds the component objects together to form a system is their interaction which is manifested as communication or requests for each other's services. In other words, the objects are dependent on each other in order to carrying out a request as a composite object. The decomposition rules enforce the idea that for each request that is made of the composite, there is a component object that acts as an interface to the service requesting object, and begins the initial processing of the request. The initial object may spawn other service requests to other component objects. This process may result in the decomposition of attributes and services of the composite object.

The decomposition rules maintains the continuity of an object when one switches from a whole object view to a decomposed object view. Every external object request must be serviced by at least one component. There are no requests in the whole object view that are not transferred to the decomposed object view.

Figure 2 depicts the four basic types of object decomposition. Figure 2(a) depicts a reallocation (division) of attributes and services by separating them as whole units among the component objects. Figure 2(b) depicts a reallocation of attributes which requires a decomposition of the service since only one service is used in the composite object. Figure 2(c) depicts a reallocation of services which requires a decomposition of the attribute since it is used by both services. Figure 2(d) depicts a decomposition of both an attribute and service in order for the composite object to be divided into its component objects.

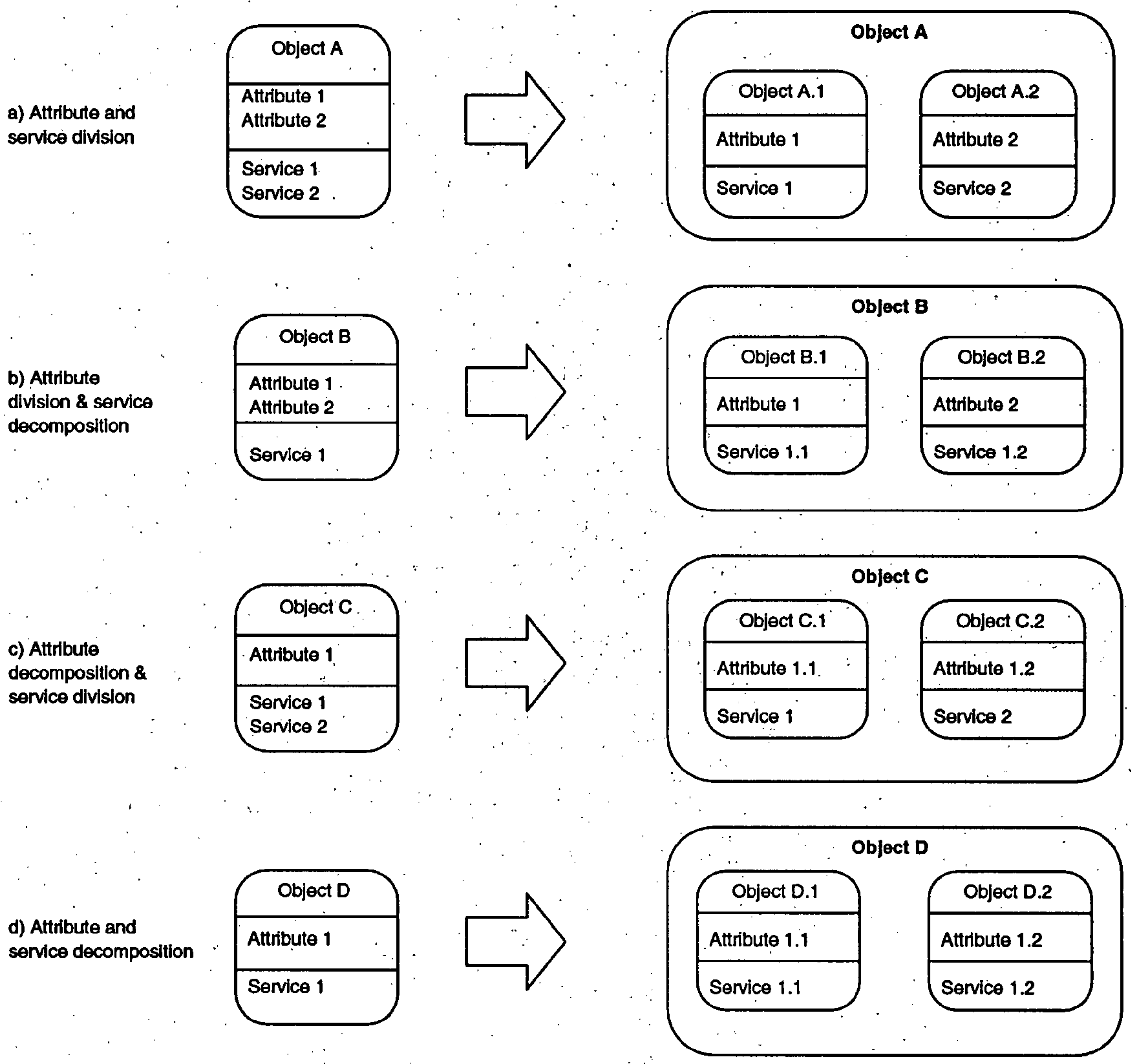

Figure 2 Types of decomposition

Figure 3 illustrates a decomposed view of the REGISTRAR'S OFFICE. It is composed of 3 objects: OFFICE CLERK 1, OFFICE CLERK 2, and the MAIL CLERK. Such a decomposition model is generated after a further probe into the working details of the REGISTRAR'S OFFICE. Such detail can be obtained either through an interview or a case description. As with the request propagation used in the OOA model, the focus of analysis is turned to a specifically chosen object (the REGISTRAR'S OFFICE in this case). This object becomes the scope of analysis. In Figure 3, OFFICE CLERK 1 is the one that handles all initial student queries. Requests are compiled and handed off to OFFICE CLERK 2 who is involved in the core activities associated with approving and inquiring whether a request to attend a course is accepted or rejected. After a *course request* is processed, the STUDENT receives a written confirmation or rejection through the mail. This mail package is prepared by a MAIL CLERK in the registrar's office after OFFICE CLERK 2 is finished with processing the *course request.*

The decomposed view of the REGISTRAR'S OFFICE object exhibits the necessary conditions for a good decomposition for a given set of external events (as discussed in section 4.1.2):

- Determinism - all events (requests) are either external or internal events.
- Minimality - all attributes are used by at least one service. There are no redundant attributes.
- Losslessness - one is able to deduce the behaviour of the registrar's office by examining the behaviour of the components.

The type of decomposition that Figure 3 exhibits, follows those patterns depicted in Figure 2 (a) (b) and (d). The attributes *student status, tuition fee table* and *student credit accumulated* are allocated to OFFICE CLERK 2 as whole units. In addition, the service *provide student credit accumulated* is also allocated to OFFICE CLERK 2 as a whole unit [pattern a]. The composite object's attribute *student requests* to the REGISTRAR'S OFFICE becomes decomposed into *student requests* (OFFICE CLERK 1), *course request* (OFFICE CLERK 2) and *mail request* (MAIL CLERK). The *process course request* service becomes decomposed into *process course request* (OFFICE CLERK 1), *process course request* (OFFICE CLERK 2), and *send mail* (MAIL CLERK) [pattern d]. OFFICE CLERK 2's service *process course request* uses both attributes *student status* and *tuition fee table* [pattern b].

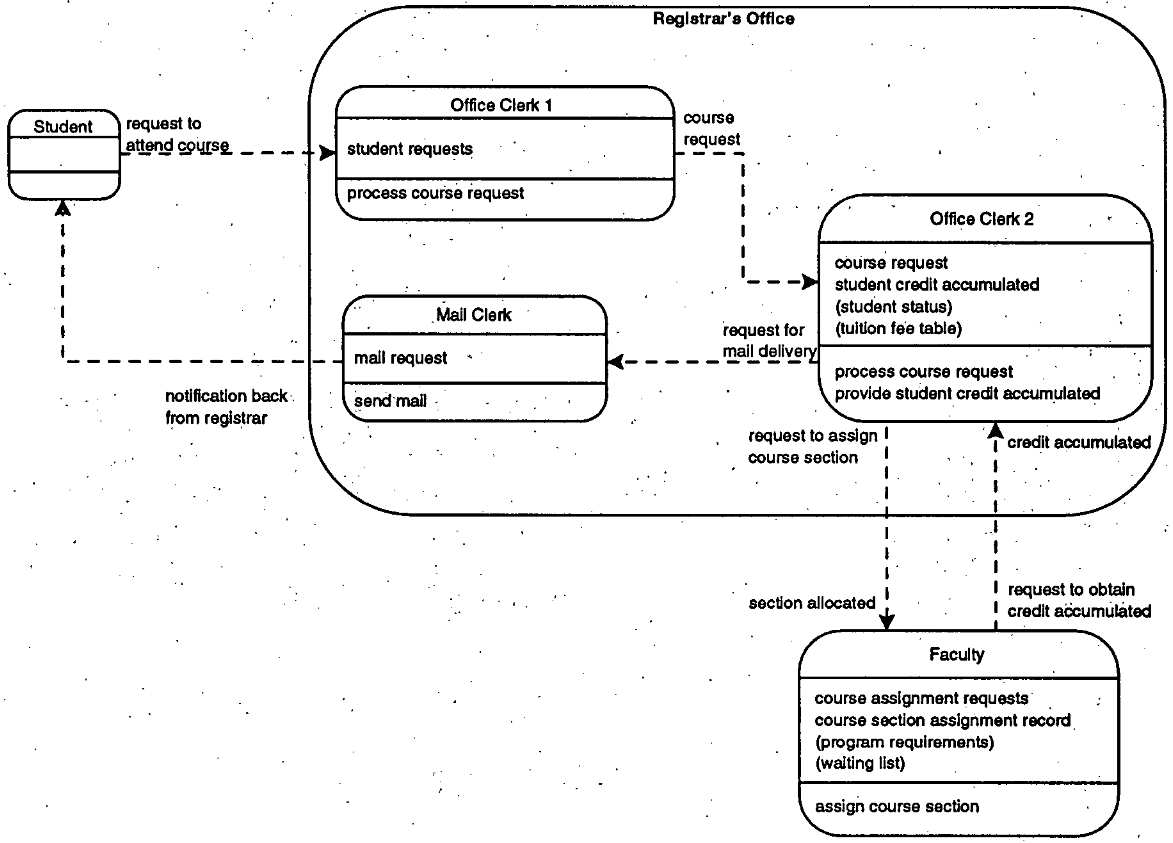

Figure 3 Decomposition of the registrar's office

4.3.2 Delegation Rule

A delegation object is used to transfer roles from a client object (delegator) to a server object (delegate). The concept of delegation that is use here is different from the notion of delegation used in object-oriented programming. In programming, delegation refers to passing a message onto another object in instances where an object could delegate responsibility of a message it couldn't service to objects that potentially are able (its delegates). Delegation is often applied in inheritance where an object inherits methods (services) from its superclass(es). The effect of inheritance allows an object of the subclass type to pass messages, for which it does not have a defined local method to handle, to an object of the superclass type (See Kim 1989, pp.9,33,99).

In a model, delegation provides a mechanism for one to start reasoning about information systems (i.e. automation, technology). The delegation rule provides the path for the transition from an enterprise model to an initial design model of an information systems. Each delegate object is a potential candidate component that can be used in the design of an information system. Delegates can be combined to form an information system, and their attributes provide the linkage to the database design. As one makes more refined requests to a delegate, the delegate's attributes would decompose into smaller data items.

