EXTENDING THE OBJECT-ORIENTED ENTERPRISE MODELING CASE TOOL
FROM ANALYSIS TO LOGICAL DESIGN
by
JINGXI LU
Ph.D., The University of British Columbia, 1997
A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE IN BUSINESS ADMINISTRATION
in
THE FACULTY OF GRADUATE STUDIES
(Faculty of Commerce and Business Administration)

We accept this thesis as conforming
to the required standard

THE UNIVERSITY OF BRITISH COLUMBIA
September 1999
© Jingxi Lu, 1999
In presenting this thesis in partial fulfilment of the requirements for an advanced degree at the University of British Columbia, I agree that the Library shall make it freely available for reference and study. I further agree that permission for extensive copying of this thesis for scholarly purposes may be granted by the head of my department or by his or her representatives. It is understood that copying or publication of this thesis for financial gain shall not be allowed without my written permission.

Department of Commerce and Business Administration

The University of British Columbia
Vancouver, Canada

Date Feb 22, 2000
Abstract

Object-Oriented Enterprise Modeling (OOEM) is a newly developed methodology for object-oriented analysis and design based on ontological theories. This study develops a new version of Computer-Aided Software Engineering (CASE) tool to support OOEM analysis, design, and redesign that aggregates information system components.

The good features of previous OOEM CASE tools have been applied to this new tool. A structured database is designed to support different modeling functionality required by analysis, design, and redesign activities. Features that are common in different modeling activities are shared, while features specific to individual modeling activities are separated. The tool implemented OOEM decomposition to which design and redesign features can be added, supported visualization of OOEM design and redesign, and automated modeling transition from design to redesign. Semantics checks are extended from analysis to design and redesign in order to ensure model integrity and consistency.

An empirical study is conducted to compare this new CASE tool with traditional pen and paper method. The tool is found to be successful in terms of system usefulness, user friendliness and modeling/design capability. The remaining problems associated with the tool and the corresponding remedies are discussed.
# Table of Contents

Abstract ........................................ ii  
List of Figure ........................................ viii  
List of Tables ........................................ x  
Acknowledgement ........................................ xi  

1 Introduction ........................................ 1  
1.1 Object-Oriented Enterprise Modeling Methodology ........................................ 1  
1.2 Computer-Aided Software Engineering (CASE) tool ........................................ 3  
1.3 Motivation and objectives ........................................ 8  
1.4 Thesis outline ........................................ 9  

2 Object-Oriented Enterprise Modeling (OOEM) ........................................ 10  
2.1 The OOEM fundamentals ........................................ 10  
2.1.1 Basic Concepts ........................................ 10  
2.1.2 Modeling principles ........................................ 10  
2.1.3 OOEM modeling constructs ........................................ 11  
2.1.3.1 Attributes and services ........................................ 11  
2.1.3.2 Objects ........................................ 12  
2.1.3.3 Requests ........................................ 14  
2.2 Building analysis model ........................................ 15  
2.2.1 OOEM rules ........................................ 15  
2.2.2 Model representation ........................................ 15
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.3</td>
<td>Decomposition</td>
<td>18</td>
</tr>
<tr>
<td>2.3.1</td>
<td>Decomposition Fundamentals</td>
<td>18</td>
</tr>
<tr>
<td>2.3.2</td>
<td>Object decomposition rules</td>
<td>19</td>
</tr>
<tr>
<td>2.4</td>
<td>Design</td>
<td>23</td>
</tr>
<tr>
<td>2.4.1</td>
<td>Delegation and delegation rules</td>
<td>24</td>
</tr>
<tr>
<td>2.4.2</td>
<td>Static object and static object rule</td>
<td>26</td>
</tr>
<tr>
<td>2.5</td>
<td>Redesign</td>
<td>27</td>
</tr>
<tr>
<td>2.6</td>
<td>Conclusion</td>
<td>28</td>
</tr>
<tr>
<td>3</td>
<td>Requirements Analysis</td>
<td>29</td>
</tr>
<tr>
<td>3.1</td>
<td>System development guideline</td>
<td>29</td>
</tr>
<tr>
<td>3.2</td>
<td>System requirements</td>
<td>30</td>
</tr>
<tr>
<td>3.2.1</td>
<td>System usefulness</td>
<td>31</td>
</tr>
<tr>
<td>3.2.1.1</td>
<td>Modeling transition</td>
<td>31</td>
</tr>
<tr>
<td>3.2.1.2</td>
<td>Visualization</td>
<td>31</td>
</tr>
<tr>
<td>3.2.1.2.1</td>
<td>Object recognition</td>
<td>32</td>
</tr>
<tr>
<td>3.2.1.2.2</td>
<td>Request and request propagation</td>
<td>33</td>
</tr>
<tr>
<td>3.2.1.3</td>
<td>Semantics checks and error report</td>
<td>33</td>
</tr>
<tr>
<td>3.2.1.4</td>
<td>Modeling and design capability</td>
<td>34</td>
</tr>
<tr>
<td>3.2.1.4.1</td>
<td>Modeling capability</td>
<td>34</td>
</tr>
<tr>
<td>3.2.1.4.2</td>
<td>Design capability</td>
<td>36</td>
</tr>
<tr>
<td>3.2.1.5</td>
<td>Model management</td>
<td>37</td>
</tr>
<tr>
<td>3.3</td>
<td>Summary</td>
<td>38</td>
</tr>
</tbody>
</table>

iv
4  System design
4.1  The system
4.2  The database engine
4.3  The system generator and mode controller
4.4  The graphics generator
4.5  The object relation generator
4.6  The subsystems generator
4.6.1 Object exclusion
4.6.2 Request elimination
4.7  The perspectives generator
4.8  The semantics reporter
4.9  The document processor
4.10 The view controller
4.11 Summary

5  System implementation
5.1  Introduction
5.2  OOEM procedural flow
5.3  Database architecture
5.4  Algorithms
5.4.1 Algorithm for decomposing an object
5.4.2 Algorithm for identifying and linking objects needed in the redesign
5.5  Summary
<table>
<thead>
<tr>
<th>Appendix D. Questionnaire for Empirical Study</th>
<th>109</th>
</tr>
</thead>
<tbody>
<tr>
<td>Appendix E. Subject Background Questionnair</td>
<td>113</td>
</tr>
<tr>
<td>Appendix F. Consent Form</td>
<td>115</td>
</tr>
<tr>
<td>Appendix G. Post Study Interview Record</td>
<td>117</td>
</tr>
<tr>
<td>Appendix H. Algorithms for Graphics Generator and Semantics Reporter</td>
<td>118</td>
</tr>
</tbody>
</table>
### List of Figure

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>A Generic OOEM Object Communication Diagram (OCD)</td>
<td>16</td>
</tr>
<tr>
<td>2.2</td>
<td>A Generic Decomposed View</td>
<td>20</td>
</tr>
<tr>
<td>2.3</td>
<td>Types of decomposition</td>
<td>22</td>
</tr>
<tr>
<td>2.4</td>
<td>OOEM Delegation Patterns</td>
<td>25</td>
</tr>
<tr>
<td>4.1</td>
<td>Working Chart of OOEM CASE tool</td>
<td>41</td>
</tr>
<tr>
<td>4.2</td>
<td>First Layer OCD of CAC Analysis (case 1)</td>
<td>44</td>
</tr>
<tr>
<td>4.3a</td>
<td>Object Relation Generator</td>
<td>46</td>
</tr>
<tr>
<td>4.3b</td>
<td>Object Relation Generator</td>
<td>47</td>
</tr>
<tr>
<td>4.3c</td>
<td>Object Relation Generator</td>
<td>48</td>
</tr>
<tr>
<td>5.1</td>
<td>Simplified Flow Chart of System Implementation</td>
<td>55</td>
</tr>
<tr>
<td>5.2</td>
<td>ER Diagram for OOEM Database Design</td>
<td>58</td>
</tr>
<tr>
<td>5.3</td>
<td>Decomposition of the Financial Office of the CAC Case</td>
<td>63</td>
</tr>
<tr>
<td>5.4</td>
<td>Algorithm for identifying and linking objects needed for redesign</td>
<td>64</td>
</tr>
<tr>
<td>5.5</td>
<td>Different Decomposition Patterns in Two Decomposed views</td>
<td>66</td>
</tr>
<tr>
<td>5.6</td>
<td>Design of Financial Office of the CAC Case</td>
<td>67</td>
</tr>
<tr>
<td>5.7</td>
<td>Design of Salesman of the CAC Case</td>
<td>68</td>
</tr>
<tr>
<td>5.8</td>
<td>Redesign of the CAC Case</td>
<td>69</td>
</tr>
<tr>
<td>A</td>
<td>An OOEM metamodel</td>
<td>100</td>
</tr>
<tr>
<td>C.1</td>
<td>First Layer of North Wind</td>
<td>105</td>
</tr>
<tr>
<td>C.2</td>
<td>Design of the Order Office</td>
<td>106</td>
</tr>
<tr>
<td>C.3</td>
<td>Design of the Warehouse</td>
<td>107</td>
</tr>
</tbody>
</table>
Figure C.4 Redesign of North Wind 108
Figure H.1 Redesign of North Wind 118
List of Tables

Table 2.1 A Generic OOEM Object Template for Internal Object (top) and External Object (bottom) 17
Table 4.1 A Summary of System Component Changes 53
Table 6.1a Summary for questions on system usefulness 78
Table 6.1b Summary for questions on system usefulness 80
Table 6.1c Summary for questions on system usefulness 80
Table 6.2 Summary of questions on user friendliness 81
Table 6.3 Summary of questions on modeling/design capabilities 82
Acknowledgement

I sincerely thank Yair Wand and Carson Woo, my supervisors, for their guidance, encouragement, and support during my graduate studies in the Management Information Systems Division at UBC.

Special thanks go to Xu Zhang at ISM-BC for providing me the OOEM CASE tool protocol from which the current study starts and to Andrew Gemino at SFU for reviewing my empirical study plan and cases. Without their help, this work could not have been done.

I am grateful to all the participants in the empirical study.
1 INTRODUCTION

1.1 Object-oriented enterprise modeling methodology

The object-oriented approach is a fast growing paradigm with its origin in the object-oriented (OO) programming paradigm that is formed largely due to the needs of good programming practice. The OO programming paradigm later developed towards encompassing system design and analysis, driven by the needs of building information systems. However, the analysis is mainly for specifying requirements for designing information systems. Embley et al. (1995) said, “Many so called analysis methods are really preliminary design methods.... [Many] of their features are more appropriate for design than analysis” (p.19).

Information processing is part of enterprise activities. Ideally, the information system should support optimally enterprise business activities. From this point of view, modeling enterprise systems, including both information processing and enterprise business activities is a good starting point for building information systems. 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). This argument is supported by the fact that information systems tend to change slowly. They also fail to meet fast changing business needs or
reorganization because there are no good enterprise models linking enterprise activities to their information processing.

There is growing interest in enterprise modeling where the emphasis is on modeling the problem domain. Most existing object-oriented modeling methods are inappropriate for enterprise modeling for two reasons: (1) They are preliminary design methods (Embley, 1995); (2) They do not provide basic principles and rules to guide how to think about a problem domain from the perspective of an enterprise (Jung, 1997, Tan, 1997). Unified Modeling Language (UML) is a notion that has been used in enterprise modeling, system design, and software development. Its core is the unification of three modeling methodologies proposed by Booch (1993), Rumbaugh (1991), and Jacobson (1992), respectively. Booch’s Object Modeling Technology is a notion of graphical symbols representing various aspects of a model. Rumbaugh’s Object Management Technology is an OO system analysis and design method focusing on real-world system components. Jacobson’s Object-Oriented Business Engineering Technique consists of interrelated use cases and object models. The use cases are narrative descriptions of domain processes in a structured prose format. The object models clarify the inner workings of a company’s processes, products, services, resources, and their relationships. The technique can proceed to a logical design and to development of an information system model. However, UML does not guide a developer in how to do OO analysis and design, or what development process to follow (Larman, 1998).
To meet the need for a sound enterprise modeling methodology, Wand and Woo (1993) initiated an object-oriented enterprise modeling methodology (OOEM) that took an ontological approach to provide the basic principles and rules for formalizing the object-oriented analysis. Zhao (1995) formed the framework for OOEM analysis in which the OOEM model representation (object communication diagram and object temperate) and the request propagation were proposed.

Most OO methods are geared towards design. As mentioned above, these methods do not have a good foundation for enterprise modeling. OOEM with a theoretical (ontological) foundation has lacked a method for generating an IS design. Jung has dealt with this issue by proposing a method to generate an IS design from an enterprise model (Jung, 1997). To that end, he has introduced two main concepts: object delegation (which is a special form of decomposition) and the notion of static objects.

1.2 Computer-Aided Software Engineering (CASE) tool

Computer-Aided Software Engineering (CASE) tool has been widely used in OO analysis and design. In this section, I briefly review some CASE tools to show why they cannot be used to do our work, and why developing our own OOEM CASE tool will make additional contributions to the CASE tools literature.

Some of the existing CASE tools are actually graphic tools (Boggs, 1999). These CASE tools only provide various graphic symbols for OO modeling. Typically, they are not
developed based on an OO method and do not have a database to store data for further applications (e.g., design or code generation). One example of this kind of tool is the SilverRun, which can be used to draw various graphics (www.silverrun.com/product). Graphic tools do not work for our purpose because they are not designed to support the OO modeling from enterprise level down to system design and implementation with an underlying OO method.

Some other CASE tools mainly support software development. One example is the Oracle Designer 2000. It allows users to model an existing database and then convert the model to a relational database design (www.oracle.com/tool). These CASE tools do not serve our purpose either, because they focus on low-level system analysis and design.

