INFORMATION TO USERS

The most advanced technology has been used to photograph and reproduce this manuscript from the microfilm master. UMI films the original text directly from the copy submitted. Thus, some dissertation copies are in typewriter face, while others may be from a computer printer.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyrighted material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each oversize page is available as one exposure on a standard 35 mm slide or as a 17" × 23" black and white photographic print for an additional charge.

Photographs included in the original manuscript have been reproduced xerographically in this copy. 35 mm slides or 6" × 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.
The integration of organization and information system modeling: A metasystem approach to the generation of group decision support systems and computer-aided software engineering

Chen, Minder, Ph.D.
The University of Arizona, 1988

Copyright ©1988 by Chen, Minder. All rights reserved.
PLEASE NOTE:

In all cases this material has been filmed in the best possible way from the available copy. Problems encountered with this document have been identified here with a check mark \( \checkmark \).

1. Glossy photographs or pages
2. Colored illustrations, paper or print
3. Photographs with dark background
4. Illustrations are poor copy
5. Pages with black marks, not original copy
6. Print shows through as there is text on both sides of page
7. Indistinct, broken or small print on several pages \( \checkmark \)
8. Print exceeds margin requirements
9. Tightly bound copy with print lost in spine
10. Computer printout pages with indistinct print
11. Page(s) lacking when material received, and not available from school or author.
12. Page(s) seem to be missing in numbering only as text follows.
13. Two pages numbered. Text follows.
14. Curling and wrinkled pages
15. Dissertation contains pages with print at a slant, filmed as received \( \checkmark \)
16. Other

________________________________________________________

________________________________________________________

________________________________________________________

________________________________________________________
THE INTEGRATION OF ORGANIZATION AND INFORMATION SYSTEM MODELING: A METASYSTEM APPROACH TO THE GENERATION OF GROUP DECISION SUPPORT SYSTEMS AND COMPUTER-AIDED SOFTWARE ENGINEERING

by

Minder Chen

Copyright © Minder Chen 1988

A Dissertation Submitted to the COMMITTEE ON BUSINESS ADMINISTRATION In Partial Fulfillment of the Requirements For the Degree of DOCTOR OF PHILOSOPHY In the Graduate College THE UNIVERSITY OF ARIZONA 1988
As members of the Final Examination Committee, we certify that we have read the dissertation prepared by Minder Chen entitled THE INTEGRATION OF ORGANIZATION AND INFORMATION SYSTEM MODELING: A METASYSTEM APPROACH TO THE GENERATION OF GROUP DECISION SUPPORT SYSTEMS AND COMPUTER-AIDED SOFTWARE ENGINEERING and recommend that it be accepted as fulfilling the dissertation requirement for the Degree of Doctor of Philosophy.

Final approval and acceptance of this dissertation is contingent upon the candidate's submission of the final copy of the dissertation to the Graduate College.

I hereby certify that I have read this dissertation prepared under my direction and recommend that it be accepted as fulfilling the dissertation requirement.

Dissertation Director Jay F. Nunamaker, Jr.
STATEMENT BY AUTHOR

This dissertation has been submitted in partial fulfillment of requirements for an advanced degree at The University of Arizona and is deposited in the University Library to be made available to borrowers under the rules of the Library.

Brief quotations from this dissertation are allowable with special permission, provided that accurate acknowledgement of source is made. Requests for permission for extended quotation from or reproduction of this manuscript in whole or in part may be granted by the copyright holder.

SIGNED: Minder Chen
ACKNOWLEDGEMENT

Writing a dissertation is like a long journey. It took me almost two years to search for a good research topic, to formulate a conceptual framework for putting all the pieces together, to develop a prototype system for justifying the design, and finally to write this dissertation describing what I have learned in this research. I am lucky enough to have had many guides to help me prepare for the expedition and to give me direction when I reached a crossroad. Professor Martin Yang, at National Chiao-Tung University, taught me what MIS is all about and encouraged me to pursue a Ph.D. degree. Dr. Ralph Martinez introduced me to the layered approach to building complex systems. Dr. Bernard Zeigler's introduction to the object-oriented approach was very helpful. Dr. Martinez and Dr. Zeigler are on my minor committee. Dr. Douglas Vogel's experiences and knowledge in group decision support systems research and Plexsys Planning Tools helped me tremendously in applying group decision support systems to systems development. Dr. E. Sue Weber brought me into the world of human cognition and helped me understand the importance of user interface design. Dr. Joey George, who sat in for Dr. Weber who was ill, gave me very constructive suggestions. Dr. Jay Nunamaker, my dissertation director, introduced me to the rationale of PSL/PSA, one of his brain children, and shared with me his enthusiasm for systems development. His vision of the possibility of designing a description language to describe organizations as well as information systems became the core concept of this dissertation. He also provided me with an excellent working environment and financial support so that I could concentrate on my research. I would like to thank Dr. Vogel, Dr. Weber, Dr. George, and Dr. Nunamaker for serving as my dissertation major committee members.

Many other teachers and friends have helped me in this journey: Dr. Benn Kon-synski, Dr. Olivia Sheng, Dr. Lynda Applegate, Chandra Amaravadi, Sergio Davalos, Kunihiko Higa, Mari Heltne, Ben Martz, Conrad Royksund, Joe Valacich, Ajay Vinze, Bill Kelly, Kendall Cliff, Betty Albert, Irene Hop, Joan Seidel, Jo Cento, ....

My special thanks go to my lovely wife Yi-Hwa Irene Liou Chen, who has taken care of the family so well while I devoted myself to the research. She has had to be a housewife, a Ph.D. student, and a mother at the same time. Amazingly, she did it all very well, far better than I could have done in her position. The coming of Justin Gejune Chen, my first son, has been the most important motivation for me to finish my dissertation in addition to bringing me so many joyful moments.

Finally, I would like to express my appreciation to my parents Mr. Lian-Chuan Chen and Mrs. Jing-Jen Wang Chen for providing me with opportunities for a good education and a sweet home in which to grow up. Finishing this dissertation makes me feel I am home again after a long journey. It's time to relax for a while, to be with Irene and Justin, to recharge, and then to start a new adventure.
# Table of Contents

List of Figures .............................................. 11  
List of Tables ............................................. 14  
Abstract .................................................... 16  

1 Introduction ............................................. 17  

1.1 Motivation .............................................. 18  
1.2 Research Questions .................................... 20  
1.3 Research Scope and Constraints ..................... 22  
1.4 Overview ................................................. 23  

2 Review of Literature ................................... 24  

2.1 System Development: Lifecycle Model and Enabling Technologies ... 24  
2.2 Computer-Aided Software Engineering Tools .......... 27  
\hspace{1cm} 2.2.1 A Case Study of CASE: PSL/PSA .......... 27  
\hspace{1cm} 2.2.2 Current Status and Future Trends of CASE Tools .......... 29  
2.3 Computer-Supported Collaborative Work .............. 33  
2.4 Human Factors in the System Development .......... 37  
2.5 The Object-Oriented Approach and Its Applications .. 38  
\hspace{1cm} 2.5.1 Characteristics of Object-Oriented Systems .......... 39  
\hspace{1cm} 2.5.2 Application Areas of the Object-Oriented Approach .. 40
# Table of Contents – Continued

3 Research Methodology ........................................... 44

3.1 Research and Its Classification .................................. 44

3.2 System Development as a Research Methodology .............. 46

3.3 Research Domain and Research Methodology .................. 50

3.4 Process of System Development Research Methodology ........ 51

3.5 Software Engineering and Engineering .......................... 57

3.5.1 Definitions of Software Engineering ........................ 58

3.6 A Typology of Research in Software Engineering ............ 60

3.7 Research Methodologies Application ........................... 63

3.7.1 Conducting Case Studies .................................... 63

3.8 Formulating a Framework ........................................ 64

3.8.1 Building a Prototype System ................................ 68

3.8.2 Designing Empirical Tests ................................... 69

4 Concepts, Knowledge Representation, and Architecture of MetaPlex 70

4.1 Concept of a Metasystem ....................................... 70

4.1.1 Definition of Metasystem Environment ...................... 71

4.1.2 Review of Existing Metasystem Environments .............. 73

4.2 Knowledge Representation in MetaPlex ........................ 74
Table of Contents – Continued

4.2.1 The Representation of Relation Types in MetaPlex .............. 77
4.2.2 Abstraction in MetaPlex ........................................ 80

4.3 Formal Definitions of MetaPlex System Description Languages and System Descriptions ........................................ 81
  4.3.1 A Formal Definition of MetaPlex System Description Languages ........................................ 82
  4.3.2 A Formal Definition of MetaPlex System Descriptions .......... 82

4.4 A Comparison of the MetaPlex Representation with Other Knowledge Representation Schemes ................................. 85
  4.4.1 Semantic Networks ................................................. 86
  4.4.2 Frames ............................................................... 87
  4.4.3 Rules ............................................................... 88

4.5 An Integrated Systems Development Environment: A Layered Approach 90

4.6 The Architecture of MetaPlex ........................................... 93

4.7 User Interface Design .................................................. 96
  4.7.1 Window-based ..................................................... 96
  4.7.2 Menu-driven ..................................................... 97
  4.7.3 Graphical Interface and Mouse .................................. 99
  4.7.4 User Interface Utilities ........................................ 99
  4.7.5 Modeless Operations ........................................... 100
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.7.6</td>
<td>On-line operation</td>
<td>100</td>
</tr>
<tr>
<td>5</td>
<td>Design and Implementation of MetaPlex</td>
<td>101</td>
</tr>
<tr>
<td>5.1</td>
<td>Language Definition System</td>
<td>101</td>
</tr>
<tr>
<td>5.1.1</td>
<td>Managing Language Definitions</td>
<td>102</td>
</tr>
<tr>
<td>5.1.2</td>
<td>Defining System Description Languages</td>
<td>104</td>
</tr>
<tr>
<td>5.1.3</td>
<td>Creating Sublanguages</td>
<td>113</td>
</tr>
<tr>
<td>5.2</td>
<td>System Description System</td>
<td>114</td>
</tr>
<tr>
<td>5.2.1</td>
<td>Managing Target System Description</td>
<td>115</td>
</tr>
<tr>
<td>5.2.2</td>
<td>Defining System Descriptions</td>
<td>117</td>
</tr>
<tr>
<td>5.2.3</td>
<td>Report and On-line Query Facilities</td>
<td>124</td>
</tr>
<tr>
<td>5.2.4</td>
<td>Query Function</td>
<td>125</td>
</tr>
<tr>
<td>5.2.5</td>
<td>Structure Browser for Relations</td>
<td>126</td>
</tr>
<tr>
<td>5.2.6</td>
<td>Using Rules to Determine Object Types</td>
<td>128</td>
</tr>
<tr>
<td>5.3</td>
<td>Transformation System</td>
<td>130</td>
</tr>
<tr>
<td>5.3.1</td>
<td>The Necessity of a Transformation System</td>
<td>130</td>
</tr>
<tr>
<td>5.3.2</td>
<td>The Architecture of the Transformation System</td>
<td>131</td>
</tr>
<tr>
<td>5.3.3</td>
<td>Transformation Languages</td>
<td>131</td>
</tr>
<tr>
<td>5.3.4</td>
<td>The Design of the Transformation System</td>
<td>134</td>
</tr>
</tbody>
</table>
Table of Contents – Continued

5.3.5 The Design of System Description Transformation System ... 138
5.4 Consistency and Completeness Checking in MetaPlex ............... 139
  5.4.1 Consistency Checking ..................................... 139
  5.4.2 Completeness Checking .................................... 140
5.5 Implementation Issues ........................................... 144
  5.5.1 The Selection of Software Platform for Implementation .......... 146
  5.5.2 Limitation of the Implementation Environment ................ 148
5.6 Issues in Using MetaPlex ....................................... 148
  5.6.1 Methods in Defining MetaPlex Languages .................... 148
  5.6.2 Selection of System Development Methods .................... 152

6 Integration of GDSS and CASE: A Metasystem Approach 154
  6.1 System Development As a Collaborative Effort .................. 154
  6.2 A Method for Collaborative System Development: JAD ........... 157
  6.3 Collaborative System Development Tools ........................ 158
  6.4 A Comparison Between GDSS and CASE .......................... 160
  6.5 Using Plexsys Planning Tools for System Development Meetings ... 162
  6.6 An Example of Generating GDSS Tools from MetaPlex: Structured Electronic Brainstorming System .................. 164
Table of Contents – Continued

6.7 A Metasystem Approach to the Generation of Computer-Supported Collaborative Work Tools ........................................ 168

6.8 Defining a System Description Language for Information Systems Planning: Critical Success Factors .................................. 170

6.9 Using MetaPlex to Generate Integrated GDSS and CASE Environments ................................................................. 172

7 Conclusion ............................................................................................................................................................................ 177

7.1 The Contributions of the Dissertation ......................................................................................................................... 177

7.1.1 Contributions to the Metasystem Approach .............................................................................................................. 178

7.1.2 Contributions to CASE and GDSS Areas ...................................................................................................................... 179

7.1.3 Contribution to the System Development Research Methodology ............................................................................... 179

7.2 Future Research ................................................................................................................................................................ 180

7.2.1 Extensions to MetaPlex ............................................................................................................................................... 180

7.2.2 Validation of MetaPlex ............................................................................................................................................... 181

7.3 Final Remarks ................................................................................................................................................................ 182

Appendix A: A Sample Language Syntax Report .................................................................................................................. 183

Appendix B: A Sample System Description Report ............................................................................................................. 185

List of References ................................................................................................................................................................... 188
## List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>A Lifecycle Model and Accompanying Enabling Technologies</td>
<td>25</td>
</tr>
<tr>
<td>2.2</td>
<td>The System Architecture of PSL/PSA on the IBM Mainframe</td>
<td>30</td>
</tr>
<tr>
<td>2.3</td>
<td>Three Representation Formats to Interface with Users</td>
<td>31</td>
</tr>
<tr>
<td>3.1</td>
<td>Research Domains and Research Methodologies</td>
<td>52</td>
</tr>
<tr>
<td>3.2</td>
<td>The Process of System Development Research</td>
<td>54</td>
</tr>
<tr>
<td>3.3</td>
<td>A Model of Requirement Definition for a System</td>
<td>66</td>
</tr>
<tr>
<td>4.1</td>
<td>MetaPlex Knowledge Representation Model</td>
<td>76</td>
</tr>
<tr>
<td>4.2</td>
<td>The Cross Referencing of a Relation Type</td>
<td>78</td>
</tr>
<tr>
<td>4.3</td>
<td>An Integrated Systems Development Environment: A Layered Approach</td>
<td>91</td>
</tr>
<tr>
<td>4.4</td>
<td>The Architecture of MetaPlex</td>
<td>94</td>
</tr>
<tr>
<td>4.5</td>
<td>An Example of a Hierarchy Menu</td>
<td>98</td>
</tr>
<tr>
<td>5.1</td>
<td>MetaPlex Language Browser</td>
<td>102</td>
</tr>
<tr>
<td>5.2</td>
<td>The Layout of the Language Editor Layout</td>
<td>105</td>
</tr>
</tbody>
</table>
List of Figures – Continued