The delegation mechanism involves reallocating or moving attributes and their corresponding services from a delegator object to a new object called a delegate. Thus, a delegate is derived from the attributes and services of the delegator. Delegation can be thought of as a type of decomposition. Instead of delegating attributes and services as whole units, one may also decompose the attributes and corresponding services (as depicted in Figure 2), and delegate decomposed or partial attributes and services.

Finding delegate objects is a design decision. The designer decides on whether or not to create delegate objects. One can start looking for potential delegates by looking for those activities that operate on attributes reflecting knowledge. An object or a set of highly interactive objects that are involved in information processing activities (actions that change the state of knowledge) and/or information transmission activities (services that primarily redirect requests) are candidates for delegation. Once roles have been delegated, these delegate objects become candidates for automation. As one proceeds with delegation, the decomposed view of the systems and/or subsystems will appear with new objects to carry the roles that have been reassigned. However, one would not delegate attributes and services that involve unstructured decision making since such tasks are not easily automated (i.e. certain decisions and tasks may still require human intervention).

The delegation rule ensures that when attributes and services are delegated to the server object, both the client and server remain internally coherent and cohesive with regards to the attributes and services. In other words, the rule ensures that the encapsulation (attribute custodian rule) of object states is not violated. This means that the behaviour and state of the system is not lost when roles are delegated. Behaviour and state of the systems remain the same before and after the delegation. The delegation rule can be refined into the following statements:

1. delegation of an attribute should be accompanied with services that use the attribute
2. delegation of a service should be accompanied with attributes that are used by the service
3. delegation of both attribute and services that are related

Figure 4 depicts OFFICE CLERK delegating the *student status*, *tuition fee table*, and *student credit accumulated* attributes to an object called STUDENT IS.

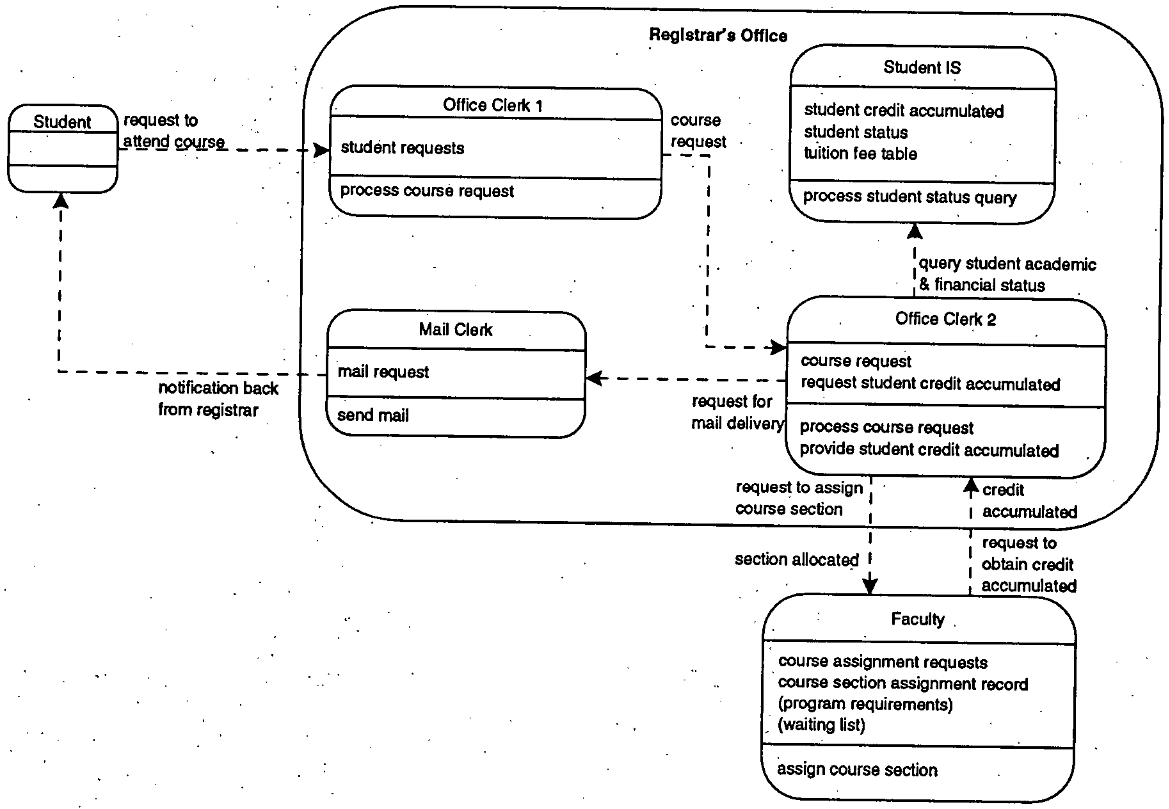
Registrar's Office
Student
request to attend course
Office Clerk 1
student requests
process course request
course request
Student IS
student credit accumulated
student status
tuition fee table
process student status query
query student academic & financial status
Mail Clerk
mail request
send mail
notification back from registrar
request for mail delivery
Office Clerk 2
course request
request student credit accumulated
process course request
provide student credit accumulated
request to assign course section
section allocated
credit accumulated
request to obtain credit accumulated
Faculty
course assignment requests
course section assignment record
(program requirements)
(waiting list)
assign course section

Figure 4 Registrar's office with a delegation object

Delegation can also be used to change the communication linkages between objects. Figure 5 illustrates the four basic patterns of delegated communication. These patterns represent degrees of automation. Figure 5(a) depicts the regular scenario of one object making a request to another objects. Figure 5(b) depicts a delegation that does not change any communication linkages. This illustrates the minimal level of automation. The delegate is used to store and retrieve information. Figure 5(c) depicts an object making a request to another object's delegate object. Figure 5(d) depicts a delegate object making or responding to another non-delegate object. Both (c) and (d) illustrate a medium level of automation. This type of delegation adds partial delegation of communication to the current delegation of information management. Figure 5(e) depicts a full delegation of object communication. A delegate object makes a request to another delegate object. This illustrates the allowance of full automation of both data processing and communication activities.

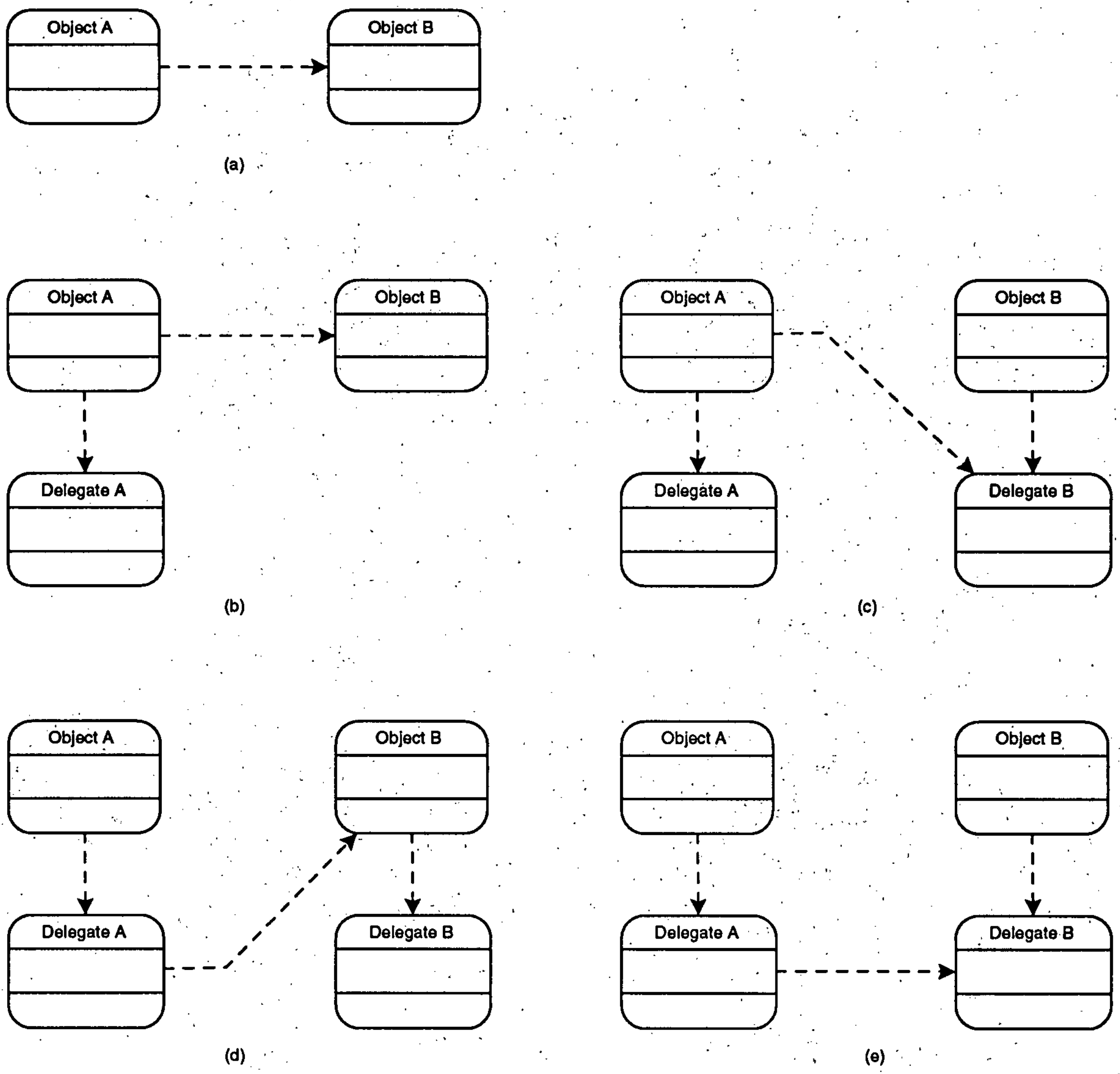

Figure 5 The 4 basic patterns of delegated communication

4.3.3 Static Object Rule

Static objects are objects which only possess state. Thus, by definition, such objects provide no services. Their states are used or changed by other dynamic objects. They represent things that do not participate or take part in the activities of an organization (i.e. they provide no services). These objects are not active in an OOEM model since OOEM is used to build models of organizations and not of information systems.

Ontologically, static object cannot occur because all things possess state and behaviour; however, they are possible with respect to the scope of a model. In a model, they may represent the internal and interface attributes of a dynamic object. A dynamic object's attributes reflect its own state or the state of another thing (i.e. knowledge about another thing). This other thing may be modelled as a static object if it doesn't provide services. One would model such an object if one is interested in tracking its changes of state, and the events that cause the state changes. When the state of a static object changes, the state of the tracking object (i.e. the dynamic object that keeps knowledge of the static object) should also change to reflect the static object's new state. A model with static objects provides a means to track or highlight the events which would alter the states of static objects, and thus indicate the need for corresponding events that must occur to bring about state changes to be made in the state tracking object (e.g. database of an information system) in order to keep the actual static object state and knowledge of that state synchronous or accurate. In addition, as one proceeds with decomposition, one may find objects which were previously designed to store knowledge. Such objects may also be modelled as static objects. Both dynamic objects, which track state information held in static objects, and static knowledge storages are primary candidates for automation through delegate objects.