Rational Rose is one of the CASE tools capable of doing OO modeling from the top down. It is an integrated object-oriented development environment encompassing OO process analysis, OO component analysis and design, and code generation. The new version of Rational Rose 98i has the following main features (www.rational.com/products/rose/reviews):

- **Business Process Modeling** - Rational Rose provides an integrated approach to modeling by integrating business process modeling into its application modeling environment. With full support for the Unified Modeling Language (UML) 1.3 specification for Activity Diagrams, Rose allows analysts to model the workflow of business processes and activities and incorporate this information into the
application model. Developers then use the same model to derive system requirements, design the application and generate code. The ability of multiple members of a development team to work collaboratively from a single design model allows them to build higher quality software more quickly and effectively.

- **Data Access** - This functionality, available in the Enterprise version of Rose 98i for Windows, allows Visual Basic developers to map business objects to physical database tables and automatically create runtime data components to access the database. Existing database schemas developed by a database administrator can be reused by mapping to the application code for generation of the data access layer of an application.

- **Parallel Development** - A new integration between Rational Rose and Rational ClearCase delivers the most open and scalable solution available today for software teams to conduct parallel model development. The new integration allows developers to work in parallel on a shared model, stored in ClearCase, and has fine grain control of model changes. To support this large-scale parallel development, Rational Rose includes the Model Integrator to improve collaboration of multi-user development teams. Together with Rational ClearCase, the Model Integrator allows multiple developers to work concurrently on the same model elements, and automatically integrates changes into a common model to ensure that the entire development team is working on a consistent model. This allows development teams to truly work in parallel, removing the obstacle of one developer having to wait for other developers on the team to
finalize their changes. In addition to ClearCase, Rational Rose also supports Visual SourceSafe and other Source Code Control (SCC)-compliant tools.

- **Application Modeling** - Rational Rose 98i provides tight integration with the leading Integrated Development Environments (IDEs) on Windows and UNIX platforms to provide seamless interchange between the application model and code. For Microsoft Visual Studio 6.0 developers, Rational Rose supports full markerless code generation and reverse engineering for Visual Basic 6.0 and Visual C++ 6.0, provides code generation and reverse engineering of Microsoft Foundation Classes (MFC) applications, and tight integration with the Visual J++ 6.0 environment. For UNIX developers, Rational Rose provides code generation and reverse engineering for C++, Java, Ada 83 and Ada 95.

For multi-platform Java development, Rose supports JDK 1.2, 100% Pure Java, Java Bean property generation, nested classes and components such as jars, zips and cabs. In addition, Rational Rose can automatically generate Java source code and reverse engineer Java source and byte code. Rational Rose also supports CORBA 2.2 including full round-trip engineering of IDL and support for other CORBA-specific features such as Constants, Enums, Exceptions, Interfaces, Structs, Typedefs and Unions.

Although Rational Rose supports many modeling activities ranged from business process modeling to code generation, it has some weaknesses related to UML: (1) UML does not provide rules and principles to guide modeling (i.e., how to create use cases and object
models); (2) UML does not support object decomposition. Instead, it uses different levels of use cases (i.e., high level format and expanded format) and different rankings of use cases (i.e., primary, secondary, and optional use cases) to describe domain processes. It then uses “includes” relation and “extends” relation to relate multiple use cases (Larman, 1998). However, this approach does not provide guideline on how to do so; (3) For UML, logical design involves identifying business objects that will use an IS. In other word, the IS does not result from a separation of business activities and information processing.

As opposed to UML, OOEM design starts from a high-level enterprise model including both information processes and enterprise business activities. Decomposition plays an important role in decomposing composite objects and separating business activities from information processes that will be captured by OOEM design. These modeling activities will be guided by OOEM principles and rules. Obviously, Rational Rose does not serve our purpose. It will therefore be an important contribution if this study can develop an OOEM CASE tool capable of supporting OOEM design.

The OOEM CASE tool development is a continuation of two previous studies. Tan (Tan, 1997) made the first move by proposing a meta-model that allowed the OOEM rules to be programmed and implemented in a CASE tool (OOEMCT). With this meta-model, model correctness can be checked semantically by a computer. Zhang (1998) continued the effort by developing a CASE tool (OOEM CASE tool) that fully supported the OOEM model representation (e.g., the OOEM object communication diagram and object template) and the successful semantics check feature of OOEMCT.
1.3 Motivation and objectives

This study is motivated by the need to develop the existing OOEM CASE tool further down to supporting the OOEM design proposed by Jung (1997). This system has to be useful and user friendly, and can increase the efficiency and effectiveness of OOEM design of information systems.

Accordingly, the objectives of this study are to build a new system on top of the OOEM CASE tool by Zhang (1998) that should be capable of doing OOEM design and ensuring integrity of the transition between analysis and design. The new system should be empirically tested to see whether it achieves the design objectives.

This study will answer the following questions:

(1) What are the requirements for the OOEM CASE tool to support both OOEM analysis and design?

(2) Given the requirements, how do we design and develop the system that meets the requirements?

(3) What are the potential and remaining problems of putting the system in real use?
1.4 Thesis outline

The thesis is organized as follows: Chapter 2 briefly describes the OOEM methodology with emphasis on the OOEM fundamentals, modeling principles and rules for system analysis and design. Chapter 3 studies the system requirements. Chapter 4 and 5 present the design and implementation of the system. Chapter 6 presents the results from an empirical study on the system. A summary and conclusion are given in Chapter 7.
2 OBJECT-ORIENTED ENTERPRISE MODELING (OOEM)

Object-oriented analysis and design are essentially modeling activities. Analysis examines existing systems, and design involves prescribing or planning for future systems (Olle et al. 1991, P.2). The OOEM analysis and design are modeling activities guided by the OOEM methodology consisting of a set of fundamental principles and rules originated from ontological theories (Wand and Woo, 1993).

2.1 The OOEM fundamentals

2.1.1 Basic Concepts

In OOEM, a system is defined as a group of interacting components that are performing certain tasks in response to triggers from its environment. In particular, OOEM applies to enterprise systems with regular business activities and information processing. A model is an abstract representation of a system based on a modeling method, subject to an analyst's or designer's (hereafter called "modeler") understanding of the method and the system, and subject to the modeler's modeling rationale. Analysis studies the system to explore its inner working processes triggered by its environment. Design lays out plans for information systems development based on the outcome of the analysis.

2.1.2 Modeling principles

For a model to be good, the modeling method must be theoretically sound and should provide rules to guide model construction without restricting a modeler's imagination and
creativity. The modeling rules are necessary for the modeling activities to be consistent and reliable.

An OOEM model simulates an enterprise system, possibly supported by information systems. Only relevant aspects of the system are modeled. These aspects are determined by a set of requests from outside the system. The model components therefore include both external components (business clients and suppliers) and system components (enterprise units), all of which are modeled as objects. The system's components communicate only to fulfil the tasks activated by the external stimuli (requests) sent by the external components that form the environment of the system.

2.1.3 OOEM modeling constructs

2.1.3.1 Attributes and services

Ontologically, a thing has properties that are modeled by the state and state transformation laws of an object. These, in turn, are modeled by the attributes and services. The attributes determining the state, therefore, represent the properties of the thing, with the internal attributes representing its intrinsic properties and the interface attributes representing its shared (or mutual) properties. For example, in case 1, the "sales request" is an interface attribute because it can be changed directly by the Salesman object, while the "customer credit" attribute is internal to the Financial Office object because it cannot be accessed directly by the Salesman. The state transformation laws are modeled as services of the object. They are the possible actions the object can take to
change the internal attributes as a result of object interactions. In ontology, object interaction has its equivalence called coupling.

2.1.3.2 Object

An object models a thing in the problem domain, with a changeable state determined by state variables called attributes. An object might have two types of attributes. Interface attributes are those accessible and changeable directly by other objects; internal attributes are those accessible and changeable only through the services of the object itself. An object can change its state and respond to requests by other objects via its services. The attributes and services of the object constitute the properties of the object.

Objects with the same properties, equivalent to things with the same functionality in the real world, belong to the same class. Object classes represent roles that are played in an organization.

An object is a composite object if it is an aggregate of several objects and has emergent properties, that is, properties in addition to the attributes and the services of its component objects. For example, the Financial Office in case 1 of the two business cases used in this study (Appendix B) is a composite object that has two component objects: the Financial Accountant and the Payroll Accountant. It possesses all their properties including attributes (sales requests, shipment info., etc...) and services (process sales and complete sales) (Fig. 4.2 and Fig. 5.3) and its emergent property (not shown). One example of emergent property is the "processing power" of a computer that emerges at
the composite (computer) level and is not a property of any its components. It can be modeled as an attribute of the computer. This thesis does not study emergent property.

Objects communicate by changing the interface attributes of each other. Changing interface attributes due to object communications is conceptualized as being made through requests. The stimuli sent from external objects into the system are called external requests. Other requests made inside the system while it responds to the external requests are called internal requests.

Ontologically, a thing has properties that are modeled by the state and state transformation laws of an object. These, in turn, are modeled by the attributes and services. The attributes determining the state, therefore, represent the properties of the thing, with the internal attributes representing its intrinsic properties and the interface attributes representing its shared (or mutual) properties. For example, in case 1, the “sales request” is an interface attribute because it can be changed directly by the Salesman object, while the “customer credit” attribute is internal to the Financial Office object because it cannot be accessed directly by the Salesman. The state transformation laws are modeled as services of the object. They are the possible actions the object can take to change the internal attributes as a result of object interactions. In ontology, object interaction has its equivalence called coupling.
2.1.3.3 Requests

Requests model the interactions among objects. An object interacts with other objects by sending requests that cause state-changes among the objects directly and possibly through state transformations of the receiving objects. State changes of objects are events. Requests have two types: internal requests and external requests. External requests are those events that originate with external objects. Internal requests are those events that originate inside the modeled system.

A request from an external object determines the relevance and the boundary of the system to the request. The request causes change in the interface attribute and, thus, state-change in the receiving object. This change can activate a service that might cause further state transformations of the object. The state transformations might need further information from other objects. This can be done via requests to other objects, which result in request propagation through the system. The boundary of the model is set by the extent of the propagation of requests.

The above discussion indicates that the OOEM modeling constructs have their ontological basis. Since an enterprise model should capture information about organizational concerns, such as products and customers, ontology that provides people with concepts on how to reason about the real world can be a foundation for enterprise modeling.
2.2 Building analysis model

The OOEM analysis provides a systematic way to reason about a problem domain. The result of an OOEM analysis is called an analysis (enterprise) model.

2.2.1 OOEM rules

OOEM rules provide the core principles for using the object-oriented constructs discussed in enterprise analysis. When starting modeling, a modeler might have the following difficulties: (1) What is the scope of the system (and its environment); (2) How to identify a thing in an organization as an object of the system; (3) What is the appropriate level of detail to stop representing things as objects; (4) How to classify an object in the system; (5) what are the attributes, services, and requests associated with events triggered by the system environment. The OOEM rules guide modelers in defining modeling scope, object identification, object composition, classification, and interactions. In Appendix A, these rules are discussed.

2.2.2 Model representation

With the OOEM rules, an OOEM model proposed by Zhao (1995) can be represented by an object communication diagram (OCD). This diagram contains external objects, internal objects, and requests, attributes, and services for the internal objects. Fig.2.1 is a generic object communication diagram where the external object has only a name and is
represented by a square box. An internal object is represented by a rounded-square box. It has a name, attributes, and services. A solid arrow represents a request that might have a "Request" and a "Response" or only a "Request". The "Request" and "Response" constitutes a request-response pair.

![Diagram of an object communication diagram](image)

Figure 2.1 A Generic OOEM Object Communication Diagram (OCD).

The Object Template (OT) provides more detailed information about an object in a table format that allows an easier semantic check concerning relationships among services, attributes, and requests by using the OOEM rules. Table 2.1 is a generic OOEM OT. In an OT, the object name is input as the header. Other key characteristics of the object are...
input according to columns. For an internal object, the requesting object column identifies the client objects which request services from the object. The service column records the object’s corresponding services invoked by the client objects. Interface attributes and internal attributes are input to reflect the state change of the object due to the requests from its clients. The access mode specifies how the attributes are accessed by the service (i.e., used or modified). The last two columns are used to record the requests spawned by the object because further information or service is required from other objects and the recipient objects that provide the information or services.

<table>
<thead>
<tr>
<th>Object Name</th>
<th>Requesting Objects</th>
<th>Services</th>
<th>Interface Attributes</th>
<th>Internal Attributes</th>
<th>Request Generated</th>
<th>Receiving Objects</th>
</tr>
</thead>
<tbody>
<tr>
<td>Object 1</td>
<td>service 1</td>
<td></td>
<td>incoming interface attributes</td>
<td>Internal Attributes to support service 1</td>
<td>Request generated from service 1</td>
<td>Object 2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>returning interface attributes</td>
<td>Access Mode</td>
<td>Request generated from service 1</td>
<td>Object 3</td>
</tr>
</tbody>
</table>

Table 2.1 Generic OOEM Object Templates for internal object (top) and external object (bottom).

OOEM is concerned with a subset of characteristics of an external object (i.e., the requests sent by the external object and the services it provides to the system). Hence, a
different OT is used, that is, only requests made to a system and received from the system and the internal objects receiving the requests and the interface attributes and services of the internal object corresponding to the requests are included.

2.3 Decomposition

Decomposition is fundamental for both system analysis and design. So far, no dominant methodology has emerged, though several methodologies have been proposed (Wand and Weber, 1989). In analysis, decomposition is used to break down an existing complex system into its component parts for further analysis. As discussed in next section, it is used to identify delegate objects and derive static objects in design.

2.3.1 Decomposition fundamentals

In OOEM, decomposition will be done based on the following fundamentals (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 (e.g., no services are lost).