5.3 The Sample Screen Dump of the Language Editor .................. 106
5.4 Attribute Type Browser ............................................. 112
5.5 Target System Browser ............................................. 116
5.6 The Layout of the Target System Editor ............................. 118
5.7 Target System Editor .............................................. 119
5.8 Attribute Browser .................................................. 123
5.9 Structure Browser .................................................. 127
5.10 The Architecture of the Transformation System .................. 132
5.11 Transformation Language Editor ................................. 135
5.12 Menu-Driven Dialogs for Defining an Object Transformation Rule ..... 137
5.13 A Completeness Checking Report of a System Description ........ 143
5.14 Menu-Driven Dialogs for On-Line Completeness Checking ........ 145
6.1 The Use of Individual-Based CASE Tools .......................... 156
6.2 Representation Model of Plexsys Planning Tools: Process and Data ... 165
6.3 The Effect of a Metasystem Approach to the Computer-Supported Collaborative Work Tools ............................................. 169
6.4 A Representation Model of Critical Success Factors Method ........ 172
List of Figures – Continued

6.5 The Integration of GDSS and CASE .......................... 174

6.6 The Use of a GDSS as a Front End to CASE Tools: A Metasystem Approach .......................................................... 176
List of Tables

2.1 The Application Areas of the Object-Oriented Approach .......................... 40

3.1 Examples of the Contribution of System Development to Research Areas 47

4.1 Metasystems in Various Application Domains ................................. 71

4.2 A Formal Definition of MetaPlex System Description Languages .......... 83

4.3 A Formal Definition of MetaPlex System Descriptions ....................... 84

4.4 The Most Common Links Used in Semantic Nets ................................. 86

4.5 An Example of Using a Frame to Represent a Program Module .............. 88

5.1 An Example of an Attribute Type Definition ................................... 111

5.2 Sample Rules for Determining the Object Types of an Object in a DFD Language ................................................................. 129

5.3 An Example of Object Transformation Rule .................................... 133

5.4 An Example of Relation Transformation Rule ................................. 134

5.5 An Example Rule for Consistency Checking .................................... 140
List of Tables – Continued

5.6  Examples of Completeness Checking Rules of a Data Flow Diagram  .  142

5.7  Sample Relation Types for Structured System Analysis . . . . . . . . . . . 150
ABSTRACT

Information systems have become an essential part of every business organization's production and management process. It is critical to an enterprise to integrate its organization and information systems. However, the lack of computer-supported tools for modeling organization and information systems has put their integration far beyond our reach. In this research, a metasystem approach that can integrate organization and information system modeling by means of group decision support systems (GDSS) and computer-aided software engineering (CASE) has been proposed. A prototype system, called MetaPlex, has been designed and implemented to demonstrate the feasibility of the proposed approach. The emphasis in design and implementation of MetaPlex has been on making the underlying knowledge representation expressive enough to meet modeling requirements and ensuring that the user interface is easy for managers and users to use.

The use of a GDSS makes it possible to capture strategic assumptions and business objectives, as well as structures of an organization, from managers through face-to-face group meetings. The application of the metasystem concepts in generating GDSS tools makes the customization of a GDSS environment possible. Because a GDSS environment driven by a metasystem can be used to acquire information about a target system from multiple experts in a structured format that can be integrated with CASE tools, this approach provides a basis for a seamless integration of GDSS and CASE tools to support both organization and information system modeling.
Chapter 1

Introduction

Information systems have become an essential part of every business organization's production and management process. Software and its development have become strategically important to senior managers of an organization [Arthur Andersen & Co., 1987]. A reported two to four year average backlog in application systems development of corporations [Martin, 1985] suggests that many companies are losing the advantage of using information system technology to compete with others.

To use information technologies effectively, issues must be addressed from the top. At the corporate level, executive managers should determine the effects of information systems on the corporation by first reviewing the cost of software development, the reliance of the management on information systems, and the role of information systems in the products and processes of the business. Based on their understanding of the effects that information systems have on their business, managers may be prompted to initiate a company-wide program to increase software productivity, but it is more important that managers align their IS technologies with their business strategies so that strategic information systems can be developed to keep their company competitive [Benson and Parker, 1986].

Since an information processing system is designed to serve a management system
(i.e., the organization, its objectives, the individuals or groups in it, and the rules and procedures by which it operates) [Teichroew and Sayani, 1971], top managers’ focus should be on such issues as how information can be managed as a shared resource and how information systems can be used in a more effective and strategic way [IBM, 1984]. This emphasis acknowledges the extent to which information and knowledge are important company assets. The need for integration of an organization with its information systems has been recognized as one of major issues in the MIS community [Benson and Parker, 1986]. In this research the parts of an organization system that are of interest are its structures, goals, assumptions, etc. The system developed for the research tries to capture the model of an organization at a very high level and focuses mainly on activities in early phases of the system development process.

1.1 Motivation

After a review of the literature, the author discovered that tools that are adequate to support the modeling of organization systems and information systems are currently not available. This lack of automated support for the modeling of organization and information systems may be a major reason why these systems are not tightly coupled.

Information systems are used to support management in an organization (e.g., organization systems). The development of an effective information system relies on an adequate understanding of the organization it is to serve. The model of an organization actually resides in the combined perceptions of its users and managers, consisting of a collection of objectives, assumptions, beliefs, etc., held by these individuals. Collaboration among managers and users therefore is critical to the success of constructing an integrated organization model. Group decision support systems (GDSS) provide a mechanism to facilitate group members’ interactions in such a way as to capture a more realistic picture of an organization system and the information system it needs. However,
although GDSS have been used in strategic planning and information system planning, information about an organization model acquired from GDSS sessions have seldom been used for information systems development. This is because most GDSS tools do not support appropriate functions to capture the structured information needed in the information system development process.

The development of a large and complicated information system is a group activity, but existing Computer-Aided Software Engineering (CASE) tools do not address the issue of group collaboration and deliberation in a system development project [Lasky, 1988]. Generally speaking, research dealing with constructing large software systems and managing large project teams (so called programming-in-the-large and programming-in-the-many) most often focuses on the coordination of programming projects where version and configuration controls are of great concern [Dart, et al., 1987]. The next generation of CASE tools should help deliberative groups deal effectively with the issues of collaboration in the upstream activities of the system development process.

Problems that the author tried to tackle in this research can be summarized as follows:

1. It is very difficult for a group of users, managers, and IS developers to define requirements for complex information systems.

2. Information systems usually are not integrated with the organization systems in which they reside.

3. Information systems that are developed often do not work for managers and users the way they were expected to.

4. The high cost of developing and using CASE tools discourages their use by an organization.
5. Current CASE tools do not directly support face-to-face interactions among project members in the system development process.

6. Current GDSS tools are not appropriate for support of systems development project meetings.

1.2 Research Questions

Given that it is increasingly obvious that integrating high level organizational strategies and goals with information systems is critical to the survival of a business in a dynamic and competitive environment, one emerging trend is toward the use of Group Decision Support Systems (GDSS) to facilitate interaction at a high level of intellectual argumentation among the people within an organization. The results of such interactions usually become the strategies adopted by the organization and the goals it sets for the future. Another trend is toward the use of Computer-Aided Software Engineering (CASE) tools to support the activities involved in system analysis, design, implementation, and maintenance. However, the need to integrate organization and information system modeling has not been appropriately addressed by previous research related to GDSS and CASE.

The basic research question of this dissertation is:

How can we build a flexible and integrated system development environment to facilitate organization and information system modeling at both individual and group levels?

This research question can be refined into the following subquestions:

1. What is the basic framework of a computer-supported system development environment?
The author has sought to determine what information is required to create a system description and has proposed a taxonomy for computer-aided software engineering environments for organization and information systems modeling. This framework is presented in Section 3.8.

2. What kind of system architecture will be useful in building an integrated system development environment?
A layered architecture which identifies the functionalities of an integrated development environment at various levels is proposed in Section 4.5. A metasystem, called MetaPlex, provides the kernel (i.e., the two innermost layers) of the system architecture. The architecture of MetaPlex is presented in Section 4.4.

3. What are the design and implementation issues involved in the development of a metasystem?
The design and implementation issues of a flexible metasystem have been major focuses for this research. The detailed design of three major subsystems of MetaPlex is discussed from Section 5.1 to Section 5.3. These three subsystems are a language definition system, a system description system, and a transformation system. Issues in implementing and using MetaPlex are addressed in Section 5.5 and Section 5.6.

4. How can a metasystem approach be used to integrate the modeling of organization and information systems?
A framework has been proposed for the integration of organization and information systems modeling. A metasystem approach is used to generate a GDSS environment and CASE tools. The GDSS environment is used to capture the organization model by facilitating interactions of groups while CASE tools are used to represent the analysis and design of an information system. This proposed approach is based on a comparison of GDSS and CASE. An example of how to use a metasystem (i.e., MetaPlex) to generate a GDSS environment is given in Section 6.6. A rep-
presentation model of a method for organization and information system modeling is discussed in Section 6.8 in the form of a case that demonstrate the feasibility of using a metasystem to integrate organization and information systems modeling.

1.3 Research Scope and Constraints

This research is confined to the development of computer support for the system life cycle from business planning to information system analysis and design. Support for the implementation of a software system is not included for several reasons: 1) Most system implementation tools are machine dependent, 2) The use of fourth-generation languages and application generators has shown promise of being able to alleviate some implementation problems, and 3) It costs more to fix errors that have occurred in the early stage of the life cycle [Boehm, 1976]. It is most useful to build computer-supported tools that will prevent errors in the early stages of systems development process.

In this dissertation the term system development refers to all the activities in a lifecycle of either an organization system or an information system. When the term information system development is used, the focus is on the early activities, such as information system planning, analysis, and design. The term organization system encompasses the structures, objectives, strategies, assumptions, beliefs, and other high level functions of organizations. The storage and manipulation of operational transaction processing data of an organization are not the focus of this research.

Instead of using a complicated knowledge representation for the metasystem, which might increase the complexity of using the system, a simple but useful knowledge representation scheme based on the extended Entity-Relation model [Chen, 1976; Teorey, Yang, and Fry, 1986] is used for defining system description languages.

Smalltalk-AT [SoftSmarts, 1987] is used for the implementation of a prototype
system, called MetaPlex. MetaPlex stands for "a Metasystem for Plexsys". Some of the functionalities have been designed, but have not yet been implemented. However, the prototype system has demonstrated the feasibility of the architecture and detailed design presented in this dissertation. The issue of storing a large volume of system descriptions at secondary memory is not included in this research. However, researchers in object-oriented database systems have suggested solutions related to this problem.

1.4 Overview

The major objective of this research has been to explore the possibility of integrating organization and information systems modeling through a metasystem approach. The author first addresses the need for an integrated software development environment to support the integration of organization and information system modeling through a literature review in Chapter 2. Research methodology issues are addressed in Chapter 3. The focus of the research is then on the architecture and design of a metasystem used in developing such an integrated software environment. Knowledge representation and architecture design issues are presented in Chapter 4. In Chapter 5, the design and implementation issues of the kernel of such an architecture, i.e., MetaPlex, are discussed. Use of such a metasystem to support the generation of CASE tools and GDSS tools is demonstrated by cases presented in Chapter 6. This dissertation is concluded with Chapter 7, in which the contributions of this research are discussed and future research possibilities are presented.
Chapter 2

Review of Literature

This research is based on knowledge from various domains, such as research in system development methods, computer-aided software engineering (CASE) tools, human factors in analysis and design, and object-oriented systems. Literature in these domains is reviewed to serve as a solid ground for the discussion of this research. Some of the related literature is discussed under appropriate topics in later chapters of this dissertation.

2.1 System Development: Lifecycle Model and Enabling Technologies

Figure 2.1 presents a simplified system development lifecycle model in which the author identified some enabling technologies. The lifecycle model extended the traditional lifecycle model by including business planning and information system planning [King and Srinivasan, 1988]. The lifecycle model has the following phases: 1) business planning and information system planning, 2) requirements analysis, 3) logical design, 4) physical design, 5) programming and testing, and 6) operation and maintenance. Including business planning and information system planning in the system development lifecycle reflects the importance of integrating organization and information systems.

The enabling technologies listed in the same figure demonstrate the extent to which
Figure 2.1: A Lifecycle Model and Accompanying Enabling Technologies
they are involved in the various phases in the lifecycle model. The group decision support systems and requirement definition and design languages can apply to the first three stages in the lifecycle. Group decision support systems work best for the facilitation of interactions among project teams. The requirement and design languages can be used to capture the information in a structured format so that it can be used through the system lifecycle.

The CASE tools, expert systems, and reusability technologies support the phases from the requirements analysis to programming and testing. Research in CASE tools focuses on the automatic supports in analysis and design, while expert system technologies help to capture expertise to make these tools perform intelligently [Lubar and Harandi, 1986]. Reuse of software components has been shown as the most prominent approach to improving software productivity [Matsumoto, 1987]. Reusable technologies are considered to be applicable not only to programs and modules, but also to requirement definitions and design specifications [Ficher, 1987].

Software factory, fourth generation languages (4GL), application generators, and configuration and version control systems are appropriately applied in the last three phases of the system lifecycle. While application generators and 4GL relieve the system developer from having to deal with implementation details, the software factory is a systematic approach which emphasizes the quality control of system implementation. Configuration management and version control help in the coordination of a programming-in-the-large environment. According to Boehm [1987], the number of source instructions is the most significant factor in software development cost. New software development tools and techniques such as fourth-generation languages, reusable components, executable specification, and application generators can be used to reduce the number of lines of code that must be developed and thus reduce software cost.

Traditionally, the information generated from each phase of the lifecycle is stored in the system encyclopedia. An integrated system encyclopedia permits traceability across
the lifecycle phases. Through appropriate categorization, information stored in the system encyclopedia should become available for reuse. Expertise of system development that is embedded in these enabling technologies to assist system developers should also reside in the system encyclopedia. The next section discusses a literature review of CASE tools.

2.2 Computer-Aided Software Engineering Tools

Structured methodologies such as structured analysis, structured design, and structured programming, etc., have been used to help software developers manage the development process in a systematic manner. The adoption of a single method not only requires a project team to use a standard set of notations to communicate, but also imposes a constraint in developers' thinking about a target system. The use of multiple methods in the development process is necessary in order to deal with various aspects of a target system. Selection, customization, and integration of system development methods for different types of systems are a little explored but important research area. CASE technology is a combination of software tools and methodologies with a focus on the total software productivity problem [McClure, 1987].

In this section, the author will first discuss PSL/PSA as a basis for the future discussion in this dissertation. The architecture and future trends in CASE are then presented.

2.2.1 A Case Study of CASE: PSL/PSA

The development of Computer-Aided Software Engineering can be traced back to late 1960 when PSL/PSA was developed. This research is partially inspired by the concepts of PSL/PSA for which the underlying philosophy is:
1. More effort and attention should be devoted to the front end of the information system development process, where a proposed system is being described from the user's point of view.

2. The computer should be used in the development process, since the system development process itself generates large amounts of information that require computer systems' help in processing these data.

3. A computer-aided approach to system development must start with "documentation." From the software development process point of view, each stage in the life cycle will generate information to be used in the later stages in the life cycle. If a system cannot document the information generated by the system development activities, the continuity and traceability in the system development process will be lost.

PSL/PSA has the following capabilities: 1) to describe information systems, whether manual or computerized, whether existing or proposed, regardless of application areas. 2) to record such descriptions in a computerized database (i.e., a PSA database). 3) to incrementally add to, modify, or delete from a description in the database. 4) To produce documentation for use by the analyst or other types of users. The PSA database is a precursor of data dictionary systems for database management systems. It is a mechanism to integrate various system aspects in a cohesive information repository.