In brief, static objects are a helpful indicator of the type of information that may be stored in a database since these objects hold state that we wish to keep. In addition, they help us learn which specific states in the modelled system are subject to change by which actions. Such information is important since it can be used in the future development of a database design.

Because state changes can only be made through a direct manipulation by a dynamic object, the rule requires that all illustrated static objects must be shown with at least one dynamic object that either reads or changes the state of the static object. The static object rule ensures that all attributes in a static object are used by at least one other dynamic object. Static objects that do not have their states changes are not modelled, in accordance with the principle of minimality, since they provide no additional information.

Figure 6 shows the static storage object RECORD CABINET. Its attributes are composed of *student course request records*. OFFICE CLERK 2 changes and uses the state preserved in the RECORD CABINET object. In addition, one may include STUDENT as a static object if the STUDENT's state represented in OFFICE CLERK 2's *student credit accumulated*, *student status*, and *tuition fee table* attributes went through state changes. However, these attributes are only used to answer queries for information, and do not go through state changes in this example's scenario, and is thus not represented as a static object. If the STUDENT was represented as a static object, there would be two STUDENT objects represented in the model. The rational for this situation is that the STUDENT would play two separate roles: one as an external object; and the other as an internal static object for which we track its state changes.

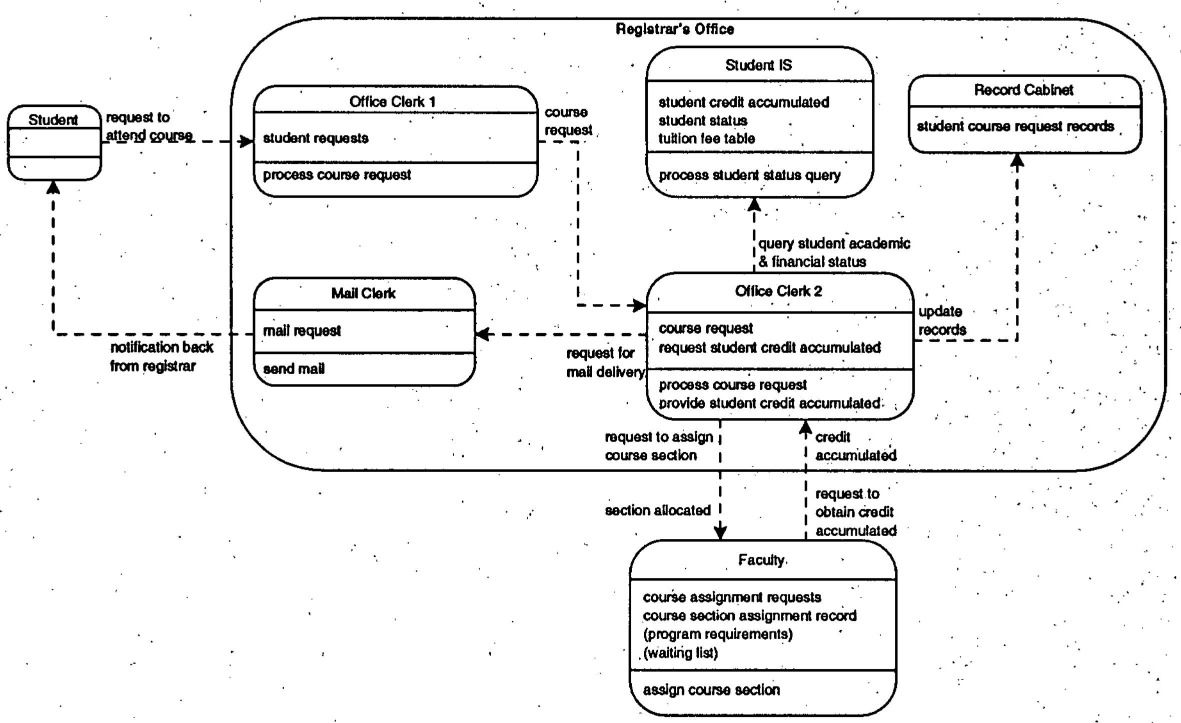
Registrar's Office
Student
request to attend course
Office Clerk 1
student requests
process course request
course request
Student IS
student credit accumulated
student status
tuition fee table
process student status query
Record Cabinet
student course request records
query student academic & financial status
Mail Clerk
mail request
send mail
notification back from registrar
request for mail delivery
Office Clerk 2
course request
request student credit accumulated
process course request
provide student credit accumulated
update records
request to assign course section
section allocated
credit accumulated
request to obtain credit accumulated
Faculty
course assignment requests
course section assignment record
(program requirements)
(waiting list)
assign course section

Figure 6 Registrar's office with a static object

4.4 Summary

The object-oriented design principles provide a means for one to reason about design. The principles provide three areas of rules: decomposition, delegation, and statics. The decomposition rules do not tell us when to decompose an object or system. The principles of decomposition only tell us the necessary conditions for a good decomposition. We don't have sufficient conditions since there exists no comprehensive theory on decomposition. However, the design principles do tell us when to delegate through the delegation rule, and they also tells us when to include and use static object with the static object rule. The three rules provide a means to approach the development of logical design models.

For the purposes of database design, the logical design model is tied to the entity-relationship model or detailed database design through the attributes of the delegate objects. These attributes reflect knowledge of static object states. The attributes reflect the persistent knowledge that can be kept in a database.

At the end of the logical design, the designer possess the following information:

- an initial design model of an information system composed of delegate objects
- a database that is held in the attributes of the delegate objects, and reflected in the states of the static objects. As one makes more requests to the delegates, the attributes will fragment and provide the detailed data for the database.
- the operations that need to be performed by the database. These operations are imbedded in the services of the delegate objects, and like attributes, will fragment as one makes more refined requests.
- references to the events or requests that will cause state changes in static objects, and thus indications of the linkages that will be needed to trigger corresponding changes to the state of knowledge held in a database.

5. Case Example: ACME Warehouse Management Inc.

The ACME Warehouse Management Inc. (Jacobson et al. 1992) case illustrates the application of the proposed object-oriented analysis and design rules. The case begins with an analysis of ACME Warehouse's operations through the development of an object-oriented enterprise model. Subsequent details of the business' operations are revealed through the process of object decomposition; whereby, the objects modelled in the initial enterprise model are successively broken down into their component objects. The design of the required system commences after the analysis has provided the elemental enterprise objects. The modelling notation used to illustrate the object models is similar to that of Coad and Yourdon (1991a). Objects are represented as rounded rectangles divided into three regions. The top records an object's name; the middle region records an object's attributes; and the bottom section records an object's services. Requests or messages are represented as dotted arrows that connect one object to another. In Zhao (1995), object templates are used to represent the OOEM model. For illustrative purposes, Appendix II provides the external and internal object templates for each object under the OOEM notation for Figure 7 and Figure 8. The initial case description is as follows:

ACME Warehouse Management Inc. offers storage facilities and redistribution services (between their different warehouses) across the nation. A customer can request space in a particular warehouse, request items to be transferred to another warehouse, or request withdrawal of items from a particular warehouse (even for items not stored there). Acme is planning on commissioning an automatic information system to support warehouse management. The idea is to offer the customers warehouse space and redistribution services with full computer support. The system needs to keep track of information on customers, items, warehouses, trucks, and warehouse places.

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

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

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

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

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

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

5.1 Object-Oriented Analysis

The description of ACME indicates that it is the customer that instigates the withdrawal process. Therefore, the internal ACME Warehouse objects can be discovered by following the request propagation that originates from the customer's withdrawal request. The customer is represented as an external object in the enterprise model (Rule 2: Object Identification Rule - external objects), and requests from this object defines the scope of the analysis (Rule 1: Scope Identification Rule). Figure 7 depicts the preliminary enterprise model of ACME Warehouse.

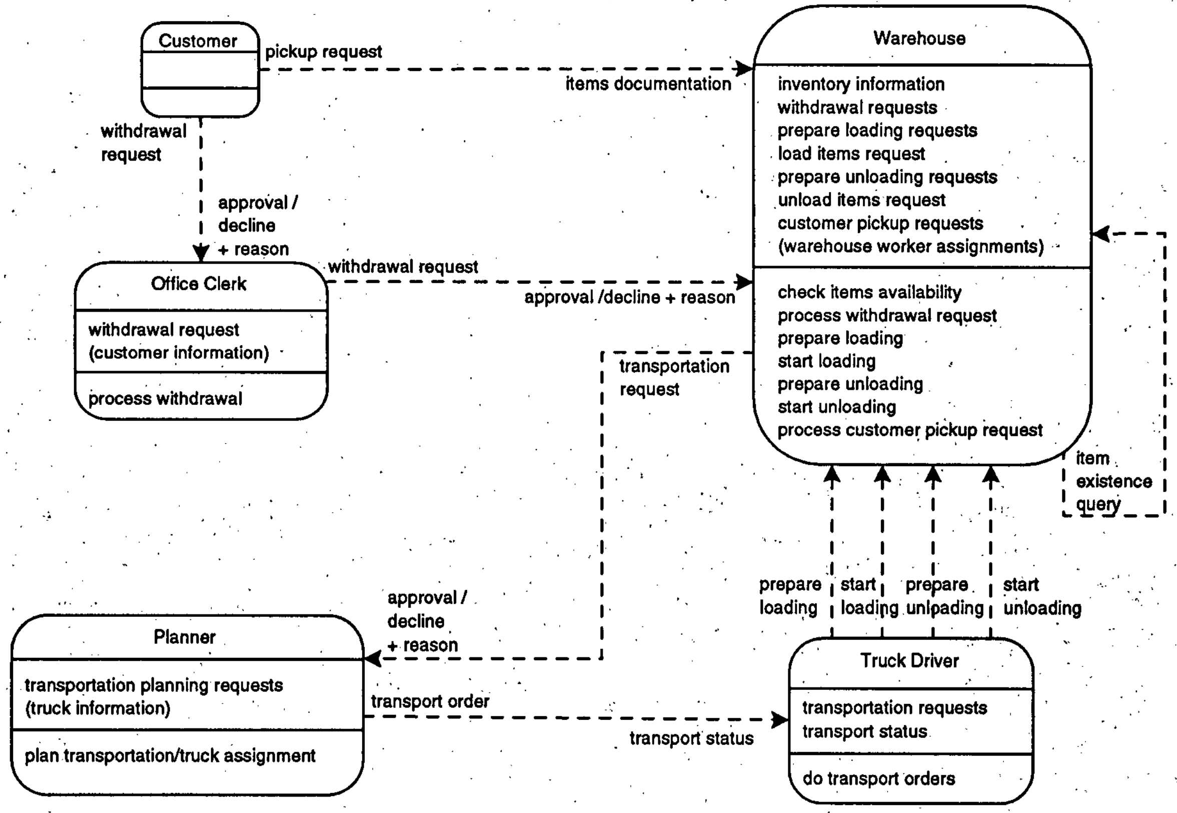

Figure 7 Initial Enterprise Model of ACME Warehouse

The CUSTOMER object initiates the *withdrawal request* by sending a request to the OFFICE CLERK object. If the request is verified, the request continues to propagate, and the OFFICE CLERK object would send a *withdrawal request* to the WAREHOUSE object. Although both warehouse manager and warehouse worker are mentioned in the description, these things are not represented in the model as objects. From the perspective of the OFFICE CLERK, he or she only needs to know that the request is passed to the WAREHOUSE. The OFFICE CLERK does not need to be concerned with whom or what will handle the *withdrawal request* inside the WAREHOUSE; however, these things will be modelled as objects when the WAREHOUSE object is decomposed to reveal the underlying mechanism of how the WAREHOUSE object provides its services. Such an abstracted perspective is used throughout the modelling process. The request propagation continues from the OFFICE CLERK object to the warehouse object; from the WAREHOUSE object to the PLANNER object; from the PLANNER object to the TRUCK DRIVER object; and so forth (Rule 2: Object Identification Rule - internal objects).