3. The decomposition of an information system should reflect a decomposed view of the system.

   - A good decomposition is defined with respect to behavior 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 that must be evaluated with respect to a particular transformation. A decomposition might be good in the context of one transformation but poor in the context of another transformation (Wand and Weber, 1989).

The same principles above are used for decomposing a system and for viewing it as separated from its environment.

2.3.2 Object decomposition rules

The OOEM assumptions about the system make decomposition a process of breaking down a composite object. A system is the highest-level composite object. A composite
object binds its components together through requests they send to each other when the system carries out a request made to it. The composite object itself is a component object of its upper level composite object. The decomposition rules enforce the idea that for each request made to a composite object there must be at least a component object that acts as an interface to begin the initial processing of the request by providing a service. The initial object may spawn other requests to other component objects for services. Object decomposition results in the decomposition of attributes and services of the composite object.

(a)

![Diagram](image1)

(b)

![Diagram](image2)

Figure 2.2 A generic decomposed view where (a) shows the whole object view of object B before decomposition and (b) shows the decomposed view (inside dash box) of B after decomposition. In this case, the request-response pair (Req1/Res1) has been split in the decomposed view.
The decomposition rules maintain the integrity of an object when one switches from the whole object view of the object to its decomposed object view. That is, there are no requests in the whole object view that are not transferred to the decomposed object view (Fig2.2).

Jung (1997) generalized four types of decomposition when breaking down an object (Fig.2.3), where (a) is the simplest decomposition that divides both the attributes and the services; (b) shows a service decomposition since the service accesses two attributes in the composite object; (c) presents an attribute decomposition since it is used by two services in the composite object; (d) depicts a decomposition of both attribute and service. If a request-response pair made to a composite object is split, such as object B in Fig. 2.2, the components (B1 and B2) will both be interface objects. An interface object is a component object receiving a request from or sending a request to an upper level object in the decomposed view.
Figure 2.3 Decomposition types. Arrow represents decomposition (Jung, 1997, p35).
2.4 Design

OOEM design is high level logical design. The whole idea of OOEM design is based on the fact that attributes of an object can include both the state of the object and the knowledge of other objects that can be separated from the object for automation and sharing. OOEM design consists of two mechanisms: the object delegation and static object derivation. These will be discussed in the next two sections. Decomposition is an important step towards OOEM design because: (1) for the sake of simplicity it is not necessary to show all object properties in the upper layer of a model, and very often modelers do not know some properties of an object before investigating it further. Under these circumstances, delegating objects or creating static objects could result in an incomplete or incorrect design model. For example, since delegate attributes are internal and represent the state of static objects, one cannot delegate them and create static objects if they are absent at a high level; (2) Even if you solve the problem in (1) by requiring users to identify detailed properties of high-level objects, delegating objects or creating static objects at the upper level is not a good practice because it will leave a question unanswered: what are the components of the high level delegate objects and how can one use them to design IS? (3) When an object is decomposed, one has a clearer idea of what static objects can be derived and what knowledge can be delegated from its component objects. This clarity will also ensure that the identified delegate and static objects correspond better to the detailed operations of the business.
As a consistent approach, OOEM OCD and OT still apply in OOEM design. The resultant model from an OOEM design is called a design (enterprise) model. It includes both organizational and information processing components. When an enterprise model is designed, a (further) design is needed to separate the business activities from the information processing parts and aggregates the information processing parts to form information system(s). This will be discussed in section 2.5.

2.4.1 Delegation and delegation rules

A delegation is a role transference in which an object (delegator) transfers some roles to a delegate object. OOEM delegation is a mechanism that enables one to automate non-human components. It involves reallocating or moving attributes and their corresponding services from the delegator to the delegate object. Delegation is therefore a specific decomposition where attributes and services are delegated from human objects to non-human objects. The OOEM modeling principles and the decomposition principles are applicable to object delegation. In particular, encapsulation (attribute custodian rule) of object states should not be violated. Jung (1997, p.39) proposed the following rules to ensure encapsulation of object states:

1. Delegation of an attribute should be accompanied by services that use the attribute.
2. Delegation of a service should be accompanied by attributes that are used by the

---

1 Though there is no restriction on delegators, in many situations where a system has been decomposed to certain levels of detail, a delegator is a human being.
3. Delegation of both attributes and services that are related.

OOEM delegation rules guide the transition from the enterprise model to the design model of an information system. Based on the rules, object delegation can separate organizational activities from information processing without violating OOEM rules and principles. Each delegate object in the resultant (design) model can be a candidate component for the logical design of an information system.

Figure 2.4 OOEM Delegate Patterns (Jung, 1997, p41).
The OCD of a design model therefore reflects degrees of automation of the design. Jung (1997) proposed four basic patterns of communication in a design model (Fig.2.4). Pattern (a) is the scenario before delegation. Patterns (b-d) reflect different degrees of automation. For example, pattern (b) presents a low level of automation. The delegate object is used to store and retrieve information; Pattern (e) shows a high level of automation where the original communication has been fully automated. Pattern (d), where a non-human delegate object makes a request of a delegator, might indicate that objects A and B cannot communicate with each other for certain reasons (For example, work in different time), they then need delegate A as a mediator in order to communicate.

2.4.2 Static object and static rule

When modeling a system, certain things may possess knowledge about the state of other, non-active things. For example, an inventory clerk will know the state of inventory items. Static objects represent things that do not participate in the activities within a system. However, these are things which dynamic objects know about and might change the state of. At enterprise model level, information about these things and the action that changes their states are represented by attributes and processes of dynamic objects. In OOEM design, the information can be delegated to delegate objects. Therefore, static objects can be included to replace attributes of original or delegate objects in order to show the objects about which the system will have information. Accordingly, the actions that apply to the state of static objects (use or modification) will be represented by requests from the original or delegate objects to the static objects. The decision what to delegate is actually
a decision what will be done by the information system (as opposed to the original, organizational objects). When the delegate objects become components of an information system design, the state changes of static objects caused by an event are then reflected as information changes in the information system.

Static objects possess only state and provide no services. A request might be used to highlight the manipulation of the state, though no corresponding service is attached to it. Therefore, the static objects must be shown with the objects that either read or change the state. Jung (1997, p.42) proposed the following static rule: static objects that do not undergo state changes are not modeled, since they provide no additional information. This is in accordance with the OOEM principle of minimality that requires that no redundant state variables be allowed in a model of a system.

2.5 Redesign

Object delegation and static object derivation of OOEM design are mainly for reasoning about what services of individual objects can be automated. Hence this stage of design might not be involved in reasoning about designing information system for the enterprise as a whole. Consequently, objects delegated or derived based on the information of individual objects under consideration might be changed (addition/deletion) when one starts designing information systems for the whole enterprise. Redesign is part of OOEM design that allows one to design (an) information system(s) for an enterprise. In this thesis, the resulting model is called a redesign (enterprise) model.
Redesign is a part of OOEM design, aiding in the aggregation of all or part of the delegate objects and static objects in order to form an information system design for the enterprise or part of it. It involves organizational changes enabled by the information system(s). Some objects might be eliminated and some might be added to the enterprise because of the introduction of the information system(s). The communication patterns among objects in a design might be changed after introducing the information system(s) in a redesign. Jung (1997, p.80) demonstrated such a case. Redesign is high-level design and no specific rules for redesign activities have been formalized in OOEM.

2.6 Conclusion

The OOEM analysis and design are modeling activities guided by a set of fundamental principles and rules originated from ontological theories. Built on top of an enterprise model, OOEM design that lays out plans to automate some of its components also applies the principles and rules for analysis. Whether or not to delegate objects is largely a subjective decision. Though Jung (1997) provided some good guidelines for this, they cannot be computerized. After finishing defining delegate objects and deriving static objects, one can in the redesign stage further reason about creating information system(s) based on those delegate objects in the design model. Again, the OOEM modeling principles and rules are applicable to this redesign activity.
3 Requirements Analysis

Requirement analysis is critical for every successful system development. The requirement analysis in this chapter answers the first research question listed in section 1.1.

3.1 System development guideline

The principal guideline for building the OOEM CASE tool is building the system to be useful and user-friendly. Building a useful and user-friendly system is the most important goal of system development. "It is not our job to produce clever computer applications that reduce productivity" (Landauer 1995, p. 663). For the OOEM CASE tool to be useful and user-friendly, four criteria are set when designing the tool:

1. The system must fully implement the OOEM model representation, the principles, and the rules for both analysis and design in an integrated fashion. Integrity means that both analysis and design use the same modeling representation and the common principles and rules. It also means that any design activities based on the analysis model should not change the analysis model, while any changes made in the analysis model must be reflected in the design model.
(2) Changing modeling activities from analysis to design or from design to analysis, whenever necessary, must be easy. It is important for the system to be able to change from one model to another easily because it can be expected that there exist frequent changes of modeling activities with the tool.

(3) The tool should have well-designed Graphic User Interfaces (GUIs), sufficient hints for interface icons and buttons, and necessary messages guiding modeling activities.

(4) The system must make modeling activities effective and efficient. System effectiveness measures how well the system helps modelers to model important facts in a problem domain. A direct measure of effective modeling is the correctness of its resulting model. Whether or not the system provides modelers with cognitive advantages in understanding, structuring, and modeling the problem domain is another measure. Cognitive advantages are subtle and hard to measure. In this study, a construct called modeling/design capability (section 6.5) is used to measure it in the empirical study. System efficiency measures the resources required to be able to model the problem domain (Wand 1993, p.2). The direct measure of system efficiency is the speed and easiness with which a modeling task is completed.

3.2 System requirements

The purpose of the research project is mainly to enhance the previous system with object-oriented design capability (Zhang, 1998). The successful features found in the previous
tool should be inherited by the new one. Therefore, this requirement analysis focuses on
finding the successful features in the previous system and the new requirements for the
new tool, based on the empirical studies conducted by Zhang (1998), user feedback and
the specific and common characteristics of OOEM analysis and OOEM design discussed
in chapter 2.

3.2.1 System usefulness

This sub-section addresses the requirements associated with system usefulness.

3.2.1.1 Modeling transition

The system will support three modeling activities (analysis, design, and redesign) with
commonality and difference. It will also support a transition mechanism that allows the
modeler to switch among the three. Features common to the three modeling activities
will be shared, while features specific to individual modeling activities will be separated.

3.2.1.2 Visualization

One major success of the previous system is its full support of the OOEM Object
Communication Diagram (OCD). A model can be constructed on a GUI. Visualization
enables modelers to directly manipulate graphical objects on the GUI and thus reduce
their mental processing operations\textsuperscript{2}. Visualization offers them cognitive advantages\textsuperscript{3} during the course of extracting, processing, understanding, and responding to more relevant information of the model. The current system is therefore required to support the GUI in design and redesign.

3.2.1.2.1 Object recognition

One key requirement for visualizing model construction and representation for the new tool is to find a way to distinguish two more types of objects (i.e., delegate object and static object) from others, because OOEM OCD uses only two graphics (square and rounded square) to represent external and internal objects, respectively.

Object changes from one object type to another object type at any stage of model construction is also required because system environment is set by a modeler whose understanding of the problem domain and modeling interest might be different from others or might be changed over time. Options for object type change enable modelers to decide object type until they are ready. Two more options for the delegate and static objects are needed.

\begin{footnotesize}
\textsuperscript{2} "Mental processing operations include requirements for the user to learn complex commands and syntax, memorize encrypted codes and abbreviations, or translate data into other units or formats before they can be applied to the problem at hand" (Brown 1988, p.4).

\textsuperscript{3} The human ability to extract information from visual scenes is much more fundamental than ability to manipulate data verbally or arithmetically (Schwartz and Howell 1985).
\end{footnotesize}
3.2.1.2.2 Request and request propagation

Request is another important component in OOEM OCD. As I have discussed in Chapter 2, a single arrow is used to represent a request from a requesting object to a receiving object. Request propagation reflects the business processes associated with an external request. It must be presented completely and correctly. One-direction propagation without response can show request propagation through the problem domain and business structure, but not through the full interactions between requesting and receiving objects. There is a need to represent the response from the receiving object to the requesting object clearly, because when the requested object is decomposed, the request and response can be received and sent by different component objects in its decomposed view. In this case, the response becomes a request sent to an upper level object in the decomposed view (see sections 3.2.1.4.1 and 5.4.2 for further discussion).

Furthermore, since a static object does not provide service, the object from which a static object is derived may or may not be able to make a request to the static object. The tool should be able to represent static objects for both cases.

3.2.1.3 Semantics checks and error report

The semantics checks introduced by Tan (Tan 1997) in his OOEMCT were successful, but with some weaknesses in reporting errors found. Zhang's study indicated that his tool has fixed these weaknesses by using a new reporter (Zhang, 1998). As discussed in Chapter 2, the OOEM rules are still in effect in OOEM logical design including redesign,
except for static object because it does not provide service. Therefore the new system is required to extend the new reporter designed by Zhang to the logical design.

3.2.1.4 Modeling and design capability

Modeling/design capacity is an overall measure of the tool. Here, it represents several important aspects of system usefulness.

3.2.1.4.1 Modeling capability

The system should be useful for idea structuring\(^{4}\). To achieve this, a sound GUI that fully implements OOEM Object Template (OOEM OT) is required. Zhang (1998) designed such a GUI that allows one to fill object details and establishes the relationships among all the object details according to the OOEM rules. With this GUI, it is expected that the course of completing an OT would invoke mental processes that help one to structure idea. The GUI must be modified, however, to support logical design where static objects do not have services and requests.