The system architecture of PSL/PSA as implemented on the IBM mainframe is depicted in Figure 2.2 [IBM, 1984a]. The kernel of the system architecture is still based on the original PSL/PSA architecture [Teichroew, Hershey, and Yamamoto, 1982]. The PSL/PSA system on the IBM mainframe has a front-end system written in ISPF to improve the usability of the system. GENAID uses templates to help users to prepare PSL statements. CMDAID shows all the parameters of PSA reports to help users define PSA reports. However, the PSL statements and PSA report commands are executed in
2.2.2 Current Status and Future Trends of CASE Tools

Software development has become so complex that computer systems are needed to assist in managing the complexity. Computer-Aided Software Engineering (CASE) is one major effort to increase software productivity [Case, 1985]. Most CASE tools support certain structured methods such as data flow diagram, structured design, and data modeling, etc. These software tools have been used to enforce and enhance the concepts and procedures of certain methods [McClure, 1987]. The basic components of a CASE environment are:

1. Interface components.

   Figure 2.3 represents three major formats for eliciting information from users or representing information to them. Structured statements have been used by PSL/PSA [Teichroew and Hershey, 1977] and the AutoDraw in ER Designer [Chen & Associate, 1987]. Graphical representation has been used in Excelerator [Index Technology Co., 1986] and most of the CASE tools. Matrix has been used frequently in some PSA reports for showing cross-referenced information. An interactive matrix tool which has been developed at the University of Arizona Management Information Systems Department allows users to enter cross-reference relations in a matrix format [Martz, 1988]. As Figure 2.3 shows, information captured is stored in the database component. Multiple representations of the same information will help users to interact with the system (for both input and output).

2. A database component.

   The database component of a CASE tool comes with various names, such as information repository, system encyclopedia, data dictionary, project dictionary, or even knowledge base, etc. The database component keeps track of all the information generated in various phases by various tools. Most CASE tools use specialized batch mode.
Figure 2.2: The System Architecture of PSL/PSA on the IBM Mainframe
Figure 2.3: Three Representation Formats to Interface with Users.
database systems to store information because the complexity of objects makes traditional database systems inadequate.

3. Analysis and design components.

The analysis components can check the consistency and completeness of a system description; they are involved in the development process only passively. Design components usually apply design algorithms to existing information in order to generate design specifications. For example, a database design tool can use the functional dependency among data elements to generate a database schema in normal forms [Sudha and Curran, 1988].

Computer-aided software engineering tools are tools that support the development of computer software systems. CASE tools can be categorized according to the following characteristics:

1. Types of software systems supported. CASE tools are designed to support different types of systems, such as business information systems, real-time control systems, computer system software, etc. Most CASE tools on the market are designed to support the development of business information systems.

2. Scope of system development life cycle covered. CASE tools usually cover part of the system life cycle, i.e., CASE Toolkits. CASE Toolkits that have been integrated to support the automation of the whole system life cycle are called CASE Workbenches [McClure, 1987]. Upper CASE or front-end CASE are CASE tools which emphasize the upstream activities (e.g., planning, analysis and design), while lower CASE tools emphasize the downstream activities (e.g., programming and maintenance).

3. Type of activities. A full CASE development system includes management, technical, and support environments [Butler, 1987].
4. Level of embedded intelligence. Some CASE tools possess limited "intelligence" to help system developers in the process of system development. Some other CASE tools contain encoded analysis and design expertise so that they can be used to help system developers to perform their task more intelligently [Lubar and Hara­­r and, 1986]. According to their intelligence level, CASE tools can be classified as document recorder, reminder, analyzer, transformer, intelligent guide, tutor, and co­­worker [Chen and Nunamaker, 1987]. Most CASE tools are designed to support the documentation of existing or future systems. Some tools can analyze completeness and consistency of specification. Knowledge-based technologies should make possible the creation of software environments which really support software developers in the analysis and design processes.

5. Level of integration. Some CASE tools do not talk to any other tools, while some CASE tools share the same database system. Taking full advantages of database and knowledge base technology is the key to CASE tools integration.

6. The horizontal span of a project. CASE tools can be used by individual software developers, by analysis and design teams, by project managers, or even by a company as a whole. Most CASE tools are designed for individuals to use. Tools to support groups rather than individuals are in great demand.

These characteristics can be used not only as criteria for the selection of CASE tools by CASE tools users, but also can be used by CASE tools developers for setting up objectives.

2.3 Computer-Supported Collaborative Work

"Software development is one domain in which the importance of coordination technology has been apparent for some time" [Greif, 1988, p. 251]. Coordination technology includes
electronic mail, teleconferencing, group support systems, etc. Poor communication has been recognized as a major problem in system design [Guinan and Bostrom, 1986]. In this dissertation, the author will focus on how a metasystem approach can be used to support communication in a software development project within the context of Group Decision Support Systems.

Structured and semi-structured messages or templates have been used to capture collaboration among group members in the following research:

1. Information Lens Project [Malone et al.; 1987, 1987a].
   In the Information Lens Project conducted at MIT, semi-structured message types are used as templates to categorize the contents and intentions of mail messages. Rules are used to determine the dissemination of outgoing messages or the way that incoming mail should be handled. The use of semi-structured messages in Information Lens have been demonstrated to be very useful. Unlike traditional free-style mail messages, the semi-structured messages allow automatic processing of mail messages.

2. Colab [Stefik, et al., 1986].
   Colab at Xerox PARC uses computers to support the collaborative process in face-to-face meetings. Tools developed for collaboration in Colab include Boardnoter, Cognoter, and Argnoter. Both Cognoter and Argnoter try to impose structures on information generated by groups. Cognoter supports collaborative planning for writing or presentation. A hierarchical structure is used to represent the structures of a paper or a presentation. Argnoter is an argumentation spreadsheet for proposals which provides structures to organize alternative proposals, their assumptions, and evaluation criteria. Cognoter and Argnoter not only provide structures to represent information generated in group process; they also structure the processes of various intellectual activities, i.e., generating ideas or alternatives, and then organizing and evaluating them.
3. The Coordinator [Winograd and Flores, 1986; Winograd, 1988; Carasik and Grantham, 1988].

The Coordinator is a conversational system designed to capture the pragmatics of languages used in various management functions by organizations. It is based on the conversation theory whose premise is that "every manager is primarily concerned with generating and maintaining a network of conversation for actions — conversations in which requests and commitments lead to successful completion of work" [Winograd and Flores 1986, p. 144]. Coordinator is a menu-driven system which facilitates the generation, transmission, storage, and retrieval of messages that occur during a conversation over a period of time.

One way or another, all the above-mentioned collaborative support systems use structured formats to represent the context of a conversation among group members. The conversation is facilitated by an electronic mail system (e.g., Coordinator), an electronic face-to-face meeting system (e.g., Colab), or a teleconferencing system (e.g., Rapport [Ahuja, Ensor, and Horn, 1988]). Capturing and presenting information in a structured format in computer-supported collaborative work reduces the ambiguity of the messages interchanged among members of a computer-supported group and increases the processability of these messages by computers. A too-rigid structure will prohibit freedom and creativity in expressing ideas, while no structure at all makes the automatic processing of information extremely difficult. A metasystem approach has been taken to balance the advantages and disadvantages of representation structures.

A study of various structures employed by systems in various domains tells us that there is no agreement on which is the best structure to represent information generated in group deliberation processes. Since developing a cooperative system is a major endeavor, a more flexible architecture which can support the generation of tools for collaboration in various domains and applications will be of great value. A metasystem (i.e., MetaPlex) described in Chapter 4 and 5, can be used to support collaborative work because of the
generality of its knowledge representation scheme and its ease of use. Since the main interest in this research is to support group activities in organizational planning, information systems planning, analysis and design, the discussion will focus on experiences drawn from use of the Plexsys Planning Tools [Vogel, et al., 1987] developed at the University of Arizona Management Information Systems Department. Although Plexsys is a set of groupware which supports face-to-face meetings, the concepts presented in this research can apply to other collaboration tools, such as electronic mail and teleconferencing.

Research has shown that there is a need for domain-specific coordination support [Greif, 1988, p. 251]. Winograd [1988] discusses the design of collaborative work from a language/action perspective. He points out the importance of incorporating the specialized language, or jargon, used by an organization to streamline the conversation of groups. Freeman [1988] calls for further research in developing domain specific languages (specification and design languages) while Winograd’s perspective derives from office automation research and Freeman’s from software engineering research, both have called for the use of specialized languages. Instead of developing a specialized language and tool for each domain in an ad hoc fashion, a metasystem environment allows users to define a specialized language for a domain and generates tools to support the use of that language. Using a metasystem approach to CSCW can improve both the structuring and the flexibility of the generated tools. However, the metasystem approach has received too little attention from those engaged in computer-supported collaborative work. This research develops a simple and useful metasystem environment that can be used to generate tools for both information systems development and group collaboration, i.e., CASE and GDSS.
2.4 Human Factors in the System Development

Software development is a process that involves software developers, managers and users. Collaboratively, they interact, present their views of reality, and create a model of the target system that they all agree upon. Effectiveness of group interactions in the software development process determines the success of a software project.

Besides the human factors at the software development level, there are organizational factors that determine the success of using information technologies in an organization. Since information systems usually change the way an organization processes information and knowledge, the organizational and power structures of a company will change accordingly [Franz and Robey, 1984; Saunders and Scamell, 1986]. Because of the perceived effects of a new information system upon an organization, there will be politics and social factors involved in implementing an information system. This is why the integration of organization and information systems modeling is so important. An integrated model of an organization and its information systems can help managers, users, and system developers in: 1) understanding the objectives of information systems from an organizational perspective, 2) estimating the impacts of new information systems on the organization structures, and 3) linking an organization's strategies and policies with its information system profiles.

The successful use of information systems not only depends on productivity in developing systems, but also on creating the proper organizational climate to encourage the incorporation and acceptance of innovative technologies. What organizational factors are involved in the use of information systems at various levels? How can an organization build a proper infrastructure to encourage a positive attitude receptive to the development of information systems? Research into software productivity should not neglect the impacts of organizational and human factors.

At the software projects level, Boehm found personnel and the ability to function
as a team to be the second most significant factor in software cost [Boehm, 1987]. Software development teams differ by factors of four to five in productivity and individual programmers’ productivity range can be as large as 20 to 1, or more [Boehm, 1987a]. What factors have caused the individual differences? What kinds of methods, tools, training, and team structures can be applied to increase software productivity by properly dealing with these factors? These are the kinds of questions that need to be addressed by researchers in this area.

Researchers of human factors in software development have found that the major difference between experienced and novice software developers is directly related to the extent of their respective knowledge bases. The knowledge base (i.e., the structure and content of the knowledge stored in the long-term memory) of an experienced software developer is larger, more structured, and much more flexibly indexed than an inexperienced software developer’s knowledge base. Knowledge stored is represented as rules and frames which capture the patterns for actions as well as the structures of the information systems and application domains [Curtis, 1985; Vitalari, 1985]. Recent research has demonstrated that a knowledge-based system can capture an experienced software developer’s knowledge and be used by inexperienced developers [Lubars and Harandi, 1986]. More research needs to be done in human cognition to find out what makes good system analysts and designers [Weber, 1985].

2.5 The Object-Oriented Approach and Its Applications

Object-oriented systems have different meanings to various people [Stefik and Bobrow, 1985]. However, people generally agree that an object-oriented system should have certain characteristics. In this section, general characteristics of object-oriented systems are first reviewed and then a discussion of application areas using the object-oriented approach is presented.
2.5.1 Characteristics of Object-Oriented Systems

Concepts, entities, and things are represented as objects in an object-oriented system. Objects have structures and behaviors. The static structures of an object are described by its private data (i.e., instance variables) which store the local status of the object. The dynamic behaviors of an object are simulated by instance methods of the object's class. Computations in an object-oriented system are done by sending messages among objects (i.e., indirect procedure calls). Objects interact with each other collaboratively to accomplish required tasks. A message has to be sent to an object to access or change the status of the object. This message passing paradigm is borrowed from the human communication model [Ingalls, 1981]. Objects can interface with the external world only through a set of predefined message patterns (i.e., protocol). Direct access to an object's private data is prohibited. The message passing paradigm facilitates information hiding which increases the modularity, modifiability, expandibility, and maintainability of a software system [Klahr, etc., 1986]. Different classes can share the same message pattern to support polymorphism. For example, the message rotate can be sent to instances of Square and Triangle but these objects will have different implementations of their instance methods for the message rotate. A message received will be responded to according to the message receiver's corresponding method. Polymorphism reduces the syntactical complexity of the system and allows the development of more generic codes.

Objects are organized by the class hierarchy (e.g., In Smalltalk, Circle is a subclass of Arc, Arc is a subclass of Path, and Path is a subclass of DisplayObject). An object can inherit structures (instance variables) and behaviors (methods) from its class and superclasses. Subclasses can implement a method with the same message pattern to override the generic behavior defined by their superclasses. The class hierarchy promotes both data and procedure abstraction in a software system, which makes the resulting system more reusable.
### Table 2.1: The Application Areas of the Object-Oriented Approach

<table>
<thead>
<tr>
<th>Application Areas</th>
<th>Systems or Literatures</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simulation languages</td>
<td>Simula [Dahl and Nygaard, 1966]</td>
</tr>
<tr>
<td></td>
<td>Ross [McArthur et al., 1986]</td>
</tr>
<tr>
<td>Programming languages</td>
<td>Smalltalk [Goldberg and Robson, 1983]</td>
</tr>
<tr>
<td></td>
<td>C++ [Stroustrup, 1986]</td>
</tr>
<tr>
<td>User interface design and implementation</td>
<td>MVC paradigm in Smalltalk [Krasner and Pope, 1987]</td>
</tr>
<tr>
<td></td>
<td>Impulse-88 [Smith, et al., 1988]</td>
</tr>
<tr>
<td>Database systems</td>
<td>Iris [Fishman, et al., 1987]</td>
</tr>
<tr>
<td></td>
<td>GemStone [Maier and Stein, 1987]</td>
</tr>
<tr>
<td>Knowledge base systems</td>
<td>KEE [Kunz, Kehler, and Williams, 1984].</td>
</tr>
<tr>
<td></td>
<td>NExpert [Neuron Data Inc., 1987]</td>
</tr>
<tr>
<td>Graphics and hypermedia systems</td>
<td>MacDraw [Apple Computer Inc., 1984]</td>
</tr>
<tr>
<td></td>
<td>Intermedia [Yankelovich et al., 1988]</td>
</tr>
<tr>
<td>Systems analysis and design methods</td>
<td>Botch [1986]</td>
</tr>
<tr>
<td></td>
<td>Pressman [1987]</td>
</tr>
<tr>
<td>Organization modeling</td>
<td>Blarning [1987]</td>
</tr>
</tbody>
</table>

#### 2.5.2 Application Areas of the Object-Oriented Approach

An object-oriented approach has been recognized as a unified paradigm in the design and implementation of programming languages, knowledge-based systems, databases, and human interface design [Zaniolo, etc., 1986]. Table 2.1 shows the progress of the object-oriented approach in various application areas (listed loosely in chronological order according to when the object-oriented approach was first applied):
1. Simulation. The earliest area to which the object-oriented approach has been applied is simulation, as in the Simula language [Dahl and Nyaard, 1966]. Simula is actually a general purpose language which introduces the concepts of inheritance between class and instance and the support of polymorphism. Static structures and dynamic behaviors of entities are simulated by data and procedures encapsulated in objects. Recent simulation languages, such as Ross [McArthur et al., 1986], incorporate more object-oriented features.