Modelling rules 1 to 5 are used throughout the request propagation process. Attributes and services for each object are identified as one proceeds through the request propagation. A request is recorded as an attribute (i.e. interface attribute) in the object that receives the request, and the corresponding service that processes that attribute is correspondingly identified. For example, when the CUSTOMER object in Figure 7 sends a *withdrawal request* to the OFFICE CLERK object, the request is recorded as an attribute in OFFICE CLERK (Rule 4: Attribute Inclusion Rule - interface). The state change of the *withdrawal request* interface attribute invokes the corresponding *process withdrawal* service in OFFICE CLERK (Rule 3: Service Inclusion Rule). The internal working of the *process withdrawal* service in turn uses the internal attribute, *customer information*, during its execution (Rule 4: Attribute Inclusion Rule - internal). The process then continues the request propagation by sending a *withdrawal request* to the WAREHOUSE object. Rule 6 and 7 are satisfied in the current enterprise model, and no additional changes need to be made in terms of aggregation and specialization.

5.1.1 Object Decomposition

Decomposing an object into its components allows one to probe further into the operational details of an object. To gain a more in-depth understanding of the warehouses' operations, it would be necessary to decompose the warehouse object into its components for analysis. If the analyst deems it necessary, the other internal ACME objects may also be decomposed; but for the purposes of this case, only the warehouse object needs to be decomposed. The same request propagation process can be used to find the component objects. Relative to the warehouse object, customer, office clerk, planner, and truck driver are viewed as external objects. A full analysis of the request propagation will take into account all requests sent by these objects to the warehouse object. A further probe into the operations of the warehouse object reveals the following details:

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

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

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

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

The decomposition of the WAREHOUSE object results in the diagram depicted in Figure 8. The WAREHOUSE object decomposes into three objects: WAREHOUSE WORKER, WAREHOUSE MANAGER, and FORKLIFT OPERATOR. The attributes and operations of the warehouse object are distributed among the three component objects. These attributes and operations may be divided among the component objects as either whole units, or alternatively, any single attribute or operation of the warehouse object may themselves be decomposed and divided among the components. (See Figure 5 - basic patterns of decomposition). The distribution is determined in the request propagation process according to the details of the case description.

We see that the *withdrawal request* from the OFFICE CLERK object is processed by the WAREHOUSE MANAGER component object. Thus, the *withdrawal request* attribute *and process withdrawal request* operation of the WAREHOUSE object is assigned to the WAREHOUSE MANAGER component object. As we follow the path of the request propagation, we also see the decomposition of the attributes and operations of the WAREHOUSE object. For example, the TRUCK DRIVER object's request to either *prepare loading* or *prepare unloading*, is processed by two objects: the WAREHOUSE MANAGER and the WAREHOUSE WORKER. *The prepare loading* request and *prepare unloading* request attributes, and the *prepare loading* and *prepare unloading* operations of the warehouse object are divided between these two component objects. The WAREHOUSE's *prepare loading requests* attribute and *prepare loading* operation becomes decomposed into the WAREHOUSE MANAGER's *prepare loading request* and *prepare loading* operation; and into the WAREHOUSE WORKER's *prepare loading schedule* attribute and *schedule loading* operations. These two component objects work together in conjunction to service the *prepare loading* request by sending requests to one another. The same breakdown is used to service the *prepare unloading* request.

The *item existence query* request that originates from the WAREHOUSE MANAGER object is purposely shown to exit and re-enter the WAREHOUSE composite object. This depiction is used to semantically represent the fact that a WAREHOUSE MANAGER sends the request to another WAREHOUSE MANAGER of a separate WAREHOUSE.

The TRUCK DRIVER object's request to *start loading* and *start unloading* are processed by three objects: WAREHOUSE WORKER, FORKLIFT OPERATOR and WAREHOUSE MANAGER. The WAREHOUSE composite object's *load items request* and *unload items request* attributes; and *start loading* and *start unloading* services are operationalized through the services and interactions between the WAREHOUSE WORKER and FORKLIFT OPERATOR component objects. Once the inventory items are either loaded or unloaded, the WAREHOUSE WORKER requests that the WAREHOUSE MANAGER update the status of the inventory.

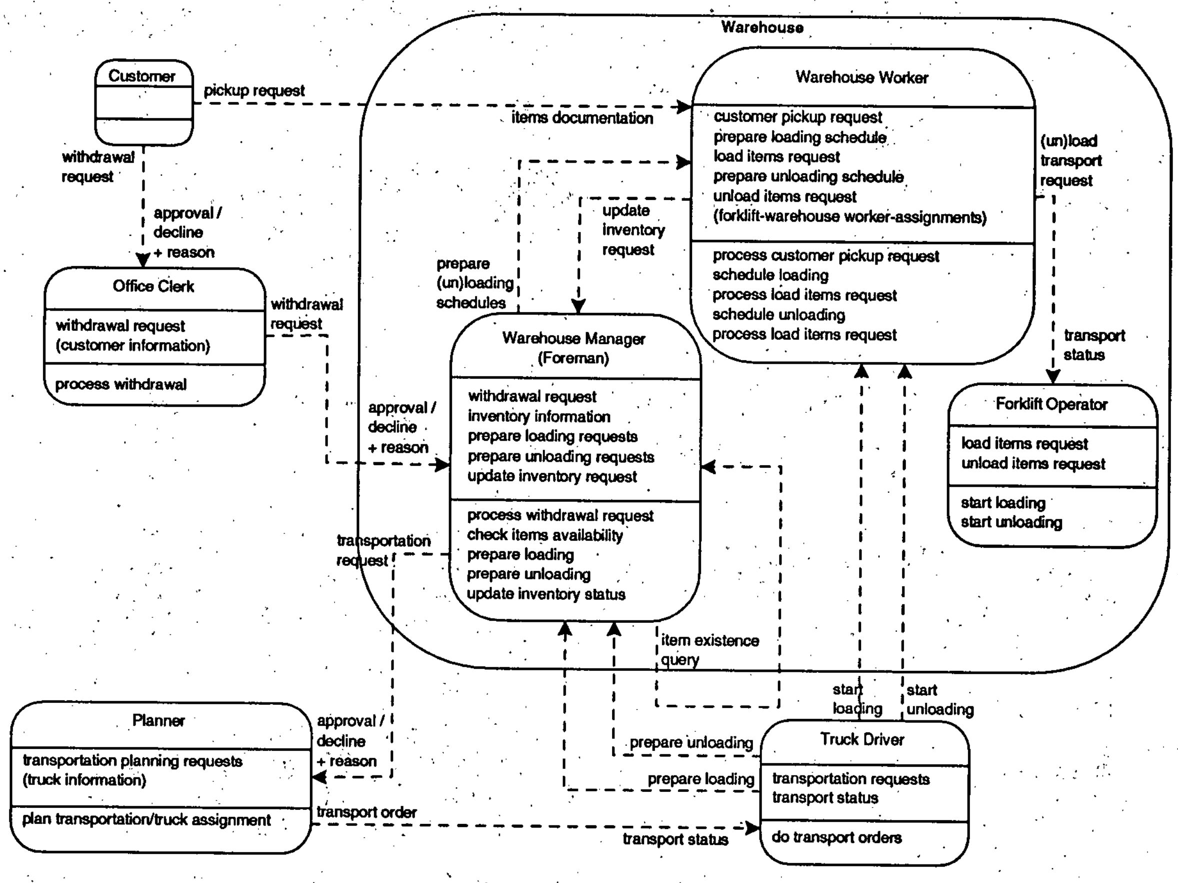

Figure 8 Decomposition of the Warehouse object

5.1.2 Static Objects

Static objects are used to depict objects for which we wish to track state or to keep knowledge about. The states of such objects can only be altered by a dynamic object since static objects have no dynamic behaviour. Static objects only have interface attributes. These attributes may either be variable or constant since dynamic objects may either manipulate or read a static object's state.

Up to this point, the enterprise model has been developed with only dynamic objects. Static objects are added to the model to explicitly show the objects which we wish to track. Static objects are discovered by proceeding with the request propagation, but exclude any references to services. Figure 9 depicts the additions of three static objects to the model: INVENTORY, FORKLIFT, and TRUCK VEHICLE.

The state of INVENTORY changes every time the FORKLIFT OPERATOR either loads or unloads items in the warehouse. This scenario is represent as a dotted arrow from the FORKLIFT OPERATOR to INVENTORY. The FORKLIFT OPERATOR reports the status of the items movement through the *transport status* response to the WAREHOUSE WORKER, who then sends an *update inventory request* to the WAREHOUSE MANAGER. This partial request propagation trace provides the mechanism for how the knowledge of INVENTORY state is updated by the WAREHOUSE MANAGER.

With reference to the TRUCK VEHICLE static object, the model shows that the TRUCK DRIVER is required to read the maximum capacity and working order of the TRUCK VEHICLE, and relay the information to the PLANNER. The dotted arrow from the TRUCK DRIVER to the TRUCK VEHICLE represents the fact that the TRUCK DRIVER object either changes or reads the attributes of the TRUCK VEHICLE object. A similar meaning is also applied in the interpretation of the FORKLIFT static object. The states of the TRUCK VEHICLE object are tracked by the PLANNER object; and the states of the FORKLIFT object are tracked by the WAREHOUSE WORKER object. The state tracking attributes of the PLANNER and WAREHOUSE WORKER objects represent knowledge that can be utilized in future database designs. In addition, the events which change the state of the static objects help identify the necessary triggers for correspondingly updating the state of the database. Although the TRUCK and FORKLIFT vehicles can be viewed as dynamic objects that react to stimulus from the drivers, these interactions are not of interest for the purposes of this model. The model is only concerned with the specific changes of state (knowledge) of these objects, and are thus modelled as static objects.