The semantics checks can play two roles – mistake correction and next round of idea restructuring. If semantic inconsistencies appear in a model, one possibility would be that one makes mistakes in constructing the model, which would be corrected by filling additional information. Another possibility would be that the problem domain has not

\(^{4}\) The process of gathering information of organization activities, discovering and collecting ideas, structuring them, and finally putting them together into a
been analyzed correctly, which necessitates a restructuring of the entire model. Both roles are expected to provide some cognitive advantages in the modeling process.

Supporting object decomposition (in particular, supporting the scenarios shown in Fig. 2.3) is fundamental for OOEM analysis. A well-decomposed system is required by the OOEM design. An enterprise system can therefore be modeled typically in several layers of activities. The layer without decomposed objects is the first layer of the model. Every time an object is decomposed, its component objects appear in one level down; for example, in the second level and so on. The component objects along with the upper level objects (external to the components) which communicate with the component objects form a decomposed view of the object. There is one scenario that is worthwhile to mention here. OOEM uses a solid arrow to represent a request/response pair. When sending a request with a response from an object to another with the tool, there is no attribute specified by the user for the response in the requesting object. The only information stored in the database is the name of the response. However, when an object is decomposed a request and its response in the upper level can be sent to and received from different component objects in the lower level (Fig. 2.2). In this case, one has to use an arrow and a “request” name to represent the upper level response. The response now must have an attribute associated with it in the receiving object because a request must change the state of the receiving object. It is inconsistent with OOEM rules because a response will always be a response no matter what it is in the upper level or in the lower comprehensible model, according to Card and Moran, is called idea structuring (Card and Moran, 1995).
level. One way to solve this inconsistency is to force users to use "/response" as the request name with an arrow in the lower level. This measure is anticipated to reduce confusion between the request and the response. In section 5.4.2, this issue will be further discussed.

3.2.1.4.2 Design capability

Once business activities are fully captured and represented by an analysis model, the tool is further required to support object delegation and static object derivation. Forming delegate objects and deriving static objects are allowed only under the decomposed view because:

(1) Delegate objects and static objects will be meaningful only when objects are decomposed into finer details. Users can better understand what services can be delegated for automation after further investigating individual business units (see section 2.4).

(2) A system often has many objects so that it is not easy to display all of them along with their delegate objects and static objects. By displaying delegate objects and static objects in the lower level, the number of objects will be greatly reduced.

When an object has no component objects, forming delegate objects and deriving static objects are still required to be completed in its decomposed view. It does not violate the
OOEM Composite Rule because delegation and static object derivation are decomposition activities in design.

Since object delegation and static object derivation are allowed only in the decomposed view of a design model, the system must be able to bring them up to the first layer so that users can do their information system design. Once the component objects are brought up, the original first layer objects of the design model must be deleted, and their communications become the communications among their component objects. Deleting the first layer objects does not affect the design model because redesign starts from an independent copy of the design model. To avoid making any mistakes when completing the above tasks, the system should support both activities automatically.

3.2.1.5 Model management

An enterprise model probably has tens or hundreds of objects, which causes a problem of how to display them. Though decomposition feature of the system developed mainly for dissecting complex system has played a role in alleviating this problem, there is a need to add a subsystem feature allowing one to partition a system into several subsystems. One emphasis is that no requests and responses will be lost in the subsystems so that the whole system can be recovered completely when combining the requests and responses together. This feature which has been developed in the previous system, will be extended to design and redesign in the new tool.
The tool should also allow one to suspend model construction at any stage and resume it at a later time. No information should be lost in any of these iterations.

3.3 Summary

The above analysis is summarized as follows:

(1) The new tool should extend the GUIs for implementing OOEM OCD and OT from analysis to design with some necessary modifications to meet the specific needs in design such as more object types, more object type changes, and handling static objects without services and requests.

(2) The OOEM semantics checks for analysis should be extended through necessary modifications for static objects in design.

(3) The system should support three modeling activities (analysis, design, and redesign) as well as the transition mechanism among them. Features, which are common to the three modeling activities, will be shared while features specific to individual modeling activity should be separated.

(4) The new GUIs and any changes made to the GUIs adopted from the previous version should be well designed, being able to provide sufficient hints for interface icons and buttons and to provide necessary messages for modeling activities.
(5) This thesis work is required to give an overall assessment of the tool, especially on its efficiency, effectiveness and user-friendliness.

The first three requirements are to meet the first two criteria in the guideline for system development. The fourth requirement is to meet the third criterion. The fifth requirement is to meet the fourth criterion of the system design requirement. It is my research interest to see whether the system is able to increase modeling effectiveness by providing users with cognitive knowledge in reasoning problem domain and idea-structuring through visualizing model construction and representation, semantics checking, and OOEM rules. It is also my research interest to explore whether the decomposition and modeling transition features can make modeling more efficient.
4 SYSTEM DESIGN

This chapter introduces system design and major system components newly developed or modified from the previous version for supporting the system requirements discussed in chapter 3.

4.1 The system

The system is a graphic tool supported by a database engine. Seven drivers and two controllers interact with the database and users (Fig. 4.1). In Fig. 4.1, the octagon represents the database and the rectangles represent the system components that can be a module or a combination of several modules. Five components of the system can access the database engine and update it as well (shown as double arrows). Some attributes of the database can be updated by the five components through data input interfaces. For example, a relationship between two objects can be established and ended using the object relation generator, which generates an object detail form for entering and changing data (Fig. 4.3a). The other four components can only access the database (illustrated by single arrows). Accessing the database means that a module can input data, output data, or do both. Components that only have access to the database cannot be used to change data. For example, the mode controller is able to input and read data, while the semantic reporter can read data only. The nine components are discussed in the following sections, respectively.
The layer structure of the database shown in Fig. 4.1 represents the three working states of
the system (i.e., analysis, design, and redesign states). The mode controller controls the
working state of the database. The outward arrow from the database engine highlights the
direction of system state changes forced by the system. The tool requires that users first
build an analysis model, then build a design model on top of the analysis model, and
finally build an information system on top of the two (see section 5.2 for details). The
view controller enables the system to reverse freely the modeling order and revisit any
models that are built at different stages.

Figure 4.1 Working Chart of OOEM CASE tool
4.2 The database engine

The database engine is modified from the previous engine to implement the layer structure shown in Fig. 4.1. This section briefly introduces the modifications. The architecture of the database and the implementation of the system are described in chapter 5.

The information stored in the database can be categorized into three types: system information, object information, and graphics information. The system information includes system types and system names for different models. It is stored in a table called subsystem, which originally stores the subsystem information of an analysis model. System information plays the role of structuring the database. SystemType is a new attribute that determines the analysis, design, or redesign system functionality, while SystemID, SubsystemID, and ParentSystemID are new attributes that identify different models and their relationship (e.g., the relationship between a full system and a subsystem). The object information is stored in object, request, service, and attribute tables. The information includes names, types, and other information for all objects and their requests, services, and attributes in a model. The object and request tables are modified to include the system information for layering different models. ParentRequestID is a new attribute added into the request table for supporting decomposition and automating the transition from design to redesign (see the algorithm in section 5.4). Graphics information includes the positions and sizes of all graphics in a model.
The database is a relational database built with the PARADOX driver accompanied with Borland's Delphi.

4.3 The system generator and mode controller

The system generator and mode controller are new modules designed to work together to tune the system functionality. Before building a model, one has to select a mode based on the type of the model to be built. The mode controller is an interface controlled by the mode button in Fig. 4.2. It allows users to select a system type that will be stored in the subsystem table. The system generator is also an interface that can be activated by the NewSystem button in Fig. 4.2. It allows users to enter the system name. A corresponding SystemID is generated and stored with the system name in the subsystem table at the same time.

4.4 The graphics generator

The graphics generator is a GUI with a series of icons and buttons and a canvas designed to fully implement the OOEM OCD (Fig. 4.2). Each icon or button on the GUI controls a module to be activated. For example, the object icon can be activated to drag objects on the canvas. The modified generator can now construct and display models of different types. The drag-drop features for creating objects and requests are retained to make model construction easier.
Figure 4.2 First layer OCD of CAC Analysis (case 1)

With this generator, the object and graphics information created can be stored in a layer structure in the database, depending on what system mode and system name are selected by users. In the analysis stage, analysis and decomposition are supported. The decomposition algorithm is given in section 5.4.1. In the design stage, decomposition is supported and delegate objects and static objects can be created only in the decomposed view of a composite object. In the redesign stage, this generator can display the model.
created automatically by the redesign algorithm (see the algorithm in section 5.4.2). Fig.4.2 shows the first layer OCD of the analysis model for case 1 of this study (Appendix B).

4.5 The object relation generator

The object relation generator is a GUI designed to fully support the OOEM OT. Fig. 4.3a-c show the GUIs with the object details for the Salesman object of case 1 of this study (Appendix B). This GUI includes three interfaces (attribute, request, interface) that can be selected by clicking the three corresponding buttons on top of the GUI. For each interface, there are several windows that allow one to input and change information and to establish the relationship among attributes, services, and requests. The name window allows one to change the object name.

Fig.4.3a is the attribute interface activated by clicking the attribute button. It is for establishing the relationships between attributes and services. From the left window and the right window, a modeler can input, edit, and delete services and attributes, respectively. A dash line separates interface attribute (order) and internal attribute (customer detail and credit). The middle windows are used to establish the relationships among the services, and attributes of a selected object. By moving attributes into the middle windows using the leftward arrows from the right window, the relationship that the service in the left window will access the attributes in the right window is established.
in the database engine. This interface provides flexibility for modelers to move the attributes back to the right window in order to end the relationship. It is convenient when delegating attributes to another object in the design stage where one has to end some relationship.

Fig.4.3b shows the request interface that is used to establish the relationships between incoming requests, invoked services and the requests generated by the invoked service to complete the service. It can be done by using the left arrows to move the requests (input
from the graphics generator) from the right windows to the middle windows. Flexibility is provided to end the relationship by using the right arrows to move the requests out of the middle windows.

Finally, Fig.4.3c is the interface designed to establish and end the relationship between an incoming request and its interface attribute that triggers a service. The design of these GUIs requires the user to find services, attributes and requests and to establish their relationships, which is a course of ideas structuring.
The object relation generator is modified to function as the following. In the analysis stage, every attribute must have its corresponding request and service. In the design and the redesign stages, a warning message and an object type confirmation interface will be given to users who try to establish an object relation for an attribute without corresponding request and service. If the users confirm that this is an attribute of a static object, the object relation generator will allow users to create a specific relation for the attribute, i.e., no service is needed but a "/" must be specified as a "request". The "/" is used to represent the manipulation to the state of the static object.

Figure 4.3c Object Relation Generator
4.6 The subsystems generator

A subsystem is a system whose components are subsets of the components of another system (Wand 1996). The tool is designed to be able to divide a full (analysis, design or redesign) system into as many subsystems as wanted. The full system is the parent system of the subsystems. The generator is an interface controlled by the NewSubsystem button in Fig. 4.2. The interface is for entering the subsystem name. The corresponding SubsystemID, along with the ParentSystemID that is the SystemID of the full system are created and saved with the subsystem name in the database by the generator. Once the subsystem name is specified, the generator allows one to create the subsystem from the full system on the screen using two modules – an object exclusion module and a request elimination module. The excluded objects and eliminated requests are respectively stored in an exclusion table and a removal table so that they can be taken out from the full system whenever users want to display only the subsystem. These two modules were developed by Zhang (1998) for analysis. The new tool incorporates the subsystem generator with the mode controller for creating subsystems for all full models.

4.6.1 Object exclusion

The object exclusion module excludes an internal object from a system by performing the following operations:

- It changes its object type from internal to external.
• It removes any requests between this object and any other external objects because the external object that accepts external requests in a model violates OOEM rule #1.

• It removes the external objects that only have communication with this excluded object.

The purpose of this operation is to exclude an object from a subsystem or an information system so that it becomes an external object of the system.

4.6.2 Request elimination

The request elimination module removes either an external or an internal request from the system. It causes the following changes:

• It removes the selected request from the system. Following the request propagation, it removes those services, attributes and spawned requests that solely exist for the selected request.

• It removes those objects that no longer communicate with the system after removing the request.

This operation is usually done in concert with object exclusion or object deletion, for the purpose of generating a subsystem or an information system. Since OOEM Rule # 1 requires that only relevant objects be modeled, some irrelevant objects must be deleted or
excluded from the subsystem or the information system (Appendix A). Before deleting or excluding an object, the requests made by this object must be eliminated.

4.7 The perspective generator

The perspective generator was developed by Zhang (1997). It provides another method for model construction and visualization. OOEM defines a perspective as, “The aspects of the system necessary to describe the organizational activities invoked as a consequence of the requests of one external thing (client)” (Wand and Woo 1993). The entire OOEM model (view) is the union of perspectives for a set of given clients. Different perspectives can include different objects or different properties of the same object. A view will include all objects in the contained perspectives.

4.8 The semantics reporter

The semantics of OOEM, capturing the dynamic behaviors of a problem domain, are governed by the seven highly condensed modeling rules (Appendix A). A semantically consistent model is in compliance with the ontological principles governing OOEM.

The semantics is built into a semantic engine based on the meta-model proposed by Tan (1997). The semantics reporter has been developed in order to check the semantics of a model and report any semantic inconsistencies with detailed information on where the
inconsistencies exist and the possible reasons for them so that users can isolate the mistakes that have been made. This module has been modified so that OOEM semantics is checked not only in OOEM analysis but also in OOEM design and redesign. For static objects, only three modeling violations (2, 6, and 9) are checked (Appendix H2).

4.9 The document processor

The document processor is a module developed by Zhang (1998) to handle data exchange between a database system and a file system. When one chooses to stop a modeling activity for whatever reason, this module will read all the data in the database and then write it to a user-specified file in text format. When one decides to resume modeling, the module will recover the data to refill the database. The text format is used to store data in order to ensure portability.