2. Programming language. Smalltalk is a purely object-oriented programming language environment [Goldberg and Robson, 1983]. Everything in Smalltalk is implemented as an object, e.g., classes, user interface, control structures, etc. Some object-oriented languages are hybrid, such as C++ [Stroustrup 1986] and Objective-C [Cox, 1986].

3. User interface design and implementation. An object-oriented user interface was first used experimentally in the early 1970s when the Smalltalk system used overlapped windows to view objects, a mouse to direct manipulation of objects, and a location-sensitive pop-up menu to select an appropriate operation. In Smalltalk-80, a Model-View-Controller (MVC) paradigm has been used as a framework for user interface development [Krasner and Pope, 1987]. Impulse-88, developed by Smith et al., [1987], provides very high level components to support the design and building of object-oriented user interfaces. The object-oriented approach has also been borrowed and implemented in the operating system in Apple’s Lisa and Macintosh. Object-oriented interfaces provide consistent and natural ways for users to interact with a computer system and thus help lower the user’s learning curve in adapting to a new system.

4. Graphics. Graphics tools such as MacDraw [Apple Computer Inc., 1984] is an object-oriented graphics drawing package which runs on Apple Macintosh. All the graphic objects in MacDraw can be manipulated by a common set of operations,
such as select, copy, move, and enlarge. The consistency in operations helps
users in learning how to use the system. Object-oriented techniques also make
the implementation of graphic and hypermedia applications such as Intermedia

5. Database systems. Object-oriented database systems come to the stage because of
the persistence of problems in storing large amount of data in an object-oriented
system as well as requirements to represent complex objects in CAD/CAM and
CASE applications. Iris [Fishman, et al., 1987] and GemStone [Maier and
Stone, 1987] are examples of object-oriented database systems.

6. Knowledge base systems. Recent developments in commercial expert system shells
also show the artificial intelligence (AI) industry’s endorsement of the object-
oriented approach for knowledge representation. For example, KEE demonstrates
how object-oriented tools can be used for knowledge-based system building. It is a
hybrid AI tool which integrates rule-based systems, frame-based systems, graphics,
and active values (i.e., demons) with an object-oriented kernel [Kunz, Kehler, and
Williams, 1984]. NExpert [Neuron Data, 1987] is a high-end hybrid rules and
object-based expert system environment with an object-oriented architecture.

7. System analysis and design methods. In an object-oriented system analysis and
design method, the decomposition of a system is based on the concept of objects
[Botch, 1986]. Object-oriented design combines data structure-oriented, procedure-
oriented (i.e., data flow-oriented), and architectural design because objects carry
the notions of data structures and procedures (i.e., methods) and at the same time
message passing among objects defines the interface of the system architecture
[Pressman, 1987].

8. Organization modeling. The information processing model was developed by Gal-
this information processing paradigm by applying an object-oriented approach to
the modeling and simulation of organization. Blanning [1987] also believed that an object-oriented approach could be applied to the analysis of organizations as well as to the design and implementation of systems (e.g., information systems, decision support systems) that support these organizations.

Using the object-oriented approach in information system analysis and design, as well as organization modeling, extends the use of the object-oriented approach to conceptual modeling, which is far beyond the original notion of objects at the implementation and programming levels. This shift in focus demonstrates the generality of the object-oriented approach. However, Blanning neither addressed the issues of integration of organization and information systems nor demonstrated how tools can be developed to support the modeling process. Botch [1986] also showed concern for the way tools can be used to support object-oriented development and called for further research in this direction.

The popularity of object-oriented systems arises from the belief that the notion of an object reduces the semantic gap between the model and the real world. In this research, the author will address issues related to the integration of organization and information systems modeling as well as the development of automated tools based on object-oriented modeling. The possibility of applying the same object-oriented approach to various phases of system development (including organization and information systems) promises a cohesive integration of organization and information systems.
Chapter 3

Research Methodology

In this chapter, several classification schemes of research are first presented. The requirements for the use of system development as a valid research methodology are defined. The difference between a research domain and a research methodology is explained. A system development research process is presented from a methodological perspective. The research domain of this research is in software engineering and a section is designated to the discussion of software engineering and related issues. Finally, various research methodologies applied in this research are described and the research process is presented.

3.1 Research and Its Classification

Research is “systematic, intensive study directed toward fuller scientific knowledge of the subject studied” [Blake, 1978, p. 3]. From literature reviewed, the author found that there are various ways to classify research. Research can be classified according to research domains and purposes, as well as the processes and tools used. Although these classifications overlap to some extent, research in various categories differ in focus, as indicated by the followings examples of research classification schemes:
1. Basic and applied research. Basic research in developing and testing theories and hypotheses is undertaken in response to the intellectual interests of the researcher, rather than for a practical reason. Applied research is the application of knowledge to solve problems of immediate concern [Blake, 1978; Bailey, 1982].

2. Scientific and engineering research. There is no logical distinction between the methods of the engineer and the pure scientist. Both are concerned with confirming their theoretical predictions. However, they do differ in the scale of their experiments and their motives. In the engineering approach, the artistry of design and the spirit of "making something work" are also essential [Davies, 1973].

3. Evaluative and developmental research. There are two types of research that are directed to solving problems: evaluative and developmental [Ackoff, Gupta, and Minas, 1962]. The developmental type of research "involves the search for (and perhaps construction or synthesis of) instructions" which yield a better course of action [Ackoff, Gupta, and Minas, 1962, p. 24]. Developmental research has largely been ignored by researchers of social and behavioral sciences. However, without research efforts directed toward developing new solutions, there will be little opportunity left for evaluative research.

4. Research and development. Development is the systematic use of scientific knowledge directed toward the production of useful materials, devices, systems, or methods, including design and development of prototypes and processes [Blake, 1978]. Hitch and McKean classified development work as: exploratory, advanced, engineering, and operational development. The first three types of development can also be labeled as applied research [Blake, 1978]. The author believes that without development, research has no use; without research, development has no base.

5. Formulative and verificational research. The goal of formulative research (also called exploratory research) is to identify problems for more precise investigation, to develop hypotheses, as well as to gain insights and to increase familiarity with
the problem area. The goal of verification research is to obtain evidence to support and refute formulated hypotheses [Grosof and Sardy, 1985].

System development as a research methodology falls into the category of applied science and belongs to the engineering, developmental, and formulative type of research. The importance of systems development research to human knowledge is addressed in Section 3.2. However, academic research generally focuses on the extension of man’s knowledge and demonstration of technical excellence [Blake, 1978]. The development of a system, especially the development of a software system, has to follow a certain research process and conform to certain criteria to be qualified as academic research. The system development research process will be discussed in Section 3.4.

3.2 System Development as a Research Methodology

The system development methodology is an age-old method and process that human beings use to study nature and to create new things. Table 3.1 lists a few examples of how system development has contributed to some research domains, listed in chronological order.

In the airplane design area, the Wright brothers built the first airplane before the aerodynamics field had been created. Aerodynamics and aerostatics are branches of engineering that were created by studying model airplanes built in the laboratory and learning from experiences gained through building real airplanes. The aircraft industry now is using the most advanced CAD/CAM tools to design next generation airplanes. These CAD/CAM tools have encoded theories developed in aerodynamics and heuristics learned from building real systems. The use of CAD/CAM tools has saved the airplane industry millions of dollars by improving the performance of new airplanes. The pattern of this research progress is: 1) build a system, 2) develop theories and principles from
<table>
<thead>
<tr>
<th>Domain</th>
<th>Progress</th>
</tr>
</thead>
<tbody>
<tr>
<td>Airplane Design</td>
<td>The Wright brothers designed the first airplane (1903)</td>
</tr>
<tr>
<td></td>
<td>Development of aerodynamics and aerostatics</td>
</tr>
<tr>
<td></td>
<td>CAD/CAM for airplane design and manufacturing (1980s)</td>
</tr>
<tr>
<td>Memory Management in Computer Systems</td>
<td>Real memory management</td>
</tr>
<tr>
<td></td>
<td>Simulation of memory usage</td>
</tr>
<tr>
<td></td>
<td>Virtual memory management</td>
</tr>
<tr>
<td></td>
<td>Mathematical models of memory usage</td>
</tr>
<tr>
<td>Software Development</td>
<td>Structured programming (Early 1970s)</td>
</tr>
<tr>
<td>Methods and Tools</td>
<td>Structured design (Mid 1970s)</td>
</tr>
<tr>
<td></td>
<td>Structured analysis (Late 1970s)</td>
</tr>
<tr>
<td></td>
<td>Automated techniques (Early 1980s)</td>
</tr>
<tr>
<td></td>
<td>Empirical studies of system development methods (1980s)</td>
</tr>
<tr>
<td></td>
<td>CASE techniques (Late 1980s)</td>
</tr>
<tr>
<td>Computer-Supported</td>
<td>Electronic mail (late 1960s)</td>
</tr>
<tr>
<td>Collaborative Work</td>
<td>Teleconferencing (1970s)</td>
</tr>
<tr>
<td></td>
<td>Group decision support systems (GDSS) (Early 1980s)</td>
</tr>
<tr>
<td></td>
<td>Evaluation of GDSS (Mid 1980s)</td>
</tr>
<tr>
<td></td>
<td>Intelligent E-Mail (Mid 1980s)</td>
</tr>
<tr>
<td></td>
<td>Integrated Computer-Supported Collaboration</td>
</tr>
<tr>
<td></td>
<td>Work Systems (1990s)</td>
</tr>
</tbody>
</table>

Table 3.1: Examples of the Contribution of System Development to Research Areas
observing behavior, 3) encode expertise in computer software for easy access, and 4) use tools to help the development of new systems.

In the case of memory management in computer systems, various memory management techniques were developed based on previous system building experiences and evaluation of the system that were built [Deitel, 1984]. Simulation was first used to study the pattern of memory usage of various memory management schemes. Later, mathematical models also were developed to study their performance. In one instance, Peter Denning developed the Working-Set Model of program behavior from the observation of the locality phenomena in a paging memory system which was developed by students at Massachusetts Institute of Technology [Denning and Schwartz, 1972]. Locality is the phenomenon that programs tend to reference main memory in non-uniform and highly localized patterns. "It is an empirical (observed) property rather than theoretical one" [Deitel, 1984; p. 222]. A working set memory management policy was proposed to improve system performance by preventing possible thrashing [Denning, 1968]. The pattern of this research progress is: 1) build a system, 2) observe its behavior, 3) develop a mathematical model to explain the behavior of the system, and 4) formulate a new mechanism to improve the system performance.

In the area of software development methods and tools, structured programming, structured design, and structured analysis have been introduced in sequence by practical experiences learned from developing real systems [Martin and McClure, 1988]. Empirical studies of programming were motivated by the publication of Gerald Weinberg's The Psychology of Programming in 1971. Empirical studies and comparison of various system analysis and design methods were started as late as 1982 [Hoffer, 1982]. The patterns of progress in this area are: 1) learn from the development software systems, 2) formulate structured methodologies to improve system development process, 3) develop automated tools to support the use of structured methodologies, and 4) study the use of various design methods and tools empirically.
In the computer-supported collaborative work (CSCW) area [Greif, 1988], the advent of electronic mail, teleconferencing [Johansen and Bullen, 1984], and group decision support systems [Huber, 1984] led to research studying the effects of these CSCW tools on organization structures and dynamics, as well as individual and group behaviors in using them. Such empirical studies became possible because of the existence of these technologies. The patterns of progress in CSCW are: 1) introduce systems (i.e., electronic mail and teleconferencing) to support collaborative work which increases demand for more technology support in human collaboration, 2) develop more new systems such as GDSS, 3) study the use of these systems empirically, and 4) apply results from empirical studies to improve existing systems, e.g., intelligent electronic mail [Malone et al., 1987].

Some basic assumptions in system development emerge from the examples discussed above and can be summarized as follows:

1. Do not assume that you already know too much about a domain. Building a prototype system always helps. A prototype system is a model of the real world system which can be used to study the domain. Researchers may learn more about all aspects of a domain from observing the behavior of a prototype and from the process of building it.

2. The process of building systems of various kinds leads to an understanding of the system building process as well as of products of the process, i.e., a good understanding of a domain. This will result in a successive refinement and improvement of both the product and the process.

3. New systems developed may change the processes and concepts in a domain and thus expand the horizon of human knowledge about their surroundings.

4. The use of tools (e.g., CASE and CAD/CAM) to support system development has also been found very useful in amplifying human intelligence and in transferring
knowledge for wider use.

5. System development methodology can be used in conjunction with other methodologies, such as laboratory experiment, field study, and case study. Empirical research may provide valuable feedback. The result can lead to the further development of the system built and a better understanding of the domain.

3.3 Research Domain and Research Methodology

System development, especially the development of software systems, is a research domain as well as a research methodology. Questions such as “Does the development of a software system constitute a research project (in the academic sense)?”, have been frequently raised. The dual nature of system development usually is the cause of confusion and is discussed in this section. Definitions of research domain and research methodology are provided for clarification.

A research domain is the subject matter under study in a research project. A research methodology consists of the combination of the process, methods, and tools which are used in conducting research in a research domain. The process which constitutes the system development research methodology is discussed in Section 3.4. Developing system development tools (i.e., a special case of system development) is the major research domain of this dissertation. Just as statistics provide a method for conducting empirical research, software engineering is a generally agreed-upon method for conducting software development research. The discussion of software engineering can be found in Section 3.5.

Some basic assumptions about the use of research methodologies are as the following [Nunamaker, 1988a]:

1. The research method is not more important than the research question.
Research methods are means of finding truth in research domains. Without an understanding of a research domain, researchers might ask a wrong question or formulate a meaningless hypothesis. No matter what research methods they apply, wrong or irrelevant questions can only lead researchers to wrong conclusions. System building as a research methodology not only can be used as a means of better understanding a research domain, can sometimes even change the processes and products in a research domain, leading to recognition of additional research interests. System development research can be used in conjunction with other methodologies.

2. A valid methodology is necessary, but not sufficient for good research.

A sound research methodology provides methods and tools for the systematic study of a subject area, but does not guarantee good research results. A critical component of a successful research process is a good understanding of the research domain. If researchers do not know what already has been done in a domain, how can they identify what can be added to the domain?

In Figure 3.1, a simple framework is proposed to explain the relationship between research domains and research methodologies. The body of knowledge includes both research domains and research methodologies. A research process involves understanding of research domains, finding out meaningful research questions, and applying valid research methodologies to address the questions asked. Results from a good research project can contribute to the body of knowledge both by expanding knowledge in a given domain and by enriching methodologies applied in a domain.

3.4 Process of System Development Research Methodology

Methodology is the philosophy of the research process which “includes the assumptions and values that serve as a rationale for research and the standards or criteria the researcher
Research Process

Apply Valid Research Methodologies + Understand the Research Domains

Body of Knowledge

Knowledge of Research Methodologies + Knowledge of Research Domains

Research results contribute to the body of knowledge

Figure 3.1: Research Domains and Research Methodologies
uses for interpreting data and reaching conclusion" [Bailey, 1982, p. 26]. Research process, the heart of research methodology, is the application of scientific method to the complex task of discovering answers (solutions) to questions (problems)" [Blalock and Blalock, 1982].