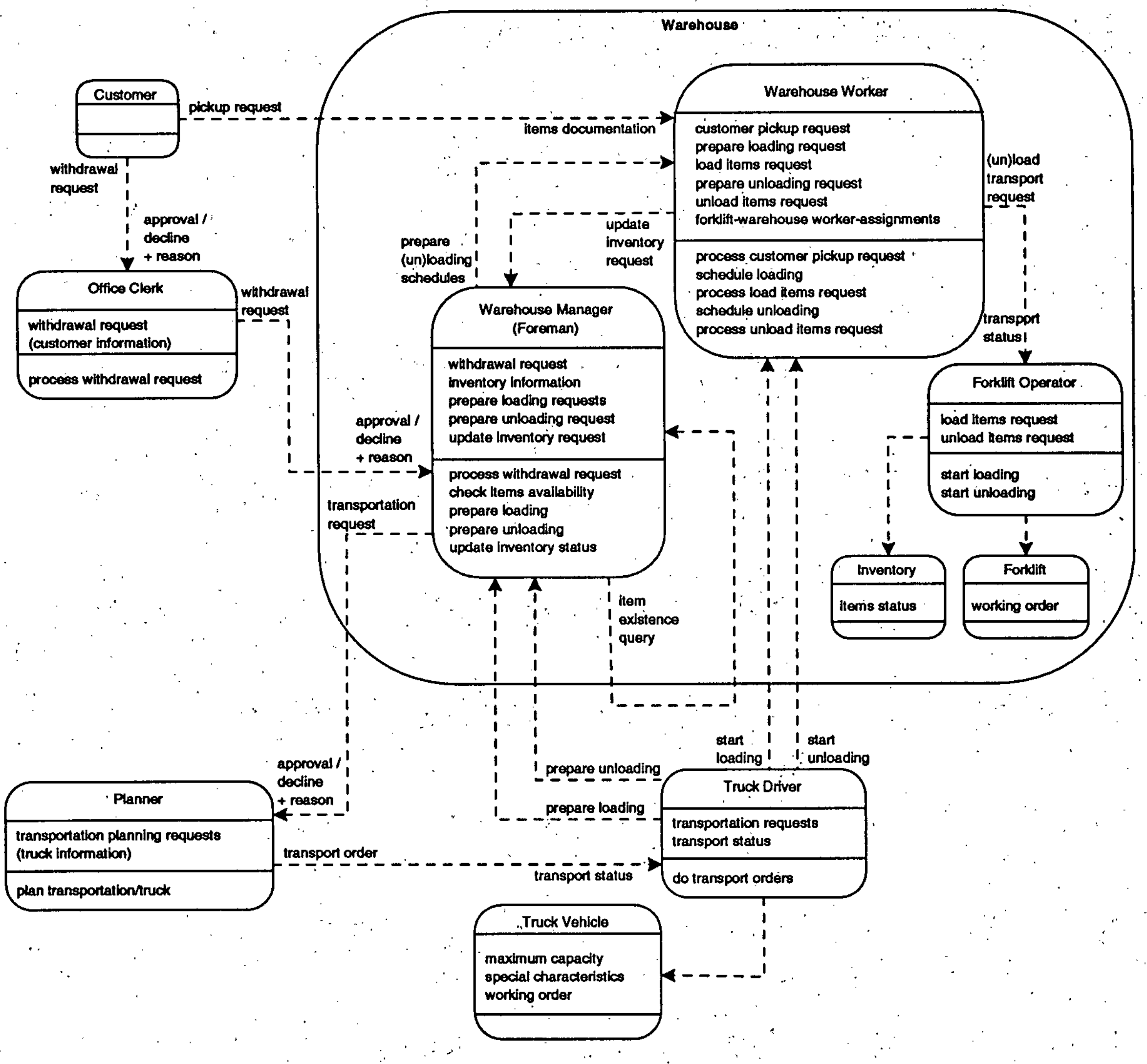

Figure 9 Enterprise model with static objects

5.1.3 Requirements Analysis and Delegation Objects

At this point, the enterprise model depicts both the organizational and information system aspects embedded together. The organizational and formal IS components can be separated by using the object delegation rule. These delegation objects correspond to objects which track the state of other objects. These IS objects form the initial requirements analysis for the information system that is to be designed. The incoming requests into the IS objects casts the requirements specifications that are expected from a client object. Such IS objects may represent automated, manual-driven or hybrid (combination of automation and manual) information systems. Figure 10 depicts the enterprise model with the addition of these delegation IS objects. Attributes and their corresponding services from the main object are delegated out to an IS object. These attributes may include the representation of databases that are used within the organization. Four delegation objects have been identified in the enterprise model: the OFFICE CLERK object delegates out the CUSTOMER SYSTEM object; WAREHOUSE MANAGER object delegates out the INVENTORY SYSTEM object; the WAREHOUSE WORKER object delegates out the FORKLIFT SCHEDULE SYSTEM object; and the PLANNER object delegates out the TRUCK SCHEDULE SYSTEM object.

The delegated IS objects must conform to the delegation rule which simply states that a delegation of each attribute must be tied with at least one corresponding service, and vice versa. In addition to delegating whole attributes and services, the delegated object may also include decomposed attributes and services. In Figure 10 the OFFICE CLERK delegates out the *customer information* attribute to the CUSTOMER SYSTEM object. In addition, the OFFICE CLERK sends an *item withdrawal request* to the delegated object. Thus, the CUSTOMER object has an *authenticate customer* service to process the *item withdrawal request*. This service uses the *customer information* attribute to carry out its operation. The original *authenticate customer* service has been decomposed from and taken out of the original *process withdrawal request* of the OFFICE CLERK, and has been delegated to the CUSTOMER SYSTEM object.

The WAREHOUSE MANAGER delegates the *inventory information* attribute, and *update inventory request* attribute to the INVENTORY SYSTEM object. Whenever the WAREHOUSE MANAGER needs to process a *withdrawal request*, perform an *items existence query*, or *update the inventory status*, requests will be sent to the INVENTORY SYSTEM.

With the PLANNER delegation to the TRUCK SCHEDULE SYSTEM, we see that the *truck information* attribute in the PLANNER becomes decomposed and delegated to the TRUCK SCHEDULE SYSTEM as *truck information* and *truck schedules* attributes. The PLANNER object's execution of its *plan transportation/truck assignments* service will require it to use the TRUCK SCHEDULE SYSTEM delegation object.

After the delegation of responsibilities from the WAREHOUSE WORKER to the FORKLIFT SCHEDULE system, the WAREHOUSE WORKER's execution of either the *process load items request* or *process unload items request* will result in the following sequence of events:

1. WAREHOUSE WORKER looks up the *schedule of forklift assignments* from the FORKLIFT SCHEDULE SYSTEM
2. WAREHOUSE WORKER directs the scheduled FORKLIFT OPERATORs to their roles
3. FORKLIFT OPERATORs sends back its *status* after completing the *(un)load request*
4. WAREHOUSE WORKER updates the *schedule information* in the FORKLIFT SCHEDULE SYSTEM
5. WAREHOUSE WORKER requests that the WAREHOUSE MANAGER change the status of the *inventory* records

The *scheduling of loading* and *unloading* requests will require the WAREHOUSE WORKER to work with the FORKLIFT SCHEDULE SYSTEM.

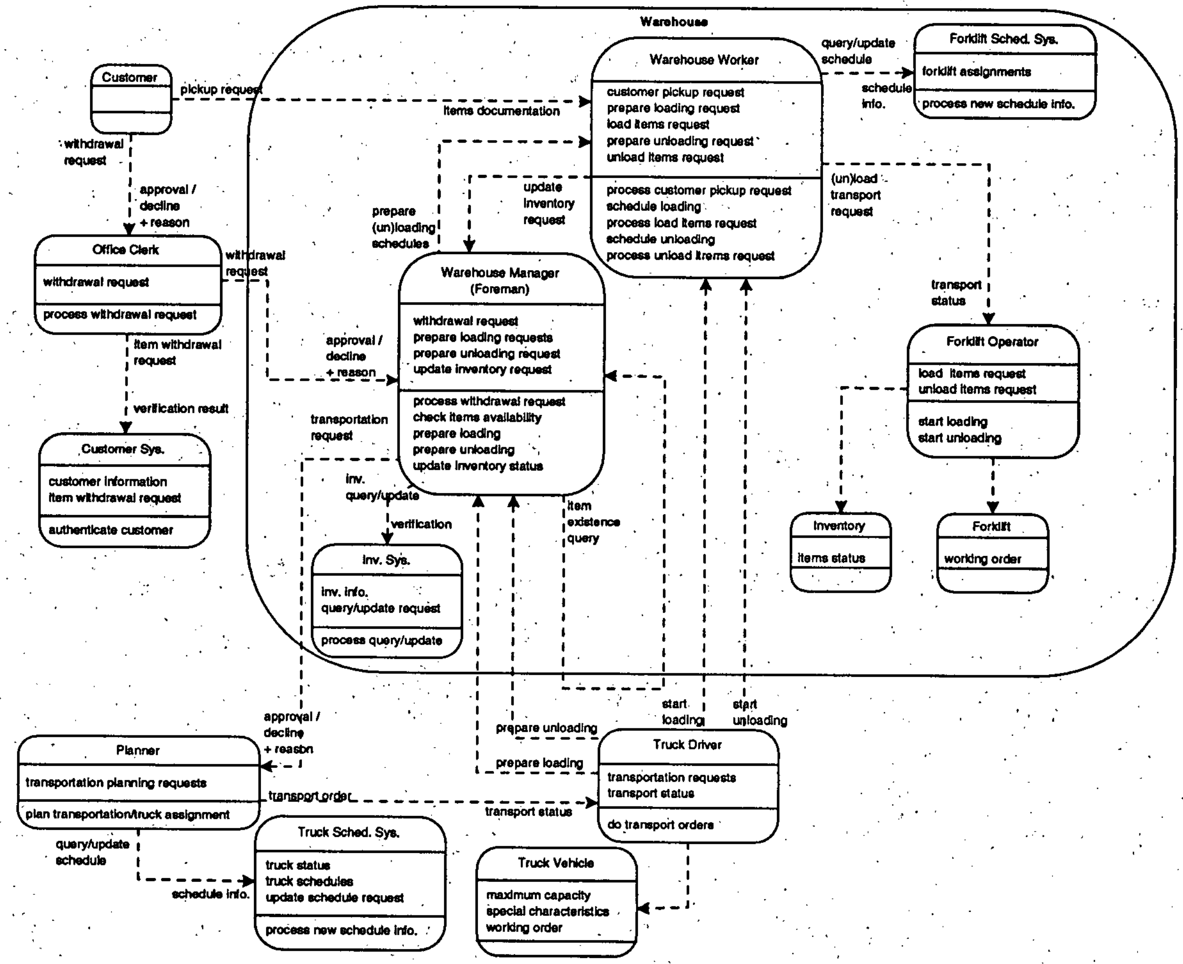

Figure 10 Enterprise model with delegation objects

5.2 Object-Oriented Design

Design involves the layout of a plan for constructing the specified components of ACME warehouse. These components may include both organizational and information systems (IS) aspects. For the purposes of illustrating the design principles, only the IS segment of ACME will be modelled. The objective of this example is to eventually reach a point where an information system can be designed. The identified systems will be designed with the intention of automation as opposed to a redesign of the organization with information technology. Appendix III shows an example of a possible representation of an organizational redesign, but will fall short of in-depth detail. Organizational design may involve complex negotiations for organizational change and will need systems to support change management. These matters are beyond the scope of this work.