4.10 The view controller

View is a modified module controlled by the view button (Fig. 4.2). It allows one to load and display different models from the database. This module finds objects based on SystemID, SubsystemID and SystemType. When a model name and a type are chosen by a user, the objects (and other information) of the model are displayed. If the model is a subsystem, view displays its parent (full) system and then "excludes" those objects not belonging to the subsystem. After displaying the model, one can continue model construction from it. Therefore, view provides a mechanism to reverse the system-forced
order of model construction. The previous view module can only apply to one analysis model and its subsystems.

4.11 Summary

The system is designed to work around a database engine modified from the previous OOEM CASE tool that can build only an analysis model and its subsystems each time. A layer structure is introduced into the database to support three modeling activities. A new attribute SystemType is designed to determine system functionality and model type. Three new attributes (i.e., SystemID, SubsystemID and ParentSystemID) play the role of identifying different models and their subsystems. Among the nine major components working with the database, the mode controller and system generator are new components and the graphics, object relation and subsystem generators, the view controller, and the semantic reporter are modified components. Table 1 summarizes the changes made to the components.

<table>
<thead>
<tr>
<th>Modules, components</th>
<th>Change</th>
<th>Descriptions of modifications</th>
<th>Section</th>
</tr>
</thead>
<tbody>
<tr>
<td>Database</td>
<td>Yes</td>
<td>Add new attributes</td>
<td>4.2</td>
</tr>
<tr>
<td>Mode controller</td>
<td>New</td>
<td></td>
<td>4.3</td>
</tr>
<tr>
<td>System generator</td>
<td>New</td>
<td></td>
<td>4.3</td>
</tr>
<tr>
<td>Graphics generator</td>
<td>Yes</td>
<td>Can create delegate and static objects</td>
<td>4.4</td>
</tr>
<tr>
<td>Object relation generator</td>
<td>Yes</td>
<td>Can establish relationship for static objects</td>
<td>4.5</td>
</tr>
<tr>
<td>Subsystem generator</td>
<td>Yes</td>
<td>Apply to all models</td>
<td>4.6</td>
</tr>
<tr>
<td>Object exclusion</td>
<td>Yes</td>
<td>Apply to redesign</td>
<td>4.6.1</td>
</tr>
<tr>
<td>Request elimination</td>
<td>Yes</td>
<td>Apply to redesign</td>
<td>4.6.2</td>
</tr>
<tr>
<td>Semantics reporter</td>
<td>Yes</td>
<td>Apply to all models</td>
<td>4.8, H2</td>
</tr>
<tr>
<td>View Controller</td>
<td>Yes</td>
<td>Apply to all models</td>
<td>4.10</td>
</tr>
</tbody>
</table>

Table 4.1 A summary of system component changes
5 System implementation

This chapter discusses the system implementation. Two algorithms for the decomposition and automation of redesign are provided.

5.1 Introduction

This system is developed on Windows 95/NT platform, using Borland Delphi 3.0 and Paradox database. Delphi is an application development tool for object-oriented Pascal, providing a comprehensive set of visual interfaces and controls. Paradox is a simple yet efficient relational database driver accompanying Delphi. The features of this package make it ideal for this system development because this system mainly uses database, visual interfaces, and controls such as buttons and forms.

5.2 OOEM procedural flow

Fig. 5.1 is a simplified procedural chart of the system implementation. Enterprise modeling starts with a new file to distinguish it from other enterprises. When a file is chosen for the enterprise, the system disables Save, View, and News subsystem controls because they are not needed, while Mode and Newsystem controls are enabled. Mode and Newsystem controls are discussed in section 4.3. An analysis mode must be chosen before choosing a new system. After analysis mode and a system name are input, both controls are disabled so that neither can change during model construction. The graphics generator and the object relation generator now have the analysis functionality (see
If Mode = Analysis & Newsystem = True then
Disable view NewSys. Save
Enable Close

Save analysis system info.

Graphics Generator & Object Relation Generator

If Close = True then
Enable all controls

To Close

Object & request deletion
Object exclusion

Save redesign system info

If Mode = Redesign, Newsys. = True
Then Disable view NewSys. Save
Enable Close, & redesign operation

If Newsubsystem = True
Disable View, Newsys. Save

Subsystem generator

To close

Save design system info

Graphics Generator & Object Relation Generator

To close

Fig. 5.1 Simplified Procedural Chart of System Implementation
sections 4.4 and 4.5). When one decides to close modeling, five main choices can be made: save (model), view (system), check semantic correctness, new subsystem, or move to system design.

Save is a procedure allowing you to call the document generator to transfer your database information into a file without closing the modeling activity (i.e. after saving the model, one can continue the modeling). If there are no objects on the screen, or if all objects on the screen are not linked to requests, the system does not create a file for the model. Otherwise, the system saves the linked objects to a file and omits the unlinked objects because they are not part of the system.

View allows one to activate the view controller. The view controller works only when both system and system type are correctly selected (see section 4.10).

Semantic correctness can be checked any time during the model construction. However, it is better to do so after a version of the model is completed because an incomplete model will result in many semantic errors.

A new subsystem can be constructed by activating the NewSubsystem button which controls the subsystem generator. During subsystem creation, no object deletions or additions are allowed. Viewing and creating new systems are also disallowed.
System design can be based on a full analysis model or its subsystem(s). After selecting the design mode and giving the design system a name, the analysis model on the screen (and in computer memory) becomes independent of the analysis in the database. A ParentSystemID is used in the database to specify that the original analysis in the database has become the parent system of the design model on the screen. The design functionality of the graphics generator and object relation generator is activated. There are five options after choosing to close design: save, semantic checks, view, newsubsystem, and redesign. Save allows one to save models. For static objects, the semantic checks only check three violations (Appendix H2). The system does not prevent one from creating subsystem(s) in the design stage. View allows revisiting the analysis and its subsystems at this stage.

After finishing the formation of delegate objects and derivation of static objects, one can start designing an information system based on the delegate and static objects in the Redesign stage. After selecting a Redesign mode and filling in a model name, the design model in the memory becomes independent of the model in the database. The ParentSystemID specifies that the original design model in the database has become the parent system of the model automatically generated by the redesign module. The Redesign module can automatically bring together all the low-level objects of the design model in the memory and automatically links them. The algorithm for the module is given in section 5.4.2. Object deletion, and exclusion and request elimination are allowed in the creation of an information system.
Figure 5.2 The Entity Relation Diagram for the database of OOEM CASE Tool

5.3 Database architecture

The database engine is a relational database based on the ER diagram in Fig. 5.2. The main difference between this engine and the previous one (Zhang, 1998, p.52) is that it includes the System and Mode entities and corresponding new attributes (SystemID,
SystemType, ParentSystemID, and SubsystemID). As described in chapter 4 and in the previous section, the system information is incorporated into object information to structure the database and layer the modeling activities. For example, the objects of an analysis model will carry an analysis system type and a name of the model. When the model is displayed, a query will find those objects with the specified system type and name. The analysis model is the parent system of a design model whose objects will carry a design system type and name. The objects of the design also carry a ParentSystemID which is the SystemID of the analysis.

Object information (object, request, service, and attribute) includes two new object types (delegate and static) in design. Analysis cannot create delegate and static objects by using the tool. For a design or a redesign model, delegate and static objects can be created and displayed with colors only when a correct SystemType is specified through the Mode control. ParentRequestID is new object information that is added to search the request and response of an upper level request/response pair if the pair is split in the lower level. It is an important attribute for the redesign module and for decomposing a request (see next section).

5.4 Algorithms

This version of OOEM CASE tool is built on top of the previous version. Most of algorithms have been introduced by Zhang (1998). Two major modules (the graphics generator and the semantics reporter) modified in this tool are discussed in Appendix H since they are not the main research results of this study. In this section, two algorithms,
the decomposition and redesign, are provided. Decomposition was originally developed by Zhang (1998) and was further completed by this study. The following section will also address the redesign algorithm which is the main result of this study.

5.4.1 Algorithm for decomposing an object

Decomposition, controlled by clicking the object to be decomposed, is an operation that allows users to decompose the object as several component objects in its decomposed view. The essence of design of this operation is that all the objects communicating with this object in the upper level should appear, in its decomposed view, as external objects communicating with its component objects. No objects or requests from the upper level should be lost in the lower level. The attributes and the services of the object can be input manually to its individual component objects in the lower level. There are two reasons for designing the tool this way: (1) It is easier to code the tool if the services and attributes are input into individual component objects by users; (2) If the upper level requests can be brought to a lower level as discussed below, the integrity among services, attributes, and requests will not be broken by the manual entries of the services and attributes.

The algorithm includes two steps: (1) From the database, the tool looks for all the objects that communicate with the decomposed object and stores them as new objects in the database. Objects that communicate with the decomposed object are treated as external
objects in the decomposed view. They may change types in the lower level and so are created as new objects. The tool then looks for all the requests that the object receives and sends, and then saves the requests in an array. (2) The tool then displays the stored objects as external objects (no services and attributes) in the decomposed view with their header colored. Users can now create those component objects of the decomposed object using the graphics generator. When creating requests, the upper level requests stored in the array are provided in a request-input form for linking those external objects with those newly created component objects. Users can split the upper level request-response pairs into two separate requests by using the request-input form as well. The selected requests linking the external objects and the component objects are stored in the database as new requests. The upper level requests in the array, once used to create the new requests, will become parent requests of the new requests and will not appear in the request-input form. This design ensures integrity because no request can be lost or added. No attributes and services can be lost because no requests are lost. Otherwise the semantics check reporter will report errors. These features were developed by Zhang (1998).

In the new tool, several features were added: (1) In the previous version, the semantics reporter checks both the external objects and the component objects in the lower level. Since the external objects have no attributes and services in the decomposed view, errors always exist for them. In the new tool, these upper level objects are identified from the database and are not checked by the semantics reporter; (2) A new attribute called ParentRequest is introduced. When the new requests linking the external objects and the
component objects are established, their ParentRequest IDs are set to be the IDs of the upper level requests in the database. This allows the system to find the lower level requests based on their upper level requests in the redesign module (see next section); (3) The delegate objects and static objects can be created only in the decomposed view.

The decomposition algorithm manages the composite object and its component objects as follows:

1. After decomposing a selected object, it becomes a composite object and carries an *Is of Composite* index (true) in run time. A dash line box is added to the outside of its original box (see the Financial Office object in Fig. 4.2). All objects in the lower level will carry an *Is Component of* index in the database. This index is the decomposed object’s object ID. If an object is not a composite object, this index is given a special value (i.e., -1) in the database.

2. When the tool displays a model, the *Is of Composite* index of all composite objects will be given a value (e.g. true) in run time. Users can visit a composite object’s decomposed view any time by clicking it. The filter technique provided by the Paradox driver enables the tool to filter out objects belonging to the composite object (external and component objects) from the database and display them. The filter criteria is the *Is Component of* index in the database. Fig. 5.3 shows the low level objects of the Financial Office object where Financial Accountant, Payroll Accountant, Warehouse Person and Salesman all hold the Financial Office’s object ID as their *Is Component of* index.
(3) An aggregation control button is activated after entering a decomposed view so that users can go back to the upper level. The algorithm checks whether the current object ID is a special value (i.e., -1) so that it can determine whether or not the button should be activated. When in a decomposed view, the current object ID is the decomposed object's ID. Thus, the button is activated.

![Object Oriented Enterprise Modeling - Analysis Mode](image)

Figure 5.3 Decomposition of the Financial Office of the CAC Case

(4) For those component objects in a decomposed view, users can decompose them further. However, those external objects from the upper level cannot be decomposed
again. For example, the Financial Accountant in Fig. 5.3 can be decomposed further, but the Salesman cannot. Salesman in the financial office view is an upper layer object.

5.4.2 Algorithm for identifying and linking objects needed in the redesign

Redesign algorithm (Fig. 5.4) involves identifying composite objects and their component objects. This is because the delegate and static objects can be created only in the decomposed view of composite objects. The algorithm also involves identifying the requests made between component objects and external objects (from the upper level) in the decomposed views of the composite object found. It also involves bringing up the component objects and linking them in a new model for information system design.

```
find all composite objects in a design model;
for each composite object found do
    find all the requests sent from the object;
end
delete all objects on screen;
draw all component objects for found composite objects; //including upper level objects as shown in Fig.5.5
for each request found do
    finds senders’ and the receivers’ IDs; //there are one or two senders and receivers depending on types in Fig.5.5
    remember them;
end
if the request meets one decomposition type (b-e)
    make new request(s) from found senders to found receivers based on the selected request-response pair ;
    delete upper level objects and requests;
end;
```

Figure 5.4 Algorithm for identifying and linking objects needed for redesign
When an object is decomposed, a request/response pair to the original object might now be sent to or obtained by any of its components. This gives rise to four possible decomposition patterns (Fig. 5.5).

Fig. 5.5 shows how the component objects of object A communicate with object B in its decomposed view (left panel) and how the component objects of object B communicate with object A in its decomposed view (right panel): (a) presents the original situation where A makes a request R of B and receives a response Re from B. In their decomposed views, A and B are shown as external objects. When the component objects of A and B are created and linked by users in their decomposed views in the design model, the four possible outcomes are: (b) depicts the request-response pair (R/Re) as not being split in both decomposed views; (c) presents the situation where the request-response pair is split in both decomposed views. Type (d) and type (e) present that R/Re as being split in only one object.