The research process in social and behavioral science can be summarized as follows [Bailey, 1982; Blalock and Blalock, 1982]: 1) choosing the research problem(s), 2) stating hypotheses, 3) formulating the research design, 4) gathering data, 5) analyzing data, and 6) interpreting the results so as to test hypotheses. The author found a parallelism between the social (behavioral) and engineering (development) types of research, although the detailed methods and tools used may differ.

Figure 3.2 shows a system building process model from a research methodology viewpoint. Research issues which should be addressed in each phase are also identified. A system building process (with emphasis on software development) consists of the following steps:


Researchers should first justify the significance of research questions pursued. An ideal research problem is one that is new, creative, and important in the field. When the proposed solution of the research problem cannot be proved mathematically and tested empirically, or simply because it proposes a new way of doing things, researchers have to develop a system to demonstrate the validity of the solution, based on the suggested new methods, techniques, or design. Once the system has been built, researchers can study its performance and the phenomena related to its use in order to gain insights into the problem of the research. A clear definition of the research problem provides focus for the research throughout the development process. The research question should be discussed in the context of an appropriate conceptual framework. Various disciplines should also be explored
**System Development Research Process**

1. Construct a Conceptual Framework
2. Develop a System Architecture
3. Analyze & Design the System
4. Build the System
5. Observe & Evaluate the System

**Research Issues**

- State a meaningful research question
- Investigate the system functionalities and requirements
- Understand the processes/procedures of the system building
- Study relevant disciplines for new approaches and ideas
- Develop a unique architecture design for extendibility, modularity, etc.
- Define functionalities of system components and how they interact with each other
- Design the data structures and processes to carry out the system functions
- Develop alternatives solutions to design problems and choose one solution
- Learn about the concepts, framework, and design through the system building process
- Gain insight about the problems and complexity of the system
- Observe the use of the system by case study and field study.
- Evaluate the system by lab. experiment and field experiment
- Develop new theories/models based on the study of the system built
- Consolidate experiences learned

Figure 3.2: The Process of System Development Research
to find additional approaches and ideas which could be incorporated in the new system. The reason that developing a system is necessary for research purposes should also be addressed.

2. Develop a system architecture.

A good system architecture provides a road map for the system building process. It puts the system components into the correct perspective, specifies the system functionalities, and defines the structure relationships and dynamic interactions among system components. In the development type of research, researchers have to identify the constraints given by the environment, state the objectives of the development efforts (i.e., the focus of the research), and define the functionalities of the resulting system to achieve the stated objectives. Requirements should be defined so that they are measurable and thus can be validated at the testing stage. In the empirical and evaluative type of research, formulating the research hypotheses is an important step in the research process. In the development type of research, researchers usually do not formulate an explicit hypothesis, but they do make assumptions about the research domain and the technical environment for developing the system. Researchers state the system requirements under the constraints of these assumptions and use system requirements to guide the system design and implementation. Depending on the focus of the research, one might emphasize the new functionalities or interface features of the proposed new system rather than the speed and response time of the system.

3. Analyze and design the system.

A research project's requirements may be driven by new functionalities envisioned by the researcher or may be determined partially by the research sponsor's requests. Design is the most important part of a system development process. Design involves the understanding of the domain studied, the application of relevant scientific and technical knowledge, the creation of various alternatives, and the synthesis and evaluation of proposed alternative solutions. Design specifications will be used
as a blueprint for the implementation of the system. For a software development project, design of data structures, database, or knowledge base should be determined at this phase. The program modules and functions also should be specified at this stage after alternatives have been proposed and explored and final design decisions been made.

4. Build the system.

Implementation of a system is used to demonstrate the feasibility of the design and the usability of the functionalities of a system development research project. The process of implementing a working system can provide researchers insights into the advantages and disadvantages of the concepts, framework, and design chosen. The experiences and knowledge accumulated will be helpful in re-designing the system. Empirical studies of the functionalities and the usabilities can only be performed after the system has been built.

5. Observe and evaluate the system.

Once the system is built, researchers can test its performance and usability as stated in the requirement definition phase. They can also observe its impacts on individuals or organizations who use it. The test results should be interpreted and evaluated based on the conceptual framework and the requirements of the system defined at the earlier stages. Development is an evolutionary process. Experiences gained from developing the system usually will lead to the re-design and continuing development of the system, or even discovery of a new theory for explaining observed new phenomena.

The use of system development as a research methodology in information systems should conform to the following criteria: 1) The purpose is to study an important phenomenon in areas of information systems through system building, 2) The results have significant contributions to the domain, 3) The system is testable against all the objectives
and requirements stated, 4) The new system can solve certain information system problems better than others, and 5) Experiences and design expertise learned from building the system can be generalized so that they can be used in other situations.

In every phase of the system development process, researchers gain insights about a domain that will lead to changing some design decisions made in previous phases. When the developed system is a software system, all the principles of software engineering methods and techniques should be used to improve the quality of both the development process and the research results. This leads us to a discussion of software engineering as a discipline.

3.5 Software Engineering and Engineering

It is essential to understand what software is and the importance of software productivity in order to appreciate research efforts in software engineering [Nunamaker and Chen, 1987; Boehm, 1987]. New software systems developed in the information systems area definitely change the way people think about information systems and the way they solve information systems problems [Lyytinen, 1987]. For example, the advent of spreadsheet software and financial modeling languages makes decision support systems a feasible solution to managerial decision making problems. The new hypertext systems will probably change the way people read and write as well as the way they think and communicate [Conklin, 1987]. Information systems is an applied discipline. If research in information systems fails to be applicable to the real world, then the research efforts are in vain [Galliers and Land, 1987]. Software, which is a critical part of modern information systems, can be broadly defined as [Freeman, 1987]: 1) the embodiment of the functions of a system, 2) the captured knowledge of an application area, and 3) the information produced during the system development process. Due to the complexity of a software system, its success relies on the application of rigid discipline in its development process,
i.e., software engineering. Relevant definitions of software engineering are discussed in Section 3.5.1.

3.5.1 Definitions of Software Engineering

There is no generally agreed upon definition of software engineering, but those provided in the following will serve as a basis for discussion:

1. Naur's definition: "The phrase software engineering was deliberately chosen as being provocative, in implying the need for software manufacture to be used on the types of theoretical foundations and practical disciplines, that are traditionally in the established branches of engineering" [Naur, Randell, and Buxton, 1976, p. 9].

2. Vick's definition: In the preface of Software Engineering Handbook Vick and Ramamorthy [1984, p. IX] give their definition of software engineering as: "Software engineering (is used to) ... interpret and apply sound engineering discipline and practice to the design, development, testing, and maintenance of software systems." "Software engineering is not just a collection of tools and techniques, it is engineering ... software engineering can learn from other engineering discipline...."

3. Wegner's definition: Wegner [1983, p. 167] emphasizes the conceptual level constructs of software development, saying that "the paradigms of software engineering are those of conventional engineering modified to take into account the fact that software is a conceptual rather than a physical product."

5. Macro and Buxton’s definition [1987, p. 3]: “The establishment and use of sound engineering principles and good management practice, and the evolution of applicable tools and methods and their use as appropriate, in order to obtain — within known and adequate resource provisions — software that is of high quality in an explicitly defined sense.”

In summary, software engineering has the following characteristics: 1) It is an engineering discipline, 2) It studies the methods, techniques, tools, processes, and management of the development of software systems, and 3) It is systematic.

The development of (software) systems to conduct research can be traced back to the engineering school approach to thought and training. Engineering research also has heavily influenced system development research methodology. The discussion of some basic principles in the engineering approach in Section 3.5.1.1 will help clarify the rationale behind system development research.

3.5.1.1 Engineering and Its Implication

Engineers generally agree that “progress is achieved primarily by posing problems and systematically following the process to construct systems that solved them” [Denning et al., 1988, p. 4]. This is why engineering has been defined as “the application of technology (technical knowledge) to meet human needs more effectively” [Roadstrum, 1967, p. 130].

Even though there is a lot of crossover between design and experimentation, in the area of software engineering design is the dominant paradigm [Denning, 1988, p. 4]. A detailed discussion of the role of design in software development can be found in Freeman [1984]. Roadstrum’s discussion on (engineering) design twenty years ago can still be applied to most current software engineering techniques in software development
1. Engineering is not technology. It is a doing (a process). It is an application (i.e., the system built).

2. Engineering is a creative process. In engineering research, the process requires both creativity and technical skills.

3. Engineering helps to provide more effective ways to perform tasks. Effectiveness in engineering can be measured in terms of lower cost, better quality, shorter system building time, ease of use, etc. Usually measurable performance standards can be defined to evaluate engineering products and their usages.

The importance of design in the engineering type of research explains why the design of the system developed in this research is the main focus of Chapter 5.

3.6 A Typology of Research in Software Engineering

Scott Morton [1984] has identified nine categories of research in Management Support Systems (MSS): 1) building a prototype system, 2) constructing a methodology, 3) developing a theory, 4) formulating a concept, 5) performing a laboratory test, 6) performing a real-world test, 7) conducting a survey, 8) describing a case, and 9) declaring the "truth."

As a research domain, software engineering can also be classified according to these nine categories. The following discussion is adapted from the Scott Morton’s typology for MSS research and applies it to the software engineering research. The author’s focus is on how other research methodologies, especially empirical research methods, can be used in studying software engineering.
1. Build a (prototype) system.

"Building a prototype system is an engineering concept" [Scott Morton, 1984]. It has been widely used in software engineering research. Researchers in software engineering often conduct their research by building a system. To qualify systems building (i.e., software development in the software engineering area) as academic research in MIS, a set of criteria is needed. Some these criteria were proposed in the engineering research process discussion.

2. Construct a methodology (i.e., a method).

Parnas’s paper [1979] on using modularization in system design basically proposes a concept of how to build a software system with improved flexibility and comprehensibility in shorter time by using modularization. Some software engineering principles such as information hiding and hierarchical decomposition are derived from the concept presented in this paper. Booch’s paper on “Object-Oriented Development” is also a good example of how to construct a new software design method, but at a more specific level [Booch, 1986].

3. Develop a theory.

Halstead has developed a theory, called software science, that calculates the operators and operands of a program to estimate some properties of that program [Halstead, 1977]. This is a classic example of building a theory for software engineering.

4. Formulate a concept (i.e., a framework).

Research in this category suggests “a framework that is found useful in organization ideas and suggesting actions” [Scott Morton, 1984]. Nunamaker’s and Chen’s [1987] work on proposing a framework to study software productivity and reusable software components is in this category.

5. Conduct an empirical laboratory test.

Basili’s paper “Experimentation in Software Engineering” has provided a frame-

6. Conduct a real-world test or a survey.

Survey studies used in software development research often focus on the evaluation of different development methods used in a real-world setting. Mahmood’s paper on comparing the SDLC (Software Development Life Cycle) and prototyping methods is an example of using a survey study in software development domain [Mahmood, 1987]. A similar study also can be found in Guimaraes’ paper [1985].

7. Describe a case.

Developing a system is learning by doing. Knowledge gained from the development process can be consolidated into a case study which describes the rationale, process, and experiences learned from developing a system. Examples can be found in [O’Keefe and Wade, 1987; Vinze et al., 1987]. In software engineering, the researcher who conducts a case study is usually the researcher who developed the system. This usually is not the case in social science study.

8. Declare the “truth.”

Something very close to the declaration of truth is found in Dijkstra’s letter to the editor of Communications of the ACM in which he declared that “go to statement considered harmful” [Dijkstra, 1968].

The main research domain of this research is software engineering, computer-aided software engineering in particular. The concepts used in building CASE tools have been extended to the building of GDSS tools so as to capture information for
organization modeling. The resulting organizational model can be used to build more effective information systems. The research methodology employed is mainly the system development research methodology. Some other research methods have also been used, and how these various research methodologies have been applied in this research is described in Section 3.7.

3.7 Research Methodologies Application

Several research methodologies used in this research are discussed in this section. The major research methodology used is system development through prototyping.

3.7.1 Conducting Case Studies

Applying case studies in software system development research means studying the use of a system or studying a system’s design and functionalities. Knowledge learned from experience can be used for the development of a new system. Calling the study of a software system’s design and functionalities a "case study" might be disputed by researchers in some fields. However, this method has been used by system development researchers intensively. Cases that represented use of CASE, GDSS, and other related areas were first reviewed in Chapter 2 to reach an understanding of the research domain and some CASE tools were studied to learn more about their functionalities. The author has had experience in using PSL/PSA on an IBM 4381 [IBM, 1984b], has lectured on PSL/PSA to undergraduate and graduate classes in system analysis and design, and has helped students use PSL/PSA on the IBM mainframe. This experience helped him learn how CASE tools are used in documenting system descriptions [Techroew and Hershey, 1977]. At the same time, he observed the frustration and confusion people experienced in learning to use the system. Such confusion and frustration comes from: 1) the complexity of the PSL language (the 17 object types and 66 relation types in PSL language sometimes
confuse more than they help), 2) poor user interface design that makes the learning curve very steep (at the operational level, users must know the syntax of the PSL language, as well as the options and steps in running PSA commands), 3) volumes of system documentation that must be read (documentation for PSL/PSA on the IBM mainframe is about 2,000 pages), and 4) limitation of the PSL language to describe information systems at the analysis and design level. Users of PSL/PSA have to use predefined object types and relation types in PSL to describe a system. Some of the object types are very easily confused, such as ENTITY and SET, CONDITION and EVENT, ATTRIBUTE and SYSTEM-PARAMETER, etc. These observations resulted in making user interface design one of the major concerns in the system development process of this research.

The Plexsys project at the University of Arizona Management Information Systems Department has extended the original notion of PSL/PSA by using computers to support documentation of information systems and to support information system planning [McIntyre, Konsynski, and Nunamaker, 1986, 1986a] and business planning [Applegate, Konsynski, and Nunamaker, 1986], both at individual and at group levels. In order to support such a variety of application domains and types of users, a powerful, flexible, and friendly software environment is needed. This software environment not only should provide a basis for defining languages to allow the description of various application domains, but also should be able to generate consistent interface tools for using the defined languages to describe target systems. A metasystem approach to building such a software environment was therefore adopted.

3.8 Formulating a Framework

A research framework organizes the concepts involved in a research domain. It also provides perspectives on the evolution of the research and points out the directions in which future research may be heading [Freeman, 1987a]. A framework for integrated
system modeling is proposed in this section. This framework, as depicted in Figure 3.3, enabled the author to identify the requirements of an integrated CASE tools. The framework is based on an analysis of the information sources required to build a system model in the context of information system development; its emphasis is on the requirements definition of a system.