One of the first IS design decisions is to decide on the level of inter-communication dependency among the IS objects. It is assumed that ACME requires a high-level of inter-communication among its IS objects. Such a design would alter the communication pathways of the message requests between objects since communication among objects would be delegated to the delegation objects whenever it is possible. Figure 11 depicts the new design for the proposed enterprise model. The model shows how the information system will change the propagation of events within the business. The solid arrows are used to distinguish message requests between the information system objects and the rest of the enterprise objects.

The following list enumerates the communication that has been changed or delegated to the IS objects:

1. *withdrawal request*: OFFICE CLERK to WAREHOUSE MANAGER => CUSTOMER SYSTEM to INVENTORY SYSTEM
2. *item existence query*: WAREHOUSE MANAGER to WAREHOUSE MANAGER => INVENTORY SYSTEM to INVENTORY SYSTEM
3. *transport request*: WAREHOUSE MANAGER to PLANNER => INVENTORY SYSTEM to TRUCK SCHEDULE SYSTEM
4. *prepare (un)loading schedule*: WAREHOUSE MANAGER to WAREHOUSE WORKER => WAREHOUSE MANAGER to FORKLIFT SCHEDULE SYSTEM
5. *update inventory request*: WAREHOUSE WORKER to WAREHOUSE MANAGER => FORKLIFT SYSTEM to INVENTORY SYSTEM

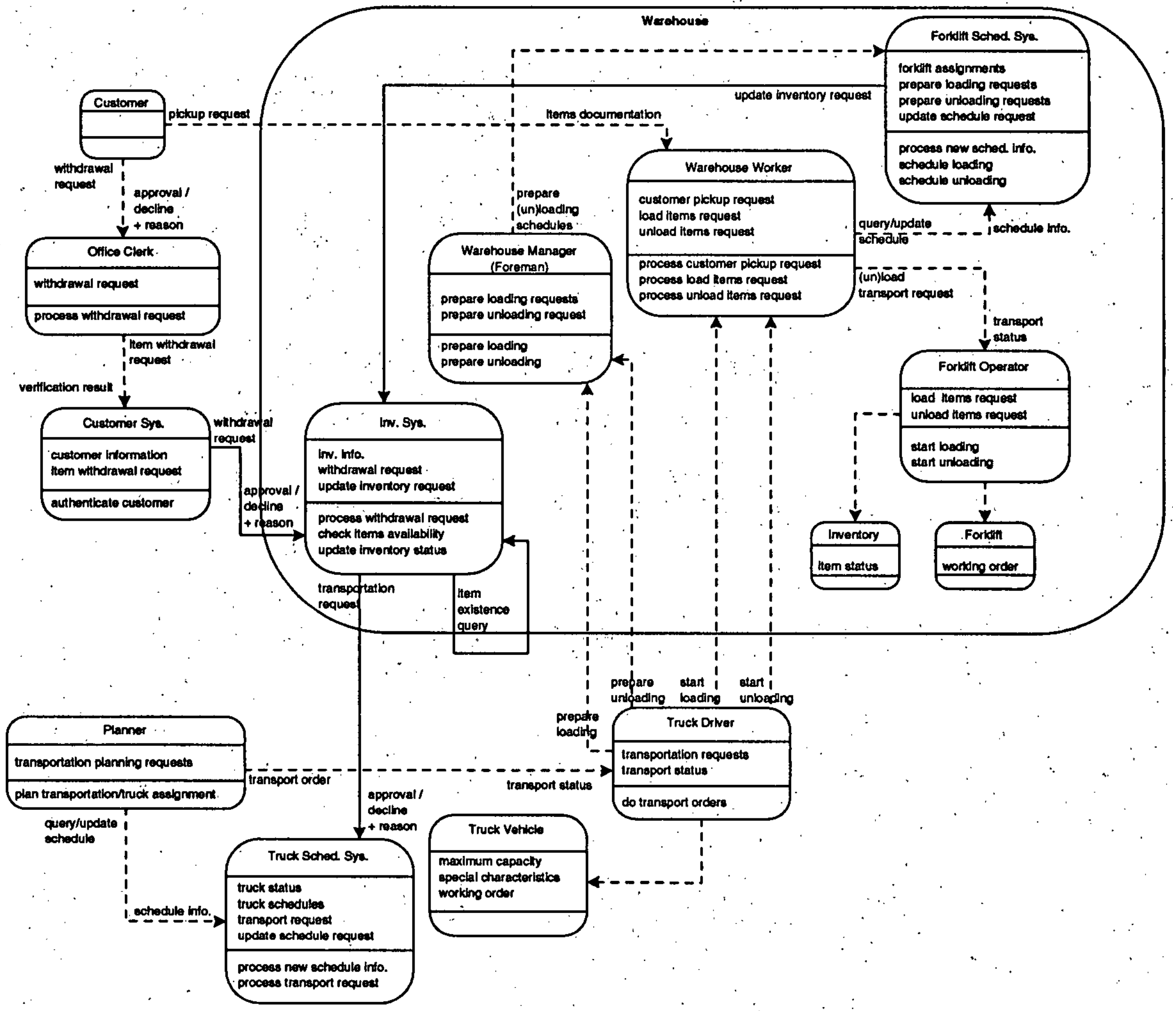

Figure 11 Proposed enterprise model with delegated object communication paths

The four identified IS objects in the enterprise model can be thought of as subsystem components of the ACME Warehouse information system. Further decomposition of each IS subsystem through the request propagation process will reveal the component objects of these four IS subsystems. For the purposes of this case, the focus of the design is based on the customer withdrawal process. A complete design would take into account all the possible events going into the subsystems.

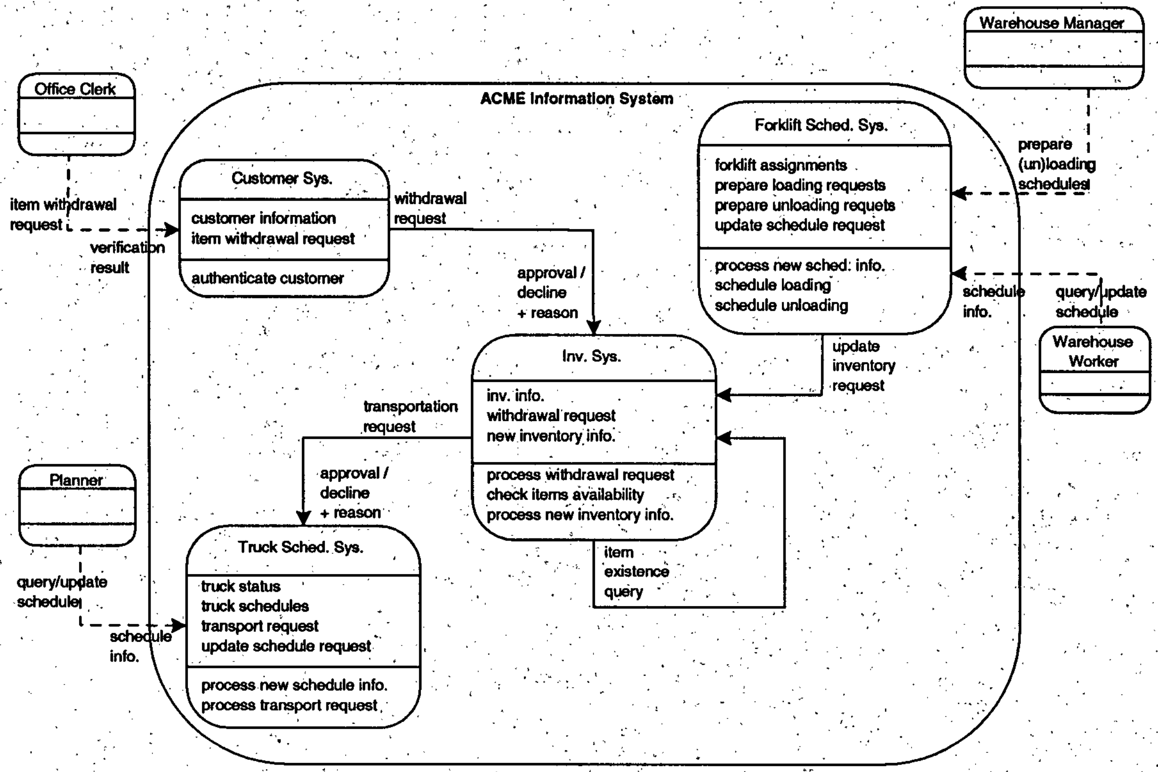

Figure 12 The proposed ACME information system

Figure 12 illustrates the interaction of the four IS objects (subsystems) that have been identified for the ACME information system. The diagram depicts a first level decomposition for the design of an automated ACME information system. The system essentially provides four types of services. It tracks the state of inventory; schedules interwarehouse transportation; schedules internal warehouse logistics; and organizes the company's information on its customers. These four roles are provided through the four identified IS objects or subsystems as depicted in Figure 10.

6. Evaluation of OOEM-Design

This chapter examines what we have learned in the process of discovering design principles. We will look at the general similarities and differences with other methods by comparing OOEM with design rules (OOEM-Design) to the design approaches from Coad and Yourdon (OOA/OOD), Jacobson (Use-Case), and Coleman (Fusion) which were summarized in chapter 2.

The objective is to examine the extent of how each of these methods deals with the proposed modelling constructs, and to explore what each method proposes for the transition from an analysis to a design model. The criteria which we will examine will concern the following areas of OOEM-Design: decomposition, delegation, and static objects.

6.1 Comparison with Coad and Yourdon OOA/OOD

Coad and Yourdon define design as "the practice of taking specifications of externally observable behavior and adding details needed for actual computer system implementation, including human interaction, task management, and data management" (Coad and Yourdon 1991, p. 5). The emphasis is on constructing a map or plan for software construction. The following criteria, from Coad and Yourdon, for altering the problem domain component illustrates the implementation focus:

- reuse design and programming classes
- group problem-domain-specific classes together
- establish a protocol by adding a generalization class
- accommodate the supported level of inheritance
- improve performance

- support the data management component
- add lower-level components

Coad and Yourdon's OOD also includes the development of the user interface, design of the multitasking code, and design of the database.

OOEM-Design's objective is to bridge the enterprise model with the software model. The emphasis is not on software implementation.

Coad and Yourdon provide constructs for depicting whole-part relationships. This corresponds to the decomposition concept of OOEM-Design. However, Coad and Yourdon does not approach design through decomposition since it's focus is on implementation. The analysis model they produce is in fact a requirements analysis for the software system. The requirements analysis model is the output of OOEM-Design.

Coad and Yourdon do not provide any constructs for static objects and have no concept of a delegate object since objects in their design model only reflect the software domain. The objects that they provide in their examples are implementation objects and not organizational objects. Instead of representing an organization, their objects represent software entities. In OOEM-Design, the objects represent organizational entities. Software objects don't come into play until an organizational entity delegates their responsibilities to an automated information system.