As long as users have specified the request (R/Re) as one of the four cases shown in Fig. 5.5 in the lower level of A and B, the algorithm can automatically link the component objects in the two decomposed views and display them when the redesign module is called. When the component objects are brought together, for case (b), A1 will make a request (R) of and receive a response (Re) from B1. For case (c), object A1 will make a request (R) of B1, while A2 will receive a request (Re) from B2. For case (d), A1 will send R to B1 but receive Re from B2.
Figure 5.5 Different decomposition patterns in two decomposed views of object A (left panel) and object B (right panel). (a) represents A and B in upper level where R/Re is a request-response pair. A1, A2, B1 and B2 are component objects.
Fig. 5.6 and Fig. 5.7 show the Salesman and Financial Office of the design model for case 1. Case 1 falls into the situation (b) discussed above because the "sales request" is not split in both the Financial Office object and the Salesman object.

Figure 5.6 Design for the Financial Office of the CAC Case
Figure 5.7 Design for the Salesman of the CAC Case

Fig. 5.8 shows the component objects for case 1 after they are brought up from the lower level to a new model. The IS are linked to give an information system with three IS components (e.g., the Payroll system, the Sales system and the Customer system). Some objects, such as the Customer, the Inventory and Warehouse objects can be deleted from Fig. 5.8 if one wants to show only the information system and its environment. A "/" is used to represent a request for the static object (Record Cabinet). "/" indicates no request
and response for this object. When one needs to add a request, this " / " allows one to click on it to pop up an interface to input the request name.

Figure 5.8 Redesign of CAC case

5.5 Summary

The system is implemented around a database. The database design has enabled the whole system to complete the requirements discussed in Chapter 3 -- the tool supports
three modeling activities, decomposition, modeling transition from analysis to design, automation of transition from design to redesign, and semantic checks.

The mode controller and the system generator determine system functionality and modeling activities. The view controller allows one to visit any models built at different stages of modeling and modify them.

The system retains the advantage of the previous version in modeling visualization and semantic integrity. Different colors are used to distinguish delegate objects, static objects, and external objects in decomposed view, from the usual internal objects and external objects. This helps modelers to visualize the modeling activities. The semantic checks for analysis are applied to decomposition, design, and redesign to ensure model correctness.

The redesign feature of the system is developed in a way that a modeler need not link the lower lever objects of different composite objects manually. It is designed to ensure the integrity of transition from design to redesign of information system, to reduce the potential heavy workload for an enterprise model with many objects, and to be easy to use and user-friendly.

The subsystem feature allows one to manage complex models, while the document processor is used to save models for later use.
6 **Empirical Study**

One important step for prototyping a system is to obtain user feedback during the course of system development. A user study is very valuable in examining whether the system has met the objectives of system design. In addition, user study identifies system weaknesses for later developers to improve the system.

6.1 **Method**

This study compared the system with the traditional pen and paper method by modeling two business cases. The relative strengths/weaknesses of each method were measured via a questionnaire (Appendix D). The previous study mainly tested the system features for OOEM analysis and graphics operations (Zhang, 1998), but this study was designed to measure the following new features developed for the new tool:

(1) OOEM design

(2) Decomposition

(3) Modeling transition

For each of these three features, a few hypotheses related to system usefulness (i.e., efficiency and effectiveness), user friendliness, and modeling/design capability were designed to test whether the system met the requirements discussed in chapter 3. This study also re-examined a few issues relating to OOEM analysis (i.e., 1-2), graphics operations (i.e., 11), and general perceptions of the tool (i.e., 7-8) because the new system included OOEM analysis and inherited the graphics operations from the previous system.
Besides the differences in features measured, a main difference between the two studies was that the second study could directly compare the system to pen and paper method. Comparison could provide a more accurate measurement for the above features, since testers now had a reference to justify their answers to the questionnaire. For example, when evaluating graphics operations (e.g., moving an object) the answer to Zhang's statement 6 (similar to the statement 11 of current study) might be biased because testers did not have a scale to measure how easy the operation was. By comparing with pen and paper method, now the testers could measure the relative ease of moving an object with the tool, because to move an object with the pen and paper method they had to erase the old object and draw a new one. The result from this study should therefore be more reliable. The models built with different approaches could also provide evidence to support the answers given by users.

Due to limited resources available for this study, a small number of subjects were used. Consequently, this study sought important evidence about the system, not statistically significant results. To achieve this goal, the questionnaire provided extra space for every question, allowing the subjects to make their comments, and an interview was given so that they could comment on other issues not covered in the questionnaire.

---

5 Though it is anticipated that choosing small number of subjects might reduce the significance of the study, past research reveals that a pool of four to five subjects was adequate for identifying major issues (Egan, etc 1990), indicating the adequateness of this choice.
Subjects with similar knowledge about OOEM modeling were chosen in order to ensure internal validity. Since two cases (case 1 in Appendix B and case 2 in Appendix C) were used in the study (one for the system and the other for pen and paper method), the experiment was conducted in two groups with four subjects each. For the first group, case 1 was modeled by two participants with the tool and case 2 was modeled by the other two subjects with pen and paper. For the second group, case 1 was modeled by two subjects with pen and paper and case 2 was modeled by the other two subjects with the tool. The two groups received the same amount of training and followed the same experiment steps which will be described in the next section. This design was used to ensure external validity by minimizing the learning effect of the participants and the effect caused by the difference in case difficulty.

6.2 Experiment

This experiment took about four hours and was conducted in a group of 8 MBA students who specialized in MIS and had taken a course on object-oriented enterprise modeling and other modeling methods.

The questionnaire in Appendix E was emailed to these students to collect their other background information including familiarity of computer, comfort with windowing environments, and experience in using CASE tools in conducting object-oriented analysis and design.
The following were the five consecutive steps for the experiment.

**Step One**

All subjects signed a Consent Form (Appendix F) in 1-2 minutes.

**Step Two**

The first half of this step took twenty five to thirty minutes to introduce the OOEM methodology, especially the ideas of delegate object and static object. It took another twenty five to thirty minutes to show the subjects a simple example using the tool. The tutorial introduced the graphic features provided by the system, including the creation of objects and requests and the manipulation of graphics such as resizing and moving. It emphasized how to decompose an object, change system modes among analysis, design and redesign modes, and finally do semantic checks. The purpose of this demonstration was to provide the subjects with basic understanding of the system so that they could use and evaluate the strengths and weakness of the system in the experiment.

**Step Three**

After the system demonstration, half of the 4 subjects modeled one business case (case 1 or 2 as described in last section) with the system, while the other half was asked to model the other business case with pen and paper. No question about the cases and the system
were answered at this stage. A description of features of the system was provided for quick reference. Paper and pen were provided for the subjects to write down whatever they wanted, which was to avoid that the subjects might forget some problems they found during the course of their modeling activities. The task was given one hour and fifteen minutes to complete. The exact time to finish the task was recorded for reference. The subjects were encouraged but not required to complete the experiment. After finishing the experiment, the two groups shifted roles.

**Step Four**

After completing both cases, every subject filled the questionnaire in Appendix D about twenty minutes.

**Step Five**

A short interview was conducted for identifying more system problems beyond the questions asked and for seeking solutions to those problems from the subjects. Some general questions (Appendix G) were prepared in case that the subjects would narrow their findings to some specific areas of interest. This final step had no time limit.

An incentive plan was used for the study. Every subject was paid $10 per hour as a research assistant. The testers finishing with the best solutions to the cases were given a bonus of $50 dollars each.
6.3 Subject Background

On average, the eight subjects had 5.6 years of experience in Windows 3.1 and 2.1 years in Windows 95/NT. Every subject had more than three-year experience with windowing operation systems and at least one-year experience in Windows 95/NT. All subjects felt comfortable to use Windows. Six people did not think they needed assistance for using a computer and two felt neutral to the question implying that they might need some help sometimes. Every subject used graphic tools to draw graphics at least twice per year and in average six times per year.

None of them had taken any course or training in systems analysis other than BAIT 506 which is course teaching OOEM and other modeling methods, neither had they worked as a systems analyst or similar role.

In conclusion, the subjects had very similar background that met the need of the experiment.

6.4 Results Analysis

Since one of the main purposes of this study is to test user’s satisfaction with some major system features, the result analysis is grouped according to the system requirements
(usefulness and friendliness) listed in Chapter 3 with modeling/design capability being separated from the system usefulness.

6.4.1 System Usefulness

The construct for measuring system usefulness can be divided into three parts. Part one includes questions 1-5 measuring the system efficiency and easiness. Table 6.1a summarizes the responses from the subjects to part 1 of the construct.

In general, subjects agreed that the system was efficient comparing to paper and pen, especially in transforming model activities from one state to another. The system, however, had a few problems in building analysis model. The main concerns included its slowness, and inefficient graphic operations. The comments for question 1 and the post-study interviews confirmed the findings by Zhang (1998) that request layout by the system was week, that is:

- The arrow bar of a request was changed to system default layout any time one move the objects sending or receiving the request.
- Request line was too thin to move.
- Request line could not connect to the top or bottom side of an object.
- Request/Response name could not be separated into multiple lines if it was too long.
- Request/response name could not adhere to the request line.
- Request arrow bars were overlapped when there were more than one request.

Font for request/response could not be changed

<table>
<thead>
<tr>
<th>Number</th>
<th>Question</th>
<th>Response</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>It was faster to build an enterprise model with the CASE tool than it was to build it I would with pen and paper</td>
<td>Agree 3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Neutral 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Disagree 4</td>
</tr>
<tr>
<td>3</td>
<td>It was faster to transform an enterprise model to a logical design of an information system with the CASE tool than I would with pen and paper</td>
<td>Agree 6</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Neutral 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Disagree 1</td>
</tr>
<tr>
<td>5</td>
<td>It was faster to transform an enterprise model with delegate and static objects into a design of an information system with the CASE tool than I would with pen and paper</td>
<td>Agree 7</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Neutral 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Disagree 0</td>
</tr>
<tr>
<td>2</td>
<td>It was easier to make changes to the enterprise model with the CASE tool than I would with pen and paper</td>
<td>Agree 4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Neutral 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Disagree 4</td>
</tr>
<tr>
<td>4</td>
<td>It was easier to make changes to a logical design of an information system with the CASE tool than I would with pen and paper</td>
<td>Agree 4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Neutral 2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Disagree 2</td>
</tr>
</tbody>
</table>

Table 6.1a Summary for questions on system usefulness

Since the transformations from analysis to design and then from design to redesign did not need to retype any text and re-draw the graphics, it made the modeling with the system faster than with paper and pen where everything needs to be redone.

The subjects generally felt that making changes with the system was easy. Comments made on question 2 and 4 indicated that subjects disagreed to the statements because they felt that deleting a service or an attribute was not very easy after their relation had been established. It was particularly difficult to delete a request made to an object that had been decomposed. The first dissatisfaction seemed to be partially because of
unfamiliarity with the system, which might be reduced when modelers become experienced. The second dissatisfaction was caused by the constraint of the OOEM rules incorporated into the system. A request appeared in different levels must be deleted from all levels if one wants to delete it from the upper level. Though it is possible to delete it automatically from top down, so far the system does do support this operation. The following are some findings from the subjects' comments and interviews:

1. A copy feature and an undo feature are necessary for the system to be more efficient. Since text input is the heaviest work in the modeling, text copying can reduce workload.

2. When decomposing an object, retyping the services and attributes of the object in the lower level is inefficient.

3. A request text tends to be long. Putting a request and response pair together makes the text even longer. It is difficult to arrange it when a model has many objects.

4. The system should be able to automatically delete a request from upper level to lower level.

Whether or not the system is easy to use for logical design is the key issue of this study. The second part of the construct seeks answers to this question (Table 6.1b). Obviously, the system is quite successful in this aspect. Among the reasons why the subjects held the positive attitude, visualization, interfaces, and object delegation have equal impacts, while the semantic check played more important role than the others did. The subjects felt that semantics check enabled them to find missing information or mistakes step by step from analysis to logical design. There were no comments on neutral attitudes toward
Using the CASE tool, it was generally easy to design an information system from an enterprise model, because of the visualization of business domain and the propagation of request. It might be because it became less impressive if they had followed the object detail interface requirements to input information and used the semantics check to trace missing information, all of which resulted in less mistakes.

Using the CASE tool, I felt more confident in applying the object-oriented method to information system design than using pen and paper. Using the CASE tool, I was able to build an enterprise model and design an information system of greater size and complexity than I might be able with just pen and paper.

The last part of the construct tested the general feel about the system (Table 6.1c). After using the system, almost all subjects gained confidence from the tool either in analysis...
and design with OOEM or in dealing with greater and more complicated problem. Note that though some subjects did not agree statement 1, but they did agree statement 8, reflecting the fact that the system is more useful for larger system.