The sources of information that are needed for building a new system may be either human experts or information residing in existing systems. Human experts who have knowledge of the target system are managers, users, and information systems (IS) personnel. All of them acquired their knowledge about the system to be built either through an understanding of some similar existing systems or through having a vision of what the system should be. Managers contribute to an information system project their managerial knowledge of how information systems can be used strategically. System objectives are usually set by executive managers. Users who will work with the new system contribute their knowledge of the operational environment to an information system project. They know the operational procedures in detail. Information systems personnel contribute their technical know-how to ensure that the functions of the new system are technically feasible. They also know what information is needed in order to design and implement the system. Information in the existing system also can be converted into a form that can be used as part of a system specification. A taxonomy is proposed to classify CASE tools based on the analysis of information sources for system requirements and the mode of acquiring the information. In this framework, CASE tools are classified as:

1. Individual-based CASE.

Currently, requirements definition is usually performed by system analysts who interview users or managers on a one-on-one basis. The information of a target system is gathered and recorded through a structured methodology, either manually or automatically. For a complex system, the description of what a system is, what a system is supposed to do, and how well the system should work is really a process of
Figure 3.3: A Model of Requirement Definition for a System
incorporating multiple viewpoints from users, managers, and IS personnel. Because individual-based CASE tools do not support the capturing of group interactions, information and rationale of design decisions are lost.

2. GDSS-based CASE.
   The use of GDSS tools (i.e., Plexsys Planning Tools) for requirements definition by one IBM plant has been demonstrated to be very effective [Martz, 1988a]. Bostrom and Anson [1988] also have suggested the use of computer-supported collaborative work (CSCW) tools to support the use of CASE. However, specific design and implementation issues have not been considered. The use of a metasystem approach to build a GDSS-based CASE is suggested in Chapter 6.

3. Reverse engineering-based CASE.
   Using information residing in existing information systems may save time and money in acquiring information about a system that is to be built. Most system development efforts involve renewing or enhancing existing systems. Forms, reports, existing programs, database schema, and documents of existing information systems in an organization can all be used in building a new system. CASE tools which use existing information about a system as a basis upon which to build a new system are called reverse engineering-based CASE. Choobineh et al., [1988] developed a rule-based expert system which can take information contained in a collection of forms used in an organization to create the conceptual schema of a database design. The rationale for using forms as a starting point for database design is that they carry structured information about the information systems of an organization and are easy to access. Bachman Information System [I/S Analyzer, 1988] has developed a system which can take data definition language (DDL) statements and data manipulation language (DML) statements of an existing application to derive its logical database design in Bachman diagram form. The logical design can be used as a base for re-design and then conversion of the new logical design into a new physical design. Most system descriptions in an organization are
documented in natural languages [Chen, 1985]. Peter Chen [1983] has suggested some heuristics for converting an English sentence into Entity-Relationship diagrams. There is a promising future for reverse engineering-based CASE when the technology in text scanning and natural language understanding becomes mature [Scott, 1988].

An integrated CASE environment should provide individual-based, GDSS-based, and reverse engineering-based CASE tools since each type of CASE tool can capture only part of the information required for building a new system. The information captured by each kind of tool should either share a common system encyclopedia or be transformed into a complete system description. An analysis system can then be used to generate analysis reports to check the completeness and consistency of the system description. The resulting reports can be used by managers, users, and IS personnel to define their requirements of the system.

3.8.1 Building a Prototype System

The research conducted in this study used system development methodology as the major research methodology. The first prototype system, written in Turbo Prolog, was built during Spring Semester 1987. The author learned from it that: 1) implementation of a metasystem is feasible on a microcomputer system, 2) attributes should be able to be attached to both objects and relations, 3) relations and relation types should be handled as first-class objects to minimize maintenance efforts, 4) the object-oriented nature of a metasystem suggests the use of an object-oriented environment for its implementation, and 5) logic-based languages, such as Prolog, are difficult to use to control the execution of programs and actions of users.

Experiences gained from implementing the first prototype system were used in the development of the second prototype system, built during the period from August 1987
to July 1988. Smalltalk [Goldberg, 1984], an object-oriented programming environment, was used for the implementation of the second prototype system, upon which the research reported in this dissertation is mainly based.

3.8.2 Designing Empirical Tests

Although conducting empirical studies related to the use of the system built for this research is beyond the scope of this dissertation, the author will discuss some potential empirical studies that can be done to test the functionalities of the system and its design. This discussion can be found in Section 7.2.2.
Chapter 4

Concepts, Knowledge Representation, and Architecture of MetaPlex

In this chapter, the metasystem concept is first introduced. Knowledge representation used in MetaPlex, as well as the formal definition of MetaPlex languages and system descriptions, are defined. A layered approach for building an integrated environment for systems modeling is discussed. The architecture of a metasystem, called MetaPlex, implemented as the kernel of the layered architecture is presented. Some of the interface design features in MetaPlex are explained at the end of this chapter.

4.1 Concept of a Metasystem

In searching for the "best" concepts, languages, theories, and methods for software engineering, Lamsweerde et al. [1988, p. 721] concluded that "in the absence of a 'best' widely accepted and domain-independent methodology, an environment kernel must fit the user's own lifecycle models, methods, and languages instead of imposing (upon) him yet another methodology." The idea of building an environment that can adapt to a wide spectrum of methodologies implies a metasystem approach.
A metasystem used in the context of an information system development environment means a system that can be used to generate a system building environment, using a meta-level language syntax. The metasystem concept has been applied to various types of systems. Table 4.1 lists some examples of metasystems in various application domains.

The acceptance of metasystem concepts in various computer related applications has shown the usefulness of the metasystem approach. In this dissertation, the term *metasystem* will be used to designate a system that can generate system analysis and design tools. The author also extends the metasystem concept to the building of group decision support systems tools. The potential benefits of a metasystem approach for building a software development environment are that such an approach makes it possible to:

1. Customize the tools according to specific methods used in an organization.

Users of a metasystem are allowed to generate a tool based on a specific method. Building customized tools for system development becomes possible. A customized system development environment can reduce the user’s resistance to the use of

<table>
<thead>
<tr>
<th>Area</th>
<th>Metasystem</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compiler building</td>
<td>Compiler-compiler [Aho, 1980; Leverett, et al., 1980]</td>
</tr>
<tr>
<td>Decision support systems (DSS)</td>
<td>DSS generator [Sprague, 1981]</td>
</tr>
<tr>
<td>Programming</td>
<td>Application generator [Cleaveland, 1988]</td>
</tr>
<tr>
<td>Program editor</td>
<td>Program editor generator [Reps and Teitelbaum, 1987]</td>
</tr>
<tr>
<td>System analysis &amp; design tools</td>
<td>Lifecycle support system generator [Yamamoto, 1981]</td>
</tr>
<tr>
<td>Expert system building</td>
<td>Expert system shell [Harmon and King, 1985]</td>
</tr>
</tbody>
</table>

Table 4.1: Metasystems in Various Application Domains

### 4.1.1 Definition of Metasystem Environment

A metasystem used in the context of an information system development environment means a system that can be used to generate a system building environment, using a meta-level language syntax. The metasystem concept has been applied to various types of systems. Table 4.1 lists some examples of metasystems in various application domains.
computer-supported tools.

2. Provide a uniform interface to both tool developers and tool users.
   All the tools generated by a metasystem employ the same interface framework so that the costs and time involved in learning a new tool can be reduced.

3. Save the cost of developing CASE tools.
   Caper Jones of Software Productivity Research, Inc., estimated that the development cost of a full CASE (i.e., management environment, support environment, and technical environment) will be about $83 million [Butler, 1987; Jones, 1987]. Metasystems have already provided some generic functions for building CASE tools and thus will be able to reduce the cost of developing them.

4. Save the cost of using CASE tools.
   Employing CASE technologies is not cheap. Jerrold Grochow [1988] of America Management Systems, Inc., estimated that for a 150-person organization the start-up CASE investment is about $1.5 million. This investment includes 75 workstations, CASE software, and training. CASE software alone will cost about $7.25 million. The annual ongoing cost will be about $347,000. Metasystems will be able to cut the cost of employing CASE technologies because they can generate a spectrum of CASE tools. The consistency in user interface and the underlying conceptual model will also reduce the costs of training and of adapting new CASE tools.

   However, the benefits of metasystems are to some extent offset by the following disadvantages:

1. Complexity and lack of guidelines in defining the meta languages prohibit the wide use of a metasystem. It takes the expertise of many experts to define a useful meta language.
2. Poor user interface design in most metasystems discourages end-users from using them.

3. Some metasystems are not generic enough to generate a wide spectrum of tools, thereby limiting the benefits of using them.

4. Running a metasystem overwhelms the capabilities of personal computers.

4.1.2 Review of Existing Metasystem Environments

To guide the design of the metasystem constructed for this research, several existing metasystems have been reviewed to determine their strengths and weaknesses:

1. SDLA (System Description and Logical Analyzer). Developed at the Hungarian Academy of Science by Demetrovics, Knuth, and Rado [1982], SDLA used a two-level meta-driven specification methods which "provide target-level flexibility but retain the power of narrow target domain" [Demetrovics, Knuth, and Rado 1982; p. 29].

2. ERA Meta-Graph System. ERA Meta-Graph System is a meta graphical language tool developed by Miyamoto [1987] at Software Engineering Research Laboratory, University of Hawaii. The system can generate a rule-based graphics editor for a particular modeling method. Logical representation based on the ERA Model [Chen, 1976] has been used for different model types and their formalism. ERA Meta-Graph has been implemented in both C language and Smalltalk language. Interactive graphic interface is the main feature of the system.

3. Software CAD. Software CAD was developed by Yabuta, Yoshioka, and Murakami [1987], at Fujitsu Limited. It supports three activities in the software development process: editing, verifying, and transforming. Software CAD is a graphical based
metasystem tools. Six basic graphical notations are used to define underlying software development methods. These notation are: form, table, node, edge, inclusion, and text field. Transformer in Software CAD uses a set of transformation rules to transform graphical documents into text form. A transformation rule uses a query to access document database and uses a production rule to convert query results into text.

The two-level meta system architecture in SDLA, the ERA model used in ERA Meta-Graph System, and the transformation rules in Software CAD have been incorporated into the prototype system (i.e., MetaPlex) developed for this research. Previous research in metasystems mainly focuses on the generation of system analysis and design tools, while this research focuses on applying the metasystem approach to the integration and organization and information system modeling, as well as to the generation of customized GDSS as a front-end to CASE tools.

4.2 Knowledge Representation in MetaPlex

Two types of knowledge are essential for system development. An integrated system development environment should be able to capture and use both knowledge of the target systems and knowledge of system design. The knowledge representation scheme used in a system development environment is very important if the structures and behaviors of a target system are to be accurately represented and if the analysis and design expertise in intelligent tools are to be properly codified. In this section, the knowledge representation scheme used in MetaPlex is presented and relevant issues are discussed.

Churchman [1971] defines design as a process for defining objects and relations to achieve some intended purpose. Objects and relations are the basic components in MetaPlex knowledge representation. A system can be defined as a group of related com-
ponents which interact with each other to achieve a high level objective of the system as a whole. Both organizations and information systems are instances of such generic systems. To describe an existing system or to design a new system, system analysts or designers are primarily concerned with the system components and relationships among them. Various knowledge representation schemes have been studied to determine whether they can be used for system specifications. A rule-based system can be used for capturing the designer's expertise on certain design decisions. A frame-based system can be used to represent general design schemata [Lubars and Harandi, 1986]. However, neither one explicitly represents the interrelationships of objects in a system. The knowledge representation scheme used in MetaPlex is based on three levels of abstraction of an object-oriented model: the axiomatic, median, and instance levels [Kottemann and Konsynski, 1984]. Figure 4.1 shows an example of this model.

Terms at the axiomatic level are built into the system. In MetaPlex, a domain is defined using object types and relation types. Object types represent the generic characteristics of concepts, entities, events, things, etc. in a domain. Relation types specify the structural relationships among objects of certain object types. Attribute types are used to represent the generic characteristics of both object types and relation types. Attribute types can be used to include unstructured and procedural knowledge, as well as decision rules.

At the median level, the object types in a domain and the relation types among them have to be defined along with the definition of their attribute types. This definition is called a system description language in MetaPlex. The prototypical knowledge of the domain is thus captured at the median level. The median level knowledge is used as template to guide the system specification process. A description of a target system at the instance level will be defined in terms of objects and relations confined by a system description language defined at the median level. The specification of a target system is called a system description or a target system description in MetaPlex.
Figure 4.1: MetaPlex Knowledge Representation Model
4.2.1 The Representation of Relation Types in MetaPlex

In MetaPlex a relation type is defined as two, or more than two, object-type groups connected by connectors among them, as shown in Figure 4.2. The language representation of the relation type in Figure 4.2(c) is:

\[
\text{[GROUP-ITEM] consists-of: [GROUP-ITEM, DATA-ELEMENT]} \\
\text{[DATA ELEMENT, GROUP ITEM] is-part-of: [GROUP-ITEM]}
\]

The complementary ways of describing a relation type for each object-type group involved in the relation type are defined in the system description language. The MetaPlex system can use the information to cross-reference the target system descriptions. For example, once a user has entered the following relation of a target system according to the relation type defined in Figure 4.2:

\[
\text{customer-basic-data} \\
\text{consists-of: customer-name, customer-address, customer-type}
\]

MetaPlex can figure out its equivalent relations as:

(a) customer-name
   is-part-of: customer-basic-data
(b) customer-address
   is-part-of: customer-basic-data
(c) customer-type
   is-part-of: customer-basic-data

By bringing all relevant information about an object together in one report, the cross reference capability reduces the user's effort in defining a system and facilitates the user's understanding of the target system specification.
name: GROUP-ITEM-DECOMPOSITION  
comment: "Decomposition of group item into group items and data items."

no. of group: 2
object types groups: [[GROUP-ITEM], [GROUP-ITEM, DATA-ELEMENT]]
connectors: ['consists-of', 'is-part-of']
sequences: [[1,2], [2,1]]
occurences: [#one, #many]
attribute type set: []

(a) An Internal Representation of a Relation Type

(b) A Graphical Representation of a Relation Type

Relation Type Statement:
GROUP-ITEM consists-of:
GROUP-ITEM, DATA-ELEMENT

Equivalent Relation Type Statement:
GROUP-ITEM, DATA-ELEMENT is-part-of:
GROUP-ITEM

(c) An External Representation of a Relation Type

Figure 4.2: The Cross Referencing of a Relation Type
Internally, the system has to capture the following information about a relation type:

1. Relation type name. A unique name of the relation type has to be specified.

2. Relation type comment. A comment is used to describe the use of the relation type.

3. The number of object-type groups. A relation type may consist of two or more groups of object types. The number of object-type groups involved in a relation type is stored explicitly. An \( n \)-ary relation can be specified with \( n \geq 2 \).

4. An array of object-type groups. A group of object types consists of one or more object types. An \( n \)-ary relation type will have \( n \) groups of object types stored in an array. The same object type may appear in different object-type groups of a relation type so that a recursive relation can be represented. Recursive relations are very useful and necessary in describing a system structure.

5. An array of connectors. Relations can be represented as structured statements. Connectors are used in these structured statements so that a statement of a relation can be easily understood by users.

6. An array of occurrence of object-type groups. Each group of objects involved in a relation has either \#one or \#many occurrence. Users can describe one-to-one (linear structure), one-to-many (tree structure), and many-to-many (network structure) relations. Any complex relation can be easily represented in MetaPlex.

7. An array of sequences of object-type groups in the equivalent statements of a relation type. There are \( n \) equivalent statements to describe an \( n \)-ary relation type. Each equivalent statement starts from a different object-type group within a relation type. A sequence specifies how an equivalent statement is constructed from object-type groups.
A graphical representation of a relation is shown in Figure 4.2. The declarative nature of internal data structures makes the representation and manipulation of languages and target system descriptions much easier.