6.2 Comparison with Jacobson's Use Case Method

Jacobson's use case approach is similar to OOEM's request propagation in its approach to finding out what services the business organization needs to provide to its clients. Jacobson's client or actor construct corresponds to OOEM's external object construct. These send requests/stimuli to the organizational system, and the organization operates to satisfy these requests/stimuli. Jacobson's approach toward software design uses the same use case method. Actors in the business require services from an information system and thus send stimuli to the computer system. The need for an information system is determined in the business modelling phase; however, there is no method to direct one to discover what kind of automated system would be needed. Unlike Jacobson, OOEM-Design does not assume that a particular information system has been decided upon. OOEM-Design's approach provides a method to discover what type of information system the organization requires. IS objects result from a decision to use delegate objects to delegate information processing tasks. These delegates represent components for an information system.

The concept of decomposition is captured in Jacobson's use of the subsystems concept. A subsystem is defined as "a package that contains functionally allied objects and/or subsystems. An object or subsystem cannot belong to more than one subsystem" (Jacobson et al. 1995, p. 142). However, decomposition is not used in the same manner as in OOEM-Design. There are no principles to guide one in finding subsystems. Jacobson uses subsystems to reduce the need to model everything at once; or to make the model more understandable and easier to present; or to package objects into units of responsibility suitable for one person who is responsible for one or more tasks.

Jacobson does not provide further details for design since it goes into the realm of software engineering. Static objects and delegates are concepts that do not exist in Jacobson's method.

6.3 Comparison with the Fusion Method

Fusion offers an approach to object-oriented software development. Fusion does not model the organization. It assumes the need for a specific information system. Analysis begins from the point at which a requirements analysis has been completed. Fusion uses the requirements analysis to develop an analysis model. Design is used to depicts how operations are implemented, and has no correspondence to OOEM-Design. Design takes the results from analysis and defines how functionality is implemented. Fusion does not provide the concept of decomposition, static object, nor for the role of a delegate. In addition, Fusion speaks of subsystems, yet there is no notation for depicting subsystems.

6.4 Summary

The chapter provided an anecdotal comparison of OOEM-Design with the design approaches of three other object-oriented modelling methods. The comparison examined the modelling domain, method of transition (from analysis to design), decomposition, static objects, and delegate objects. The major findings are as follows:

- design is mostly approached from the viewpoint of software development
- decomposition is not used as a primary approach to design
- the static object and delegate object concepts are not included in the examined methods
- there exists no primary set of rules or principles for approaching design. Heuristics, experience and intuition are the main factors for making and revealing discoveries during the creation of the model.

The following table summarizes the findings for the examined modelling approaches to design in comparison to OOEM-Design. An (x) signifies that the modelling approach covers the characteristic and a (-) signifies that the approach excludes the use of the concept:

	OOA/OOD (Coad&Yourdon)	**Use Case** (Jacobson et al.)	**Fusion** (Coleman et al.)	**OOEM-Design**
Decomposition	x	x	-	x
Static Objects	-	-	-	x
Delegate Objects	-	-	-	x
Modelling Rules	-	-	-	x
Organizational Viewpoint	-	x	-	x

Table 2 Summary of Methods Comparison

The major contribution of OOEM-Design is its provision of a systematic approach to reason about design decisions for the transition from an enterprise model to the initial logical software model. Other methods rely on heuristics and implementation criteria for developing a design model. The other methods make no provision for reasoning about a transition since they focus on building an implementation model.

7. Conclusion and Future Research

This thesis continues the work in the development of Object-Oriented Enterprise Modelling (OOEM) by providing extensions to the analysis modelling rules to encompass design concepts. The design principles work to provide a bridge for the transitional work from an enterprise analysis model to an initial logical information system model. The key aspect of the work is to provide a structured method for reasoning about design when we examine an analysis model of an enterprise.

Design was approached by continuing with the ontologically derived object-oriented analysis modelling constructs. By expanding on the concepts and use of request propagation and decomposition, we further refine the analysis model by discovering the inner workings of an object. The specified object becomes the scope of analysis for which we discover its components, the components' attributes, the components' services, and interactions. Delegate objects are introduced to provide the transition that moves the modelling arena from the business enterprise to the realm of information systems. The attributes of delegate objects provide the linkage to the data for a database design. Static objects are used to depict static entities in an enterprise during a decomposition, and they provide the events that should lead to a triggering of state changes within a database.

7.1 Contribution

OOEM-Design extends the original OOEM modelling rules through the addition of three new rules to the original seven. The rules cover (1) object decomposition, (2) delegation and (3) static objects.

In correspondence with introduction of the new rules, further elaboration was provided for the concept of aggregation and static objects. The concepts of decomposition and delegation were defined. The work illustrated four basic types of decomposition patterns and four basic types of delegation patterns that complement the decomposition and delegation rules. These patterns are mixed and matched in practice to form more complex patterns of decomposition and delegation. A case example was provided to illustrate how one would perform the migration from analysis activities to design.

The primary benefit of the new modelling rule extensions is the provision of a method for a transition between analysis and design models. Much of the current literature on formal models deal with analysis and design from the software point of view. Methods which model organizations do not provide rules for the transition to design. By extending the original OOEM rules, the design rules provide this transition point from modelling organization to modelling information systems. Furthermore, the use of delegates and static objects provide us hints to composition of a systems database through the attributes.

7.2 Limitations and Future Research

This work provides areas for continued future work. The proposed method of reasoning about design lacks a formalized approach for model representation. Further work into applying the method to business process reengineering (BPR) and detailed design is also warranted.

There is a need to cement OOEM-Design in a modelling language. As it stand, OOEM-Design only presents principles for which one can reason about design independently from any specific modelling representation (i.e. language). For effective communication, one may wish to develop a standard set of symbols to represent the modelling constructs.

The main focus of this work was to show the transition from analysis to a preliminary logical information systems or software design. Another use for this type of modelling is in the area BPR (Jacobson et al. 1992, Wang 1984). In Appendix III, an example is given for a possible BPR model. Further developments into BPR modelling are possible from this work.

Furthermore, the work can be taken towards the development of detail or software design models. One can either examine current object-oriented software design approaches and merge it with OOEM-design, or provide a completely new approach to software design by beginning with OOEM-Design principles. More recently, object-oriented analysis and design modelling for software development has entered into a search for a pattern language for objects. This idea was influenced by the work of Christopher Alexander, an architect, who wrote Notes on the Synthesis of Form (1964), The Timeless Way of Building (1979), and A Pattern Language (1977). Coad's (1995) new method for OOA/OOD is based on patterns. Further study into OOEM and OOEM-Design may provide a more solid grounding for any type of pattern language based on an ontological foundation.

References

Alexander, C. (1964). *Notes on the Synthesis of Form.* Cambridge: Harvard University Press.

Alexander, C., Ishikawa, S., Silverstein, M., Jacobson, M., Fiksdahl-King, I., and Angel, S. (1977). *A Pattern Language: Towns, Buildings, Construction.* New York: Oxford University Press.

Alexander, C. (1979). *The Timeless Way of Building.* New York: Oxford University Press.

Booch, G. (1993). *Objects-Oriented Analysis and Design with Applications* (2nd ed.). Redwood City, CA: Benjamin/Cummings.

Bunge, M. (1977). *Ontology I: The Furniture of the World Vol. 3 of the Treatise on Basic Philosophy.* Dordrecht: Reidel.

Bunge, M. (1979). *Ontology II: A World of Systems Vol. 4 of the Treatise on Basic Philosophy.* Dordrecht: Reidel.

Coad, Peter, and Edward Yourdon. (1991a). *Object-Oriented Analysis* (2nd ed.). Englewood Cliffs: Prentice Hall.

Coad, Peter, and Edward Yourdon. (1991b). *Object-Oriented Design.* Englewood Cliffs: Prentice Hall.

Coad, Peter, North, D., and Mayfield, M. (1995). *Object Models: Strategies, Patterns and Applications.* Englewood Cliffs: Prentice Hall.

Coleman, D., Arnold, P., Bodoff, S., Dollin, C., Gilchrist, H., Hayes, F., Jeremaes, P. (1994). *Object-Oriented Development: The Fusion Method.* Englewood Cliffs: Prentice Hall.

Curtis, B., Kellner, M.I., and Over J. (1992). Process Modelling. *Communications of the ACM.* Vol. 35 (9), pp. 75-90.

DeMarco, T. (1979). *Structured Analysis and System Specification.* Englewood Cliffs, NJ: Prentice Hall.

Embley, D.W., Jackson, R.B. and Woodfield S.N. (1995). OO Systems Analysis: Is It or Isn't It?. *IEEE Software*, Vol 12 (4), 19-32.

Gane, C., & Sarson, T. (1986). *Structured Systems Analysis, Tools and Techniques*. Englewood Cliffs, NJ: Prentice Hall.

Kim, W. & Lochovsky, F. (Eds.). (1989). *Object-Oriented Concepts, Applications and Databases*. Reading, Mass: Addison-Wesley.

Morris, William (Ed.). (1980). *The Houghton Mifflin Canadian Dictionary of the English Language*. Markham, Ontario: Houghton Mifflin Canada.

Jacobson, I., Christerson, M., Jonsson, P., and Övergaard, G. (1992). *Object-Oriented Software Engineering: A Use Case Driven Approach*. Don Mills, Ontario: Addison-Wesley.

Parsons, J. & 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., Eddy, F., and Lorenson, W. (1991). *Object-Oriented Modelling and Design*. Englewood Cliffs: Prentice-Hall.

Shaw, M. (1984). Abstraction Techniques in Modern Programming Languages. *IEEE Software*. Vol. 1 (4), p. 10.

Simon, Herbert. (1981). *The Sciences of the Artificial* (2nd ed.). Cambridge: The MIT Press.

Synder, Alan. (1993). The Essence of Objects: Concepts and Terms. *IEEE Software*, January 1993, pp. 31-42.

Wand, Y. (1988). An Ontological Foundation for Information Systems Design Theory. *Proceedings of the IFIP 8.4 Working Conference on Office Information Systems: The Design Process*. Linz, Austria. Amsterdam: Elsevier Science Publishers B.V. pp. 201-222.

Wand, Y. (1989). A Proposal for a Formal Model of Objects. *Object-Oriented Concepts, Applications, and Databases*. W. Kim & F. Lochovsky (eds.). Reading, Mass: Addison-Wesley, 537-559.

Wand, Y. & Woo, C. (1993). Object Oriented Analysis: Is it Really that Simple. *Proceedings of the Workshop on Information Technology and Systems*, Orlando, Florida, pp. 186-195.

Wand, Y. & Woo, C. (1996). *Rules for Object-Oriented Enterprise Modelling*. Tutorial Notes. University of British Columbia.

Wand, Y. & Weber, R. (1989, December). A Model of Systems Decomposition. *Proceedings of the Tenth International Conference on Information Systems*. Boston, Mass., pp. 41-51.

Wand, Y. & Weber, R. (1990). An Ontological Model of an Information System. *IEEE Transactions on Software Engineering*. Vol. 16 (11), pp. 1282-1292.

Wand, Y. & Weber, R. (1991). A Unified Model of Software and Data Decomposition. *Proceedings of the Twelth International Conference on Information Systems*, New York, pp. 101-110.