6.4.2 User friendliness

User friendliness is measured by question 9-15 (Table 6.2). The subjects generally felt that the hints and messages prompted were sufficient for them. No comments were made on why the system did not provide sufficient hints for icons. Most of them agreed that the resizing and moving features provided them with greater flexibility for design. It was not a surprising outcome because information system design needed a lot of moving and

<table>
<thead>
<tr>
<th>Number</th>
<th>Question</th>
<th>Response</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>The CASE tool provided sufficient hints for icons, buttons and so on that are essential to my design.</td>
<td>Agree</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Neutral</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Disagree</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>10</td>
<td>The CASE tool provided sufficient information to correct inappropriate activities and incorrect inputs I made in design.</td>
<td>Agree</td>
</tr>
<tr>
<td></td>
<td></td>
<td>8</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Neutral</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Disagree</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>11</td>
<td>The graphical operations provided by the CASE tool (e.g. resizing, moving) made my design activities generally more efficient and effective than with pen and paper.</td>
<td>Agree</td>
</tr>
<tr>
<td></td>
<td></td>
<td>6</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Neutral</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Disagree</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>12</td>
<td>Using view, it was generally easy to review my modeling/design activities and to correct my mistake.</td>
<td>Agree</td>
</tr>
<tr>
<td></td>
<td></td>
<td>6</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Neutral</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Disagree</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>13</td>
<td>It was generally easy to change among the three modes (analysis, design or redesign) to do modeling/design/redesign work.</td>
<td>Agree</td>
</tr>
<tr>
<td></td>
<td></td>
<td>8</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Neutral</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Disagree</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>14</td>
<td>It was easy to create, specify and recognize a delegate object with the CASE tool.</td>
<td>Agree</td>
</tr>
<tr>
<td></td>
<td></td>
<td>8</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Neutral</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Disagree</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>15</td>
<td>It was generally easy to create, specify and recognize a static object with the CASE tool.</td>
<td>Agree</td>
</tr>
<tr>
<td></td>
<td></td>
<td>8</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Neutral</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Disagree</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
</tr>
</tbody>
</table>

Table 6.2 Summary of questions on user friendliness

<table>
<thead>
<tr>
<th>Number</th>
<th>Question</th>
<th>Response</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>It was generally easy to create, specify and recognize a static object with the CASE tool.</td>
<td>Agree</td>
</tr>
<tr>
<td></td>
<td></td>
<td>8</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Neutral</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Disagree</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
</tr>
</tbody>
</table>
resizing. With the tool, it became much easier. However, as mentioned above, some subjects felt that moving objects caused request lines and texts to either overlap or change shapes, which reduced the efficiency and effectiveness of design. Model transitions of the system were easy, which allowed the subjects to visit different models back and forth and make modifications. Creating and specifying a delegate object or a static object were also easy. The objects colored differently from other objects made them easy to be recognized.

6.4.3 Modeling/Design capability

The modeling/design capability construct included two subsets of questions. The first one (question 16) was devoted to the decomposition feature of the system because it is one of the most important functionality of the system. The second set of questions (question 17-...
18) was designed for object delegation and static object derivation, which are the core concepts of the OOEM logical design.

All the subjects agreed that the decomposition that allowed them to analyze in-depth the businesses was successful. Semantics check helped them to find out missing inputs, especially when they were not familiar with the cases. They perceived that when a business was more complicated, this feature would play an even important role. Two subjects oppose statements 16.3 because it was slow to go back and forth in between two levels.

Every subject felt that object delegation and static object derivation provided a systematic way of designing information systems. Some of them score these two questions lower scales because they felt that after delegating objects, changing the services, attributes, and requests of delegator could be time consuming. There must be a better way to deal with this problem.

6.5 Discussion

To prove that the above results are meaningful, two things can be investigated: time used and the models built by the two methods. However, the resultant models might be affected by the two methods or the understanding of the OOEM methodology or even by OOEM itself. Resultant models can be assessed by completeness and correctness. Completeness means that subjects had gone through all modeling activities (analysis,
design and redesign) and that all the system features that were measured in the questionnaire have been used though their models are less perfect. Though correctness is a good measure, especially for evaluating how well were the resultant models built by using the system, it is difficult to do.

Though testers can stop modeling at any stage of this experiment, no one stopped until time was up. For modeling with paper and pen, all subjects used about a half-hour more than the time planned. Two of them did not complete redesign. But they did reason about the information system design and highlighted the information system components in the design model. It implied that more time was required if they were willing to or forced to redesign the information system. One subject (not in the above two subjects) used some shortcuts like “the attributes here are the same as those in the analysis model”. More time was required if complete attribute inputs were entered. Since all of them have thought and gone through the model activities, their modeling is complete and their answers were valuable. Because of time limit, the resulting paper and pen models were not neat. Some of them were hard to read. In contrast, everyone using the system could finish modeling in the designed time period. In conclusion, using the system was found to be more efficient and effective than using paper and pen method. This conclusion is consistent with the findings in section 6.4.

As to correctness, two types of mistakes were found in the resultant models. One was missing information (e.g., service, attribute, or request). This happened in every model built with pen and paper. But it did not happen in the models built with the system.
because of semantic checks. However, a missing object (e.g., Warehouse Person in case 1) was found in models built with both the system and the paper and pen methods. The request made from the Warehouse Person to the Financial Office to send the shipment slip to complete a sales request was found to be made from the Financial Office to the Warehouse Person in the models built with both methods. As mentioned above, these problems resulted from the misunderstanding of the case and OOEM and not from the tool. Correctness checks also provided evidence that the comparison method has provided more accurate and reliable results. For example, without the comparison, it was difficult to justify the reliability of the answers to the statements 6.3 and 16.2 about semantics checks. Now, since there was missing information in the models built with the pen and paper method and there were not similar errors in the models built with the tool, the favorable answers to the feature of semantics checks of the tool were reliable.

No significant differences were found when comparing the resulting models started from different methods and different cases. It indicated that the learning effect was small and the effect associated with case difficulty was small for the experiment. Therefore, the results of this experiment were reliable.
Summary and Conclusion

This thesis is a new effort in the direction of developing an Object-Oriented Enterprise Modeling (OOEM) methodology. The objective is to develop an enhanced OOEM CASE tool with the capability for OOEM logical design of information systems.

The new system should support OOEM design, including OOEM redesign and the transition among modeling activities (i.e., analysis, design, and redesign), without losing the merits of the previous system. The system built must be useful, user-friendly, and capable of implementing the underlying methodology.

To achieve the above requirements, the system was designed based on the idea that the system features for OOEM analysis should be retained in the analysis stage. If the features for analysis are common to all modeling activities, they were shared by design and redesign. The new features specific to design and redesign were added to the system by modifying the original system modules and did not appear in the analysis. To do so, the original database engine was modified by introducing a layer structure supported by four new attributes (i.e. SystemID, SystemType, SubsystemID, and ParentSystemID). The SystemType attribute determined the different functionality of the system in different modeling stages. The ParentSystemID and SystemID attributes played the role of layering different models along the direction from analysis to design and then to redesign, while the ParentSystemID and SubsystemID attributes determined the relation between a full model and its subsystem(s). A new attribute (ParentRequestID) was used to identify
the upper and lower layer requests so as to automate the transition from design to redesign. Two components, the mode controller and system generator, were newly developed by this study in order to implement the layer structure of the database and the different functionality of the system. With the support of the modified database, five components (i.e., the graphics generator, the object relation generator, the subsystem generator, the semantic reporter, and the view controller) were modified from the previous system by adding new features to support OOEM design and redesign.

The layer structure of the database also ensured that different modeling activities and transition between activities were implemented in an integrated fashion; no information should be lost when changing from analysis to design or vice versa; the design activity should not affect the analysis result and the redesign activity should not affect the design result.

This research raised an open OOEM question, i.e., when an upper layer request/response pair was split in the lower layer; the response could become a request that demanded an interface attribute that did not exist in the upper level. This research further investigated different possible splitting of a request/response pair and developed an algorithm to automate the transition from design to redesign.

A comparison study was conducted to examine the system usefulness (e.g., efficiency and effectiveness), user friendliness, and the modeling/design capability of the new system for supporting OOEM design, redesign, decomposition, and modeling transition.
Eight testers used this system to successfully model two business cases and compared the system with the pen and paper method. This comparison provided a direct and more accurate way to test the new system. For example, the feedback from the testers indicated that this tool was more useful for modeling big and complicated business enterprises than the pen and paper method. The resultant models from both methods also provided information to verify the findings from the tester's feedback. Through this empirical study a list of questions was compiled for future research (section 7.2). Some suggestions for improving the tool were provided in section 7.3.

7.1 Contributions

This research has achieved its purpose of building an OOEM CASE tool with the capability of doing high-level OOEM design of information systems. The new system has proven by empirical study to be useful and user friendly. It is helpful for users to gain confidence in applying OOEM analysis and design methodology to model a system with greater size and complexity. The system also provides a systematic way of analyzing in-depth business activities through decomposition, and a systematic way of doing design and redesign of information system through object delegation and derivation of static objects. The system is user-friendly because it provides hints and messages to guide users in how to use the tool and to correct errors in the course of modeling.

The system increases the efficiency and effectiveness of OOEM design of information systems in several ways. Efficiency is realized by visualizing model construction and
problem domain, and by extending semantics checks to design; it is also achieved by easing modeling activities including graphic operations, model review and modifications, and transition between different modeling activities. All of these reduce the resources required for design.

Effectiveness is realized by extending the semantic checks of the OOEM rules to OOEM design and redesign of information systems. With this tool, the modeling transition from design to redesign is automated, which reduces the chance of making mistakes in OOEM design of information systems.

This study successfully conducted an empirical study that gave a more complete picture of the tool, which will be helpful for further development of OOEM CASE tool either by enhancing the current tool or by redeveloping it.

The successful development of the CASE tool is a move towards achieving information system development from top down. This success indicates that further effort put forth in the same direction will be promising and that the CASE tool possesses great potential for practical use. The modified database engine presents the opportunity for new developers to further develop the tool towards implementation phase such as code generation, since it has linked three modeling activities together.
7.2 Future Research

Though the new system is quite successful, some problems are quite obvious and are of immediate interest of research as well:

1) System slowness – it is caused by many factors, for example,
   • Heavy database querying to ensure referential integrity
   • Need to frequently repaint OOEM OCDs

2) Request layout is a graphical problem that reduces the efficiency and effectiveness of the system. Current study confirms this problem found by Zhang (1998).

3) Repeated typing for lower level attributes and services should be eliminated because this will increase efficiency and effectiveness. An algorithm is needed to delete upper level request that has been carried to the lower level.

4) More standard features such as 'undo' and 'copy' should be developed so that greater efficiency can be achieved.

5) When a request/response pair appears at lower level of a decomposed view, a response can become a request made to an upper level object or from an upper level object to a component object (Fig. 5.5). It requires that we input an attribute that does not exist in the
upper level. For example, in Fig.5.5 (c), A2 needs to have an attribute for response /B which does not exist in the upper level. This is not entirely consistent. So far, a modeler needs to realize this difference, i.e., a response at upper level is stored in database but it is not an attribute. However, at the lower level, this response might become a request, which needs to have an attribute for the receiving object. There is a close linkage between the response information and the new attribute in the database. The current system cannot handle this, as it is an open question about OOEM. Further study of OOEM is needed.

6) More comprehensive empirical study is important before further development. In particular, it is important to get feedback from programmer who can provide information about how to improve the system technical point of view (e.g., system and GUI design) and about what they will need from the system for implementation such as code generation, database design and development, group work, and communication between different systems.

7.3 Suggestions for Improvements

Here are some suggestions to each of the questions (1-4) in section 7.2 to improve the tool:

(1) The two problems in future research question (1) (e.g., heavy querying and repainting) call for redesigning the system architecture. Though current approach that
all modules communicate with the database eases system coding, it sacrifices speed of the system because the I/Os needed for the modules to access data are usually much slower than other processes. Since an enterprise system needs not to take much space to store information, using memory instead of disk to store frequently used data is possible. The problem of repainting CODs can be overcome by using multiple-forms technique (i.e., several forms can be used to display several models) and concurrent processes. Techniques such as multi-threads, tree, and linked list are required to do so. Other factors causing system slowness are the recursive algorithm and semantics check. These two problems are less critical than the first two because they slow down the system less frequently. Zhang (1998) proposed to find a non-recursive algorithm for solving the first problem. If the system use the structure proposed above, these two problems can be resolved together with the first two problems because recursion can be avoided.

(2) There are many possible solutions to research question (2) (that is, graphic layout); one solution is to modify the current code. Since half of the code (about 7500 lines) is devoted to graphics operations, it is worth thinking of other choice – rather than developing graphics for the system, the system can be built on top of some packages with well-developed graphics options that need little programming to use.

(3) Future research question (3) (that is repeated typing) is relative easy to solve because this problem is caused by the decision on how to build the tool. Since searching for attributes and services for a composite object is easy, they can be put into object
detail form for decomposition. When users create a component object, they can be all input in its detailed form so that users can delete those not belonging to it. Deletion is much easier than retyping all of them.

(4) If the system architecture is changed to use memory, question (4) (adding undo and copy features) may be easier to resolve.
References


Appendix A. OOEM Rules and Meta-model

OOEM rules are the most important semantics needed to be incorporated into the OOEM CASE tool (Wand and Woo, 1996). They become programmable with Tan’s meta-model (Tan, 1997, p25). Here show the OOEM rules and the meta-model for your references (Fig. A).

Rule #1: The Scope of Identification Rule

This rule defines the boundary of interest in the candidate model. It emphasizes the separation of activities and events between those within the sphere of interest and influence (the enterprise) and those without (the environment). Objects outside the system are said to generate Requests to the system. The system then becomes unstable until the requests are satisfied by the events that take place within the system.

The metamodel encapsulates this rule by requiring that objects be assigned to either Internal or External (object) roles. Once assigned roles, objects are obligated to assume their common and specialized roles as defined in the metamodel.

Rule #2: The Object Identification Rule

This rule specifies that each object must Provide or Request a service as part of system activities to be included. It states that an object is included if and only if it provides at
least one service, or makes at least one request to the system. An internal object is part of the system and provides at least one service. An external object is part of the environment that interacts with the system.

Rule #3: The Service Inclusion Rule

The rule states that a *Service* is included in the system if and only if it is directly invoked by at least one *Request*. This rule does not prevent a service to be invoked by multiple requests.

In the metamodel, this rule is enforced by the relationship *Invoke* and its cardinal specification.

Rule #4: The Attribute Inclusion Rule

The rule determines which attributes should be included in a model. To be necessarily included, an attribute must be either used or affected by at least one service, or known to at least one other object. An internal attribute distinguishes from an interface attribute by its invisibility from outside.

In the metamodel, the relationship *Access* and its cardinal specification ensures that *Internal* attributes within an objects must be accessed by at least one *Service* belonging to that object.
Rule #5: The Attribute Ownership Rule

The rule determines which object is responsible for an attribute value. It states that an attribute belongs to one and only one object. This object is known as the custodian of the attribute. The value of every attribute can only be modified or obtained by the services of its custodian object.