4.2.2 Abstraction in MetaPlex

Researchers have identified three major abstraction mechanisms for describing a target system: Classification, Generalization/Specialization, and Aggregation [Gibbs, 1985]. The equivalent representations of these abstraction mechanisms in MetaPlex, discussed below, demonstrate the expressive power of the MetaPlex knowledge representation scheme.

1. Classification. The "object type" and "object" in MetaPlex are equivalent to "class" and "instance". For most applications, two levels of classification are sufficient [Mylopoulos, Berstein, and Wong, 1980]. In MetaPlex, relation types have been handled as first-order classes so that relations can be managed flexibly.

2. Generalization/Specialization. The class hierarchy in an object-oriented system can be represented by using an "ako" relation among object types. For example, we can define "REPORT" is-a-kind-of: "DATA" and "MONTHLY REPORT" is-a-kind-of: "REPORT". However, property inheritance along the class hierarchy has to be handled by a specially designed inference engine.

3. Aggregation. There are two types of aggregation: Cartesian aggregation and cover aggregation. In MetaPlex, Cartesian aggregation means that an object is an aggregation of its attributes. The cover aggregation can be specified by using a decomposition relation in the following format:

\[
\text{[OBJECT TYPE]} \text{ has-parts: [OBJECT TYPE]}
\]

\[
\text{[OBJECT TYPE]} \text{ is-part-of: [OBJECT TYPE]}
\]
4.3 Formal Definitions of MetaPlex System Description Languages and System Descriptions

Instead of using the traditional Backus Naur Form, set theory [Pinter, 1971] and first-order logic [Barr and Feigenbaum, 1983] are used to represent the basic structures of MetaPlex languages and system descriptions. "The essence of set concept is abstraction. Set operations allow us to manipulate abstraction without having to specify their detailed nature" [Zeigler, 1984; p. 21]. Zeigler has successfully used set theory to represent discrete event simulation models. Set and first-order logic also were selected because of the declarative nature of set and logic notations and the natural mapping between the set concept to the underlying object-oriented implementation.

The use of notational conventions are explained below:

1. Variables that start with a capital letter stand for sets, although a set can be an element in another set. Variables that start with lower-case letters stand for a basic element, such as an integer or a string.

2. As a naming convention, "Set" is concatenated at the end of a variable name to emphasize that this variable stands for a set. "Class" is concatenated at the end of a variable name to emphasize that this variable stands for a class. "Set" is used here to represent a set with a finite number of elements while "Class" is used to represent a set with infinite elements. "OC" is concatenated at the end of a variable name to emphasize that this variable stands for an ordered collection. An ordered collection can be referenced by using indexes. An array in conventional languages is a common implementation of an ordered collection.

3. ∀ means "for all", → means "infer", ∧ means "AND", ∨ means "OR", and ∃ means "there exists". Notations in first-order logic are used to represent functions or operators. For example, p(x) means the element p of the object x will be returned.
if x has an element called p.

4.3.1 A Formal Definition of MetaPlex System Description Languages

A MetaPlex system description language is a template which is used to define target systems of a domain. Table 4.2 is a formal definition of MetaPlex languages.

A MetaPlex language consists of a name, a comment, a set of ObjectTypes and a set of RelationTypes. Each element of ObjectTypeGroupOC in a RelationType is a subset of the ObjectTypeSet in the language. Names of elements in ObjectTypeSet and in RelationTypeSet should be unique.

4.3.2 A Formal Definition of MetaPlex System Descriptions

System descriptions are specifications of target systems. A system description is confined by the MetaPlex language used. Table 4.3 is a formal definition of system descriptions in MetaPlex.

A MetaPlex system description consists of a name, a comment, a set of Objects and a set of Relations. Each element of ObjectGroupOC in a Relation is a subset of the ObjectSet. Names in ObjectSet and in RelationSet should be unique. All the attributes of an object and a relation in a system description are defined according to the AttributeTypeSet of the objectTypes and relationTypes that the object and the relation belong to.
MetaPlex-Languages = \{(name\textsubscript{i}, comment\textsubscript{i}, ObjectTypeSet\textsubscript{i}, RelationTypeSet\textsubscript{i}) | name\textsubscript{i}, comment\textsubscript{i} \in \text{String} \land ObjectTypeSet\textsubscript{i} \subset \text{ObjectTypeClass} \land
( \forall x. x \in RelationTypeSet\textsubscript{i} \rightarrow \text{ObjectTypeGroup}(x) \subset \text{ObjectTypeSet}\textsubscript{i} ) \land
(name\textsubscript{i} \neq name\textsubscript{j} if i \neq j) \}\}

ObjectTypeSet = \{objectType\textsubscript{i} | objectType\textsubscript{i} \in \text{ObjectTypeClass} \land
(name(objectType\textsubscript{i}) \neq name(objectType\textsubscript{j}) if \text{objectType}\textsubscript{i} \neq \text{objectType}\textsubscript{j} ) \land
i \in \{1,...,n\} \}\}

RelationTypeSet = \{relationType\textsubscript{i} | relationType\textsubscript{i} \in \text{RelationTypeClass} \land
(name(relationType\textsubscript{i}) \neq name(relationType\textsubscript{j}) if relationType\textsubscript{i} \neq relationType\textsubscript{j} ) \land
i \in \{1,...,n\} \}\}

ObjectTypeClass = \{ (name, comment, AttributeTypeSet) | name, comment \in \text{String} \land \text{AttributeTypeSet} \subset \text{AttributeTypeClass} \}\}

RelationTypeClass = \{ (name, comment, noGroup, ObjectTypeGroupOC, SequenceRelation, ConnectorOC, OccurrenceOC, AttributeTypeSet) | name, comment \in \text{String} \land \text{noGroup} \in \{x \mid x \geq 2 \land x \in \text{Integer} \} \land
ObjectTypeGroupOC = \{\text{ObjectTypeGroup}\textsubscript{i} | \text{ObjectTypeGroup}\textsubscript{i} \subset \text{ObjectTypeClass} \land
i, j \in \{1,...,\text{noGroup}\} \}\ \land
SequenceOC = \{s_{i,j} \mid s_{i,1} = i \land \{s_{i,2},...,s_{i,\text{noGroup}}\} = \{1,...,\text{noGroup}\} - \{i\} \land
i, j \in \{1,...,\text{noGroup}\} \}\ \land
ConnectorOC = \{\text{connector}\textsubscript{i} | \text{connector}\textsubscript{i} \in \text{String} \land i \in \{1,...,\text{noGroup} - 1\} \}\ \land
OccurrenceOC = \{\text{occurrence}\textsubscript{i} | \text{occurrence}\textsubscript{i} \in \{\#one, \#many\} \land i \in \{1,...,\text{noGroup} \} \}\ \land
\text{AttributeTypeSet} \subset \text{AttributeTypeClass} \}\}

AttributeTypeClass = \{ (name, comment, prompt, dataType, LegalValues, defaultValue) | name, comment, prompt \in \text{String} \land
\text{dataType} \in \{\text{Integer, Real, Boolean, Description, String} \} \land
\text{LegalValues} \subset \{\text{values of dataType}\} \land
((\text{defaultValue} \in \text{LegalValues} \text{ if LegalValues} \neq \phi ) \lor
(\text{defaultValue} = \phi \text{ if LegalValues} = \phi)) \}\}

Table 4.2: A Formal Definition of MetaPlex System Description Languages
MetaPlex-System-Descriptions = \{ (name_i, comment_i, language_i, ObjectSet_i, RelationSet_i) | 
name_i, comment_i \in \text{String} \land
( \forall x \in \text{ObjectSet}_i \rightarrow (\text{objectType}(x) \in \text{ObjectTypeSet}(\text{language}_i)) \land
(\forall a \in \text{AttributeSet}(x) \rightarrow (\text{attributeType}(a) \in \text{AttributeTypeSet}(x))) \land
( \forall y \in \text{RelationSet}_i \rightarrow (\text{relationType}(y) \in \text{RelationTypeSet}(\text{language}_i)) \land
(\forall b \in \text{AttributeSet}(y) \rightarrow (\text{attributeType}(b) \in \text{AttributeTypeSet}(y))) \land
( \forall \text{ObjectGroup}. \text{ObjectGroup} \in \text{ObjectGroupOC}(y) \rightarrow \text{ObjectGroup} \subseteq \text{ObjectSet}_i ) \land
( \text{name}_i \neq \text{name}_j \text{ if } i \neq j \text{ and language}_i = \text{language}_j ) \}

\text{ObjectSet} = \{ \text{objectType}_i \in \text{ObjectClass} \land
( \text{name}(\text{object}_i) \neq \text{name}(\text{object}_j) \text{ if } \text{object}_i \neq \text{object}_j ) \land i \in \{1, \ldots, n\} \}

\text{RelationSet} = \{ \text{relation}_i \in \text{RelationClass} \land
( \text{name}(\text{relation}_i) \neq \text{name}(\text{relation}_j) \text{ if } \text{relation}_i \neq \text{relation}_j ) \land i \in \{1, \ldots, n\} \}

\text{ObjectClass} = \{ (\text{name}, \text{comment}, \text{objectType}, \text{AttributeSet}) | 
\text{name}, \text{comment} \in \text{String} \land \text{objectType} \in \text{ObjectTypeClass} \land
\text{AttributeSet} \subseteq \text{AttributeClass} \}

\text{RelationClass} = \{ (\text{name}, \text{comment}, \text{relationType}, \text{ObjectGroupOC}, \text{AttributeSet}) | 
\text{name}, \text{comment} \in \text{String} \land \text{relationType} \in \text{RelationTypeClass} 
\text{ObjectGroupOC} = \{ \text{ObjectGroup}_i | \text{ObjectGroup}_i \subseteq \text{ObjectTypeClass} \} \land
\text{AttributeSet} \subseteq \text{AttributeClass} \}

\text{AttributeClass} = \{ (\text{attributeType}, \text{value}) | 
\text{attributeType} \in \text{AttributeTypeClass} \land
\text{value} \in \{ \text{values of dataTypc}(\text{attributeType}) \} \}

Table 4.3: A Formal Definition of MetaPlex System Descriptions
Current development in knowledge representation provides several representation schemes to capture more meaningful descriptions of target systems in the real world. In this section, after several knowledge representation schemes and the ways they can be used to represent organization or information systems have been discussed, the author shows the equivalence of MetaPlex's representation to theirs.

The knowledge representation methods reviewed are discussed in terms of their adequacy for describing a system. Detailed discussion of knowledge representations in general can be found in several sources [Barr and Feigenbaum, 1983; Brachman and Levesque, 1985; Brodie and Mylopoulos, 1986].

Because the real world consists of physical and logical objects which interact within a time frame, software requirements specification should first of all capture what the real world is about. Knowledge concerning what, when, how, and why in the real world includes [Barr and Feigenbaum, 1983]:

1. Objects. Physical and logical objects (what), sometimes called entities, abstract data type, and resources, that are described by their static attributes and dynamic behavior.

2. Events. The knowledge of the time sequence (when) and cause-and-effect relations (why) of things that happen.

3. Performance. When external events or internal events happen, the target system will perform certain functions and procedures (how) to change the states of the system or to trigger a sequence of events.

4. Meta-knowledge. The knowledge of what the system knows. Based on meta-
knowledge, the system can allocate the knowledge resource for appropriate problems.

Three of the most popular knowledge representation schemes – semantic networks, frames, and rules – are discussed in terms of their basic structures, inference methods, usage in system modeling, and equivalent representation in MetaPlex.

4.4.1 Semantic Networks

The study of semantic networks was triggered by psychological studies of human associated memory [Quillian, 1968] and became very popular in the area of natural language processing.

1. Basic structures. Nodes that represent different concepts or objects linked together so as to show relationships among them. The most common links used in semantic nets are listed in Table 4.4.

2. Inference methods. Methods that demonstrate property inheritance through the "isa" hierarchy and pattern matching by the structures of semantic networks. Inference methods are weak for semantic networks.

<table>
<thead>
<tr>
<th>COMMON LINKS</th>
<th>REVERSED LINKS</th>
<th>MEANINGS</th>
</tr>
</thead>
<tbody>
<tr>
<td>ISA_INSTANCE</td>
<td>MEMBER_OF</td>
<td>Instance, membership</td>
</tr>
<tr>
<td>HAS_SUBCLASS (AKO)</td>
<td>HAS_SUPERCLASS</td>
<td>Subtype, classification</td>
</tr>
<tr>
<td>HAS_PART</td>
<td>IS_PART_OF</td>
<td>Aggregation, decomposition</td>
</tr>
<tr>
<td>HAS</td>
<td>ATTRIBUTE_OF</td>
<td>Property, attribute</td>
</tr>
</tbody>
</table>

Table 4.4: The Most Common Links Used in Semantic Nets
3. Use of semantic network in system modeling. The following two statements can be used to construct a simple semantic network:

    information-system has input, output
    order-processing-system isa information-system

The conclusion that the order-processing-system also has input and output can be drawn by property inheritance from the above statements.

4. Equivalence in MetaPlex. In MetaPlex, nodes are represented by objects and links are represented by relations. Users are allowed to build n-ary links. The inheritance along the ISA hierarchy must be handled by building an inference engine (currently not implemented in MetaPlex).

4.4.2 Frames

The basic idea of a frame is that many objects, acts, and events are stereotyped. This stereotyped knowledge can be chunked together into a frame.

1. Basic Structures. A frame is a structured object with slots to describe the attribute of the object. A procedure can be attached to a slot as a demon to monitor an active value. Certain procedures will be called when the active value has been added, deleted, accessed, or modified.

2. Inference Methods. Several inference methods can be used in a frame-based system: a) default reasoning by fill-in-the-slot procedures, b) frame matching, first by matching known facts with the lowest level slots in the frame systems, and then by trying to match the higher level slots, c) property inheritance based on special slots, such as ISA or MEMBER-OF, and d) procedural reasoning by the procedure attachment.
3. Use of a frame in system modeling. Table 4.5 is an example of using a frame to represent a program module.

4. Equivalence in MetaPlex. In MetaPlex, an object type is similar to a frame. Attribute types associated with an object type are similar to the slots of a frame. Methods in object-oriented languages on which the MetaPlex is built can be used to simulate the procedure attachment capability in a frame-based system.

4.4.3 Rules


1. Basic structures. IF conditions (situations) THEN actions (responses).

2. Inference methods. Inference methods in rule-based systems include: a) backward chaining (goal-driven), b) forward chaining (data-driven), and c) bi-directional
3. Use of rules in system modeling. Rules can be used in several ways in system modeling:

- Rules can be used to check the completeness of a system description. For example:

  IF X is a process AND has no output
  THEN notify that the description of X is not complete.

- Rules can be used to capture a heuristic in system design. An example based on the BIAIT method [Carlson, 1979] and BICS method [Kerner, 1979] is given in the following rule:

  IF X is an order and the supplier of X will bill the customer of X
  THEN the information system built for handling order X should include Credit Records and Customer Accounting

- Rules can be used to define the transformation between two representations. In Section 5.3, the author describes the use of rules to represent the transformation from one system description to another.

4. Equivalence in MetaPlex. In MetaPlex, a system class called Rule has been defined. The Rule class has two instance variables, conditions and actions. Methods defined in the Rule class will carry out the inference by pattern matching and rule chaining. Actual rules will be instances of the Rule class. Eventually users will be able to define rules as attributes for both an object and a relation.