Wand, Y. & Weber, R. (1995). On The Deep Structure of Information Systems. *Journal of Information Systems*. 5, pp. 203-223.

Wang, S. (1984). OO Modeling of Business Process: Object-Oriented Systems Analysis. *Information Systems Analysis*. Spring 1984.

Yourdon, E. (1989). *Modern Structured Analysis*. Englewood Cliffs, NJ: Yourdon Press.

Yourdon, E. (1992). *Decline & Fall of the American Programmer*. Englewood Cliffs: Yourdon Press/Prentice Hall.

Zhao, H. (1995). *Object-Oriented Enterprise Modelling*. M.Sc. Diss. The University of British Columbia.

Appendix I Bunge's Ontological Constructs

The summary of ontological construct definitions in this appendix are was taken from Zhao (1995).

The following sections are mainly an abstraction of Bunge's work (1977, 1979). Much of the wording has been taken directly as quoted from these books. The section regarding to the dynamic model of system is from Wand and Weber (1990).

Static model of a 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 a 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 a 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 an emergent property. The notion of emergent property is an important assumption in Bunge's ontology. To him, every concrete system is assembled from, or with the help of, things in the same or lower order genera and must possess 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 a 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 a 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 a 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 Object-Oriented Modelling in Template Form

The following are examples of the internal object templates and external object templates that Zhao (1994) uses to represent an object model. These templates provide samples based on the ACME warehouse case. Figure 7 (Initial enterprise model of ACME) and Figure 8 (Decomposition of the warehouse object) are presented as an example. (Note: Under the area labelled access mode, the letters M (modify) and U (use) are used to designate the mode of operation on the state of the internal attribute.)

Object Name						
Requesting Objects	**Services**	**Interface Attributes**	**Internal Attributes**		**Request Generated**	**Receiving Objects**
Object 1	service 1	incoming interface attributes	Internal Attributes to support service 1	Access Mode	Request generated from service 1	Object 3
		returning interface attributes			Request generated from service 1	Object 4
Object 2	Service 2	incoming interface attributes	Internal Attributes to support service 2	Access Mode	Request generated from service 2	Object 5

Table 3 Internal Object Template (IOT)

Object Name			
Requesting Objects	**Requests to system**		**Receiving Objects**
	request 1		Internal Object 1
	request 2		Internal Object 2
	Services	**Interface Attributes**	
Internal Object 2	Service 1	Interface attributes of Service 1	
Internal Object 3	Service 2	Interface attributes of Service 2	

Table 4 External Object Template (EOT)

Initial Enterprise Model of ACME

Customer			
Requesting Objects	**Requests to system**		**Receiving Objects**
	Withdrawal request		Office Clerk
	Pickup request		Warehouse
	Services	**Interface Attributes**	

Table 5 EOT for the Customer Object

Office Clerk						
Requesting Objects	**Services**	**Interface Attributes**	**Internal Attributes**		**Generated Requests**	**Receiving Objects**
Customer	Process withdrawal request	Withdrawal request	Customer information	U	Withdrawal request	Warehouse
		Approval/ decline + reason				

Table 6 IOT for the Office Clerk Object

Warehouse						
Requesting Objects	**Services**	**Interface Attributes**	**Internal Attributes**		**Generated Requests**	**Receiving Objects**
Office Clerk	Process withdrawal request	Withdrawal request			Transport request	Planner
		Approval/decline + reason				
Warehouse	Check items availability	Inventory information			Item existence query	Warehouse
Truck Driver	Prepare loading	Prepare loading request	Warehouse worker assignments	M		
Truck Driver	Start loading	Load items	Warehouse worker assignments	U		
Truck Driver	Prepare unloading	Prepare unloading request	Warehouse worker assignments	M		
Truck Driver	Start unloading	Unload items	Warehouse worker assignments	U		
Customer	Process customer pickup request	Customer pickup request				
		Items documentation				

Table 7 IOT for the Warehouse Object

Planner						
Requesting Objects	**Services**	**Interface Attributes**	**Internal Attributes**		**Generated Requests**	**Receiving Objects**
Warehouse	Plan transportation/ truck assignment	Transportation planning request	Truck information	U	Transport order	Truck Driver
		Approval/ decline + reason				

Table 8 IOT for the Planner Object

Truck Driver						
Requesting Objects	**Services**	**Interface Attributes**	**Internal Attributes**		**Generated Requests**	**Receiving Objects**
Planner	Do transport orders	Transportation request			Prepare loading	Warehouse
		Transport status			Start loading	
					Prepare unloading	
					Start unloading	

Table 9 IOT for the Truck Driver Object

Decomposition of the Warehouse Object

Warehouse Manager						
Requesting Objects	**Services**	**Interface Attributes**	**Internal Attributes**		**Generated Requests**	**Receiving Objects**
Office clerk	Process withdrawal request	Withdrawal request			Transportation request	Planner
		Approval/ decline + reason				
Warehouse manager	Item existence query	Inventory information			Item existence query	Warehouse manager
Truck Driver	Prepare loading	Prepare loading request			Prepare loading schedules	Warehouse worker
Truck Driver	Prepare unloading	Prepare unloading request			Prepare unloading schedules	Warehouse worker
Warehouse Worker	Update inventory status	Update inventory request				

Table 10 IOT of the Warehouse Manager Object

Warehouse Worker						
Requesting Objects	**Services**	**Interface Attributes**	**Internal Attributes**		**Generated Requests**	**Receiving Objects**
Warehouse manager	Schedule loading	Prepare loading schedules	Forklift assignments	M		
Warehouse manager	Schedule unloading	Prepare unloading schedules	Forklift assignments	M		
Truck Driver	Process load items	Load items request	Forklift assignments	U M	Transport request	Forklift operator
Truck Driver	Process unload items	Unload items request	Forklift assignments	U M	Transport request	Forklift operator

Table 11 IOT of the Warehouse Worker Object

Forklift Operator						
Requesting Objects	**Services**	**Interface Attributes**	**Internal Attributes**		**Generated Requests**	**Receiving Objects**
Warehouse worker	Start loading	Load items request				
Warehouse worker	Start unloading	Unload items request Transport status				

Table 12 IOT of the Forklift Operator Object

Appendix III Object-Oriented Organizational Design

In addition to laying out plans for automating an organization's information system, design may also involve the use of information technology to enable organizational changes. Such changes often require the organization to break its current business rules and assumptions, and replace them with new rules. Object-oriented design can be used as a tool to illustrate how a changed organizational process will operate. For such changes, one may examine the possibilities of eliminating objects, adding new objects (roles), changing communication paths, and automating large parts of the organization. Research into these matters is a possible avenue for future work.

Figure 13 portrays substantial changes to how the ACME warehouse processes a withdrawal request. The example demonstrates only one of many possible changes one can make to the model. The warehouse manager object in the process has been eliminated . Its role has been diffused by changing the rules of operations and by delegating the remaining responsibilities to the inventory system and truck scheduling system. One of the rules that have been changed is that truck drivers are expected to arrive at a warehouse at a predetermined time rather than a self-determined time. Rather than having the truck driver receive a transport order from the planner, and then telling a warehouse to prepare for (un)loading, the predetermined time is scheduled by the truck schedule system. The transport order that the planner sends to the truck driver will include a newly added arrival time. In addition, the warehouse worker directly requests that the inventory system update any changes in the inventory status, instead of sending a request to the warehouse manager to perform this operation.

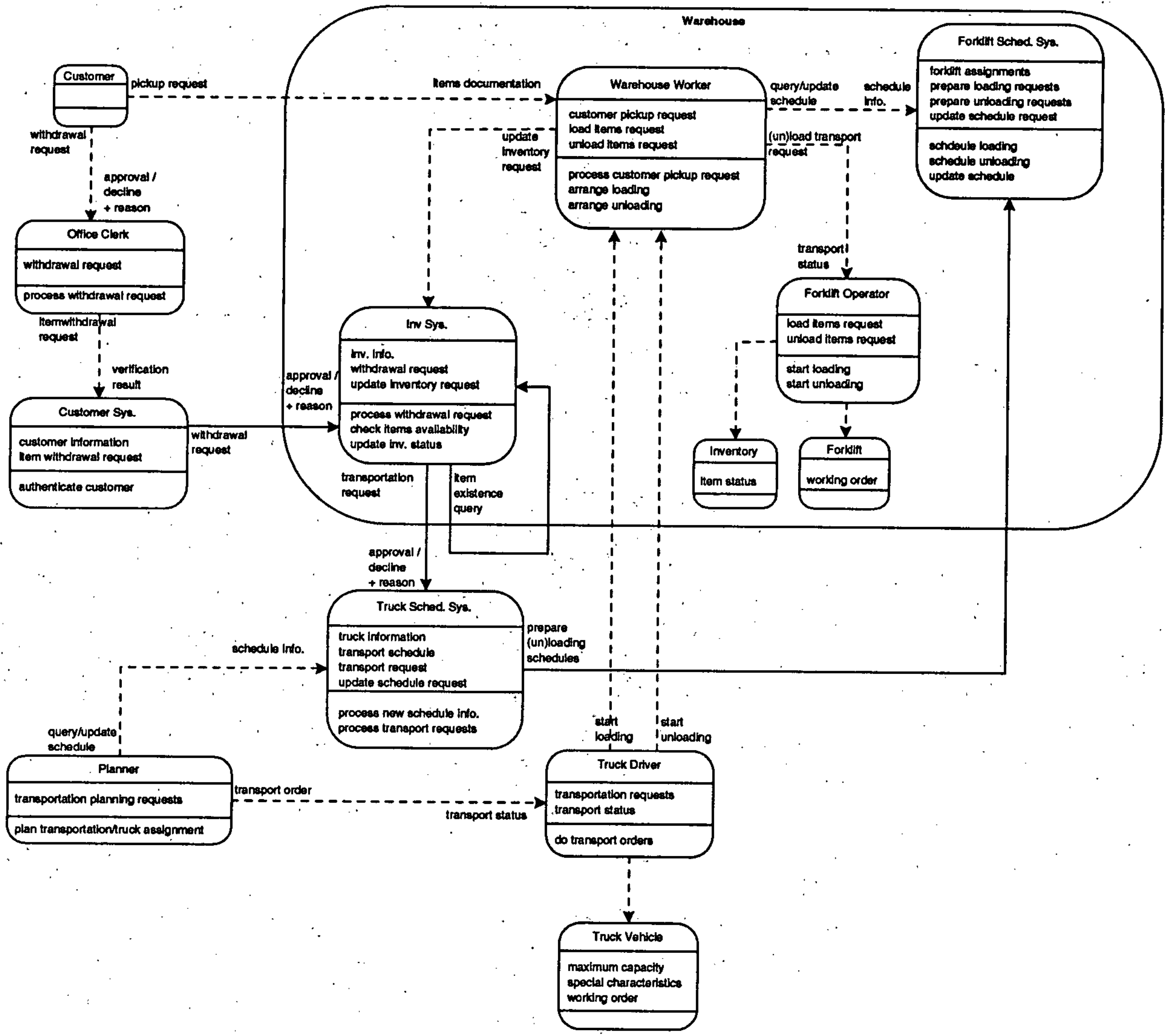

Figure 13 Enterprise model redesign