In the metamodel, a particular Attribute must be uniquely Own-ed by an object. This same object also Provide-s one or more Services that Access-es that attribute.

Rule #6: The Composite Object Rule

This rule determines when to include composites or component objects in the model. Here the term composite is used to refer to an aggregate object that can be decomposed into component objects in the problem domain.

The Composite Object Rule states that a composite may only be included if it provides services that are not provided by any of its component objects. Only the emergent properties may be included in a composite object.

Rule #7: The Sub-Classification Rule
This rule determines when to create a general object class. This general object class (super-class) has the common services that are provided by two or more objects (sub-classes). All common 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. The Sub-Classification Rule represents the model's generalization and specialization structure.

Figure A An OOEM Metamodel (Tan, 1997, p25).
Appendix B. Custom Aircraft Corporation

Case 1 for empirical study

Adapted from Information Systems Analysis (V. J. Hall and J. W. Mosevich, p396-404, 1988).

Custom Aircraft Corporation (CAC) produces aircraft-related products for a number of aircraft manufacturers. At the present time, the processing of order information involving the sales department, inventory control department, finance department and the warehouse are handled manually.

The order process begins when customer places an order to the salesman, accompanying with a 25% deposit required. The CAC salesman verifies the order by referring to the customer file with customer's detail (name, address, phone number) and credit. Any changes to the customer details can be updated at this time. If the customer is creditable, a sales commission notice, two verified order slips are created.

The CAC salesman makes the sale by sending the customer's deposit, the sales commission notice, and a verified order slip to the financial office, and sending the other verified order slip to the inventory control department which determines if the order can be filled from the current stock in the warehouse or must wait for manufacturing.

---

6 Example solutions for this case are in Fig.4.2, Fig.5.3, and Fig.5.5-5.8.
Financial accountants in the financial office are responsible for recording customer’s deposit, updating customer details, and charging customer accounts for the remaining 75%, by referring to a customer file with customer detail, account information, and the shipment slips delivered from the warehouse. The accountant notifies the payroll accountants when the shipment slip of the order from warehouse is received. The accountant will inform the salesman of the customer’s credit earned after the 75% on account is paid.

The payroll accountant updates the sales commission file after receiving the sales commission notice from the financial accountant. The commission amount due to each salesperson is finalized after verifying the file with the notice received from the financial accountant.

Now CAC wants to automate its order process and you are hired first to analyze the original process and then design an information system that can automate the process so that inefficiency can be avoided. CAC has decided that it will keep its original organizational structure and your design should reflect this requirement.
Appendix C. The Northwind Case

Case 2 for empirical study

Adapted from Beginning Object with Visual Basic 5 (Wright, pg. 220-223, 1998).

Northwind, a product distributor, distributes a catalogue of products sold by its suppliers, to its customers. NorthWind’s sales are made exclusively through telephone, from its order office. Products are distributed to customers from its warehouse. To be a Northwind customer, a consumer must register with and accepted by Northwind.

A customer phones a sales representative to place an order. The rep processes the order by first checking the stock sheets and the client’s detail file. If products are enough and the customer has good credit, a shipment slip is then passed to the warehouse for dispatch and the stocks are marked down. The rep also sends an acknowledgment to the customer and charges on the customer’s account when the shipment slip is marked and returned from the warehouse. The customer earns credit when the account is paid.

A consumer contacts the rep to subscribe as Northwind’s customer. The rep fills out the customer details form with provided information and open an account. The customer will receive a “Welcome to Northwind” letter and a catalogue of products. Customers can

---

7 Example solutions are given in the end of Appendix C.
phone Northwind to make changes to their details (name, address, phone number) any time.

Upon receiving the shipment slip, the warehouse person processes the order by finding, checking the condition of, packing, shipping the physical products, and then updating the stock sheets. The shipment slip is marked and then passed to the order office. If the remaining stock hits the reorder level, an order is sent to the suppliers to fill the stock vacancy. If the condition of a product is not good, the warehouse person will determine the cause of it. If it is a defect, the product will be shipped back to the supplier for an exchange.
Figure c.1 First Layer of North Wind
Figure c.2 Design of the Order Office for North Wind
Figure c.3 Design of the Warehouse for North Wind
Figure c.4 Redesign of the North Wind
Appendix D. Questionnaire for Empirical Study

Questionnaire #2
OOEM CASE Tool Study
Fall, 1998

In the questions below, please indicate the degree to which you agree with each statement. A rating of 7 indicates that you strongly agree with each statement. A rating of 1 means that you strongly disagree with the statement.

A. System Usefulness

1. It was faster to build an enterprise model with the CASE tool than it was to build it I would with pen and paper

   1   2   3   4   5   6   7

   Comment

2. It was easier to make changes to the enterprise model with the CASE tool than I would with pen and paper

   1   2   3   4   5   6   7

   Comment

3. It was faster to transform an enterprise model to a logical design of an information system with the CASE tool than I would with pen and paper

   1   2   3   4   5   6   7

   Comment

4. It was easier to make changes to a logical design of an information system with the CASE tool than I would with pen and paper

   Comment

5. It was faster to transform an enterprise model with delegate and static objects into a design of an information system with the CASE tool than I would with pen and paper

   1   2   3   4   5   6   7

   Comment

6. Using the CASE tool, it was generally easy to design an information system from an enterprise model
because of visualization of business domain

6.2 propagation of request

6.3 semantic check

6.4 interfaces

6.5 object delegation

6.6 please specify and rate any other reason -

7. Using the CASE tool, I felt more confident in applying the object-oriented method to information system design than using pen and paper.

8. Using the CASE tool, I was able to build an enterprise model and design an information system of greater size and complexity than I might be able with just pen and paper.
**B. User friendliness**

9. The CASE tool provided sufficient hints for icons, buttons and so on that are essential to my design.

   1  2  3  4  5  6  7

Comment

10. The CASE tool provided sufficient information to correct inappropriate activities and incorrect inputs I made in design.

   1  2  3  4  5  6  7

Comment

11. The graphical operations provided by the CASE tool (e.g. resizing, moving) made my design activities generally more efficient and effective than with pen and paper.

   1  2  3  4  5  6  7

Comment

12. Using view, it was generally easy to review my modeling/design activities and to correct my mistake.

   1  2  3  4  5  6  7

Comment

13. It was generally easy to change among the three modes (analysis, design or redesign) to do modeling/design/redesign work.

   1  2  3  4  5  6  7

Comment

14. It was easy to create, specify and recognize a delegate object with the CASE tool.

   1  2  3  4  5  6  7

Comment

15. It was generally easy to create, specify and recognize a static object with the CASE tool.

   1  2  3  4  5  6  7

Comment
C. Modeling/Design Capability

16. The decomposition feature improved the CASE tool with capacity to model/design more complicated business activities

   1 2 3 4 5 6 7
Comment

because

16.1 it provided a systematic way of analyzing in-depth business activities

   1 2 3 4 5 6 7
Comment

16.2 it provided semantic check for correctness

   1 2 3 4 5 6 7
Comment

16.3 it was generally easy to go through and review different layers of business activities

   1 2 3 4 5 6 7
Comment

16.4 please specify and rate any other reason

   1 2 3 4 5 6 7
Comment

17. By supporting object delegation and static object derivation, the CASE tool provided a systematic way of conducting logical design of an information system.

   1 2 3 4 5 6 7
Comment

18. The CASE tool provided a systematic way of linking delegate objects to form an information system design

   1 2 3 4 5 6 7
Comment
Appendix E. Subject Background Questionnaire

Questionnaire #1
OOEM CASE Tool Study
Fall, 1998

Name: (Please Print) ____________________________________________

Please circle your choice where appropriate.

Have you ever used a windowing system e.g. Apple MacIntosh, Microsoft Windows?

Yes / No (if No, go directly to the next question)

If yes, which one(s) and for how many years?
(please circle and write length of use)

<table>
<thead>
<tr>
<th>System</th>
<th>Years</th>
</tr>
</thead>
<tbody>
<tr>
<td>MS Windows 3.x</td>
<td></td>
</tr>
<tr>
<td>MS Windows 95/NT</td>
<td></td>
</tr>
<tr>
<td>Apple MacIntosh</td>
<td></td>
</tr>
<tr>
<td>X-Windows</td>
<td></td>
</tr>
<tr>
<td>Other</td>
<td></td>
</tr>
</tbody>
</table>

How comfortable are you with using a windowing system?

Very Uncomfortable 1 2 3 Neutral 4 5 Very Comfortable 6 7

I always need an experienced person to assist me when I use a computer.

Strongly Disagree 1 2 3 Neutral 4 5 Strongly Agree 6 7

Have you used graphic tools (e.g. Microsoft Office Drawing, Apple MacIntosh KaleidaGraph) to draw graphics?

Yes / No (if No, go directly to the next question)

If yes, for how often? ________ (times/per yr)
Have you taken any course or training in systems analysis or related areas other than BAIT 506?

Yes / No (if No, go directly to the next question)

If yes, at what level (Undergraduate/Graduate/Professional/Other) ________________?

How long was the course/training ________________ (months)?

Did you use CASE tools in the course/training?

Yes / No (if No, go directly to the next question)

If yes, please specify ________________

Have you ever worked as a systems analyst or similar role?

Yes / No

If yes, for how many years? ________________ yrs

Did you use CASE tools in your work?

Yes / No

If yes, please specify ________________
Appendix F. Consent Form

OOEM CASE Tool Study

Fall 1998

Thank you for participating in this study.

We are seeking your help in evaluating OOEM CASE Tool, an Enhanced Version of the Object-Oriented Enterprise Modeling Computer-Aided Software Engineering Tool developed in the Management Information Systems Division at UBC. OOEM CASE tool is aimed at helping analysts to build enterprise models of target organizations and helping analyst/designers to automate part of the systems in a systematic and semantically consistent way.

Your contribution lies in your participation in testing the system and providing your feedback that we collect for empirical study. The course of this study might be videotaped entirely or partially. Videotaping is solely for review purpose and will not be used for any other purposes without your prior consent in writing. The result of this study may play an important role in subsequent changes or improvements to the system.

Please keep in mind that this study is to assess the system, not you. In fact, result of this study will be aggregated and your responses cannot be traced back to you. So we would appreciate your frank opinion.
Once the session is over, you are requested not to discuss the study with your friends and classmates.

The estimated time for this study is four hours. Participation in this study is completely voluntary and you may discontinue your participation at any time during the session.

Participant Acknowledgement

I have read and understood the contents of above terms and conditions and agree to participate in the study.

_________________________  _________________________  ___________
Name (Please Print)        Signature                 Date
Appendix G. Post Study Interview Record

OOEM CASE Tool Study
Fall 1998

Name: (Please Print) ____________________________

How do you feel about this modeling tool compared to other CASE tools/graphic
drawing tool that you can do system analysis and design?

__________________________________________________________________________

__________________________________________________________________________

What are the strengths of this system?

__________________________________________________________________________

__________________________________________________________________________

What are the weaknesses of this system?

__________________________________________________________________________

__________________________________________________________________________

What are the specific areas that need to be improved immediately?

__________________________________________________________________________

__________________________________________________________________________

Please include here any other comments that you want to make.

__________________________________________________________________________
Appendix H. Algorithms for graphics generator and semantics reporter

H1. Algorithm for graphics generator

The graphics generator developed mainly by Zhang (1998) and modified by this study, allows a modeler to build an OCD or display a model. It also provides a set of operations such as insertion, modification and deletion of model constructs. Figure H1 is the pseudocode for the graphics generator. The code has omitted some operations like insertion etc.

The algorithm is executed when modeling starts or a view is called. After that, one can work directly on the GUI through operations such as moving the position of an existing object, adding new objects, or changing the name of an existing request, etc. These changes will be saved to the database before the model is closed.

```
if view is selected then
  if mode is correctly selected then
    select all objects belong to a model from database;
    draw and link them
  end
end

if object button is selected then
  pop up object input interface with different type options depending on mode;
  input object type and name and save them to database;
  draw it with a default size;
end

if request button is selected then
  pop up a request interface for inputting request and response names and for specifying the sending and the receiving object names;
  draw a arrow from the sending object to the receiving object and display the request and response name;
  save the request name, response name, object names, position and lines;
end
```

Figure H1. Algorithm for graphics generator
H2. Algorithm for semantic check

The semantics check reporter, based on the meta-model by Tan (1997), examines ten modeling errors violating four of the OOEM rules. Each of the errors is checked by querying the database with one SQL query and is reported through the semantic reporter as one of the ten errors listed below. The OOEM rule that each error violates is included in the parentheses for reference. These ten modeling errors have been applied to decomposition, design, and redesign in the new tool.

1. The request sent to an internal object does not invoke any service. (Rule #1)
2. The request sent to an internal object modifies no interface attribute. (Rule #1)
3. The request from an external object is sent to another external object. (Rule #1)
4. The request from an internal object is not spawned by any service. (Rule #1)
5. The service spawns the same request more than once. (Rule #1)
6. This object has no communication with any other object. (Rule #2)
7. The service is not invoked by any request. (Rule #3)
8. The interface attribute is not accessed by any service. (Rule #4)
9. The interface attribute is not modified by any request. (Rule #4)
10. The internal attribute is not accessed by any service. (Rule #4)

The algorithm identifies which object should be checked and then check them for some or all of the ten errors. For static object, only 2, 6, and 9 are checked. There must be an
owner object for every error message whose nametag will be painted red if any of the checks are not passed. Semantic report is available for every problematic object.

The Attribute Ownership Rule (Rule #5) is enforced by the database design. Since Rules #6 and #7 deal with the structure aspect of a model, they are not checked by the system.