To represent the complexity of the real world requirement, a knowledge-based system should include: a) Multiple knowledge representations: modularity and abstraction of knowledge can reduce the burden of managing large knowledge based systems and cut off the irrelevant knowledge in searching solutions for a problem, and b) Multiple
inference methods: the system should support an embedded set of control strategies (such as the focus control block in IBM ESE [Hirsch, et al., 1986]) and the control of accessing knowledge.

4.5 An Integrated Systems Development Environment: A Layered Approach

A metasystem environment can be built on top of the MetaPlex Kernel. A layered approach is proposed to manage the dynamics of the metasystem environment as depicted in Figure 4.3.

The functionalities of each layer from inside out are outlined as follows:

1. MetaPlex Knowledge Base. The MetaPlex Knowledge Base at the center serves as a centralized repository for language definitions, system specifications, transformation knowledge, as well as analysis and design expertise.

2. MetaPlex Knowledge Base Management System. The MetaPlex KBMS manages knowledge about the specification language, analysis and design expertise, and target system descriptions. MetaPlex supports general utilities and tools for managing the knowledge acquired, generating reports, processing queries, checking completeness, as well as supporting the navigation of system descriptions.

3. Tool Base. A spectrum of tools for organization modeling such as Front-End Planning System (FEPS), Critical Success Factor (CSF) and Strategy Identification and Assumption Surfacing (SIAS); for information systems analysis and design such as PSL [Plexsys Planning Tools, 1987] are developed by using the generic or customized user interface. The system will be built on an integrated programming environment where system builders can directly access the programming mechanism to build customized tools. Tools in the Tool Base can use knowledge base system access methods to communicate with MetaPlex’s knowledge base.
Figure 4.3: An Integrated Systems Development Environment: A Layered Approach
4. Process Management Systems. The Process Management System at the outermost layer has a tool selection expert system that can select and configure a set of tools in the Tool Base for a specific organization, problem, or process. The input and output relationships, functionalities, and compatibilities among tools can be represented in MetaPlex. This tool selection expert, based on the Information Center Expert architecture [Helme, et al., 1987], recommends tools for the users to use for solving various problems. The Process Management System can help users by suggesting procedures for a planning or decision making session, selecting a series of tools to facilitate a development process, controlling the progress of system development processes, and documenting results and responsibilities of processes. Process Management System relies on the underlying knowledge base system to represent tools and processes used in a lifecycle model.

The kernel (the two innermost layers) of the architecture can be used as a stand-alone system for building a knowledge base directly so that users are able to use the system without an existing tool base. Tools can be developed incrementally, registered in the Tool Base, and managed by the Process Management System layer. Users of the system can interact with the system at different levels. Eventually, the whole development process is under the control of Process Management System so that users without any prior knowledge about the knowledge base system still feel comfortable using it. The architecture presented allows users to interact with the system at different levels, depending on how much they know about the system and what has been developed. It provides a base for growth and integration of the development environment.

Because of the complexity of the data and procedures involved in the development process, an object-oriented programming environment has been suggested for the implementation of this architecture. The object-oriented approach is most suitable for the following reasons: 1) The object-oriented approach is very close to the mental models of users who describe a system. 2) Object-oriented systems work the best in managing the
heterogeneous data and procedures that exist in the system development environment. 3) Late binding and polymorphism allow the system builders to add new object types or new operations to the existing environment without recompiling existing programs.

Because of the high cost (an estimated $85 million) and long duration (an estimated 5 years) of building a full CASE environment as well as the acquisition costs of using CASE, Caper Jones [1987] proposed an alternative strategy called incremental CASE. At the same time, James Martin called for the building of an integrated CASE (I-CASE) [Butler, 1987]. Layered architecture and the object-oriented implementation provide a basis for the development of both an incremental CASE and an integrated CASE. In the next section, the kernel (the two innermost layers) of the layered architecture, called MetaPlex, is discussed.

4.6 The Architecture of MetaPlex

The design goal for MetaPlex is to develop a simple, but flexible, computer-aided system specification tool that can support and be applied to various domains. The ease of use of MetaPlex is achieved through an interactive menu-driven user interface and a graphic representation of a target system description.

While other metasystems use the compilation approach [Yamamoto, 1981; Demetrovics, et al., 1982], MetaPlex uses the interpretation approach for language definition and target system specification. The interpretation approach makes it much easier to develop description languages and experiment with them. Eventually, users will be able to develop languages of their own without any help from language definers.

The architecture of the MetaPlex Kernel is shown in Figure 4.4. It has three subsystems: Language Definition System, System Description System, and Transformation System. At the meta level, language definers can use the Language Definition System
Figure 4.4: The Architecture of MetaPlex
to define system description languages and their consistency and completeness checking rules. Language definers can also use the Transform System to define languages for the transformation of system descriptions from one language to another. Language Syntax Report and Consistency and Completeness Checking Rules Report can be generated by the Language Definition System. It can be used by language definers to check the completeness of the language defined. Or it can be given to description definers as a user manual of a system description language.

At the instance level, *description definers* can use the System Description System to define a target system description. Report facilities, on-line query functions, and a Structure Browser can be used by the description definers in the specification process to understand an existing system description. Consistency and completeness checking rules defined for a system description language can be applied to check the consistency and completeness of its system descriptions. The Consistency Checking Report and the Completeness Checking Report can be generated and used by the description definers. Description definers can also use a proper transformation language defined by language definers to convert a system description in one language into a system description in another language. The detailed design of MetaPlex is discussed in Chapter 5.

In MetaPlex, some functions at the meta level are made available to description definers. For instance, description definers can formulate a simple completeness checking rule and check incomplete objects on-line. They can also create a dynamic sublanguage from an existing language and then open an editor to interact with a subset of a system description. Making some meta level functions available to description definers provides them with additional flexibility to fulfill unique requirements of a target system.
4.7 User Interface Design

Before we present the design and implementation of MetaPlex in the next chapter, some general user interface design issues are first presented in this section. Since the MetaPlex system will be used by some casual users, user interface design has been an important focus of this research. Because various system components in MetaPlex share a common user interface, MetaPlex should be easily learned and used. Some of the user interface features are discussed from Section 4.7.1 to Section 4.7.6.

4.7.1 Window-based

Multiple and overlapped windows are used in MetaPlex. There are standard operations on a window as a whole. Users can perform these operations by clicking the right button of the mouse when the mouse cursor is inside the window. The standard operations on the window are move, reframe, collapse, and close. These operations let the user move a window around on the screen (i.e., move), change the size of a window (i.e., reframe), reduce a window into an icon which is a bounding box containing only the title of the window (i.e., collapse), and close a window (i.e., close). Flexibility in placing the windows in a location determined by the user allows users to configure the screen layout according to their preferences. The window (called topview in Smalltalk) is used by various editors and browsers. Usually a window is further divided into various panes (called subview in Smalltalk). A pane may contain text, a scrollable list, a form, etc.

Interdependence among panes within a window can be set so that changes made in one pane can be propagated to its dependent panes. These dependent panes can update their display on the screen according to the changes.

Using multiple windows on the screen allows users to open various editors to manipulate objects in a target system, and even to work on more than one target system.
Language definers can define a language and at the same time start defining a target system by using the language. The availability of the system at both levels helps language definers in defining a language.

In MetaPlex, windows are used by browsers or editors to view and interact with the underlying models. A model can be a system description language, a target system description, a list of attributes for an object, relations associated with an object, etc. The standardization of using windows and other user interface components, such as scrollable list views, text windows, dialog boxes, and menus, makes the interface of MetaPlex consistent in all of its subsystems. The consistency in user interface helps users learn to operate the system.

4.7.2 Menu-driven

A menu-driven approach is used extensively in several places. For example, context sensitive pop-up action menus are used to choose the available operations from each pane. Menu selection and templates have been used to avoid typing errors and possible syntax errors. Hierarchical menus like the one shown in Figure 4.5 have been used so that the number of items for menu selection is not limited by the size of the screen. Items in a hierarchical menu are listed according to their alphabetical order. A symbol → will be concatenated to the end of an item’s name when there is a lower level menu below this item. Users can move the cursor to the right of the currently highlighted menu item to get the next level menu. If each level of the menu is set up to contain 20 items, it takes only 3 levels of a hierarchical menu to permit selection from 8,000 items. The maximum number of items for each level of menu has been defined as a system parameter so that the system can be configured according to various screens and the user’s preference.

Menu selection has been used wherever possible to prevent typing errors and to increase productivity when using the system. For example, a new relation can be entered
Figure 4.5: An Example of a Hierarchy Menu
through menu selection without typing in any object names that exist in the system
description. Since the system will only show objects of correct object types, some syntax
and semantic errors in specifying a system also can be avoided.

4.7.3 Graphical Interface and Mouse

Graphical representation is the foundation of most structured methodologies. Graphics
are very useful in representing complex structure relations. A graphical Structure Browser
is used in MetaPlex to show the relations in a system description. Users can examine the
relations connected with an object one type at a time without having to worry about the
layout of the graphic symbols, so a messy picture of really complicated relations can be
avoided. Eventually, users will be allowed to create objects and relations dynamically in
the Structure Browser. Structure Browser is discussed in detailed Section 5.2.5.

Most system operations in MetaPlex are driven by a mouse. A detailed description
of how to use a mouse in the Smalltalk programming environment can be found in
[Goldberg, 1984]. Users can use a mouse to select an item from a menu by clicking on
the item, to scroll a list of items in a scrollable pane by moving the cursor of the scrolling
bar of the pane, to choose an active window by clicking in the window, and to select a
current item in a scrolling list or in the Structure Browser by clicking on it. The use of
a mouse as an input device facilitates a sense of direct manipulation so that the system
should be easier to learn and to remember [Coats and Vlaeminke, 1987].

4.7.4 User Interface Utilities

User interface components have been spread out all over the Smalltalk environment.
Most of them cannot be used directly for the MetaPlex system. The author has created a
class called PlexInterface where many useful interface components are defined as its class
methods. For example, selection ← PlexInterface on: aListOfObjects withHeading: aTitle is the message which is used to ask the user to select an item from a list of objects using a title as a prompt. This approach reduces the effort required to find appropriate methods for user interface programming.

4.7.5 Modeless Operations

The modeless operations principle in user interface design has been followed so that users will not be afraid of making mistakes [Tesler, 1981]. Users can drop most operations or processes at any time by making no selection, by entering a blank string, or by deleting an entry they have already made. The modeless operation encourages users to explore the functions of the system without fear. The Smalltalk environment also provides crash recovery management utilities to assist in recovery from an unexpected system crash [Goldberg, 1984].

4.7.6 On-line operation

Most of the operations in MetaPlex are available on-line, such as on-line query, on-line completeness checking, update information of objects in a target system on-line, defining a language on-line, etc. Some CASE tools allow users to connect objects of incompatible object types in a relation (e.g., an information flow from a data store is received directly by another data store) and only check the consistency in batch mode. MetaPlex checks the consistency of a new relation when a user enters it into the system so that consistent specification is maintained at all times. The completeness checking of a system specification is also available on-line. On-line query of objects and relations is available through a menu-driven dialog. Making functions available on-line improves the response time in performing user requested tasks. This should improve the productivity of analysts and designers.
Chapter 5

Design and Implementation of MetaPlex

In the previous chapter the architecture of the MetaPlex system was presented. The detailed design of MetaPlex is discussed in this chapter. There are three major subsystems in MetaPlex: 1) Language Definition System, 2) System Description System, and 3) Transformation System. In CASE tools, the consistency and completeness checking function is an important feature which is related to both Language Definition System and System Description System. Because of its importance, one section in this chapter is designated for the discussion of the consistency and completeness function in MetaPlex. Issues in implementing and using MetaPlex are also presented at the end of the chapter.

5.1 Language Definition System

There are three major functions in MetaPlex Language Definition System: 1) managing language definitions, 2) defining system description languages, and 3) creating sublanguages. The design of these functions is discussed in the following subsections.
5.1.1 Managing Language Definitions

In order to keep track of all the system description languages defined, a class variable called *Languages* has been defined in the *MetaLanguage* Class to maintain a list of languages existing in the system. Because the Smalltalk system uses garbage collection for memory management, an object without at least one persistent object referencing to it will disappear after the program execution. Instead of sending the *allTheInstances* message to the *MetaLanguage* Class to get all the instances of system description languages, the class variable *Languages* is used to ensure that all the languages defined remain persistent.
The MetaPlex Language Browser as depicted in Figure 5.1 is a browser designed to allow language definers to have access to all of the language managing functions in a window-based tool. The upper pane of the Language Browser is a scrollable pane which displays a list of languages existing in the system. Only the names of the languages are displayed in the Browser, the lower pane displays comments associated with the language highlighted in the upper pane.

All the managing functions of language definitions are available to language definers through the pop-up action menu of the upper pane in the MetaPlex Language Browser:

1. **add a new language.**
   
   Language definers can create a new language definition by giving a unique name to the new language. Comments about a language can be entered through the lower pane in the MetaPlex Language Browser.

2. **rename a language.**
   
   The currently highlighted language can be renamed.

3. **remove a language.**
   
   A highlighted language can be deleted. The system should ask the language definer to confirm whether any system descriptions have been defined in this language, since these also will be deleted if the language has been deleted.

4. **generate Language Syntax Report.**
   
   The Language Syntax Report can be generated by selecting this operation. All the relation types in which an object type is involved in will be cross-referenced under the object type in the report. This report can be used directly as a user manual. Eventually, MetaPlex can *file in* a Language Syntax Report generated by another workstation to create a new language definition. A sample Language Syntax Report can be found in Appendix A.
5. *open Target System Browser.*

A directory of all the target system descriptions defined for each language in the system is maintained. Opening the Target System Browser initiates a listing of all the system descriptions of the currently highlighted Description Language.

6. *open Language Editor.*

The Language Editor will be opened for the currently highlighted description language in the upper pane of the MetaPlex Language Browser.

The Language definer can use the Language Editor to define a system description language. These functions in the Language Editor are discussed in Section 5.1.2.

5.1.2 Defining System Description Languages

The language definer has to describe a set of object types and a set of relation types to define a system description language. Both object types and relation types can be further described by a set of attribute types which will be used to describe the characteristics of the types of objects and relations in a target system description.

The Language Editor is the tool used to define a MetaPlex system description language. The layout of the Language Editor window is depicted in Figure 5.2. The screen dump of a Language Editor is shown in Figure 5.3. The upper left hand pane of the Language Editor is a scrollable pane which contains a list of object types existing in a system description language. Only the names of the object types are displayed in this scrollable pane. The upper right hand pane displays comments or a definition of the currently highlighted object type.

The functions involved in defining a set of object types of a system description language are available through the pop-up action menu of the upper left hand pane:
<table>
<thead>
<tr>
<th>Language</th>
<th>Editor: Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>List Menu</td>
<td>CodeView (Text)</td>
</tr>
<tr>
<td>Object Types</td>
<td>Object type comment or</td>
</tr>
<tr>
<td></td>
<td>Object type definition</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Relation Types</td>
<td>Relation type comment or</td>
</tr>
<tr>
<td></td>
<td>Relation type definition</td>
</tr>
</tbody>
</table>

Figure 5.2: The Layout of the Language Editor Layout
<table>
<thead>
<tr>
<th>Language Editor: PSL</th>
<th>Object Type Name: PROCESS</th>
</tr>
</thead>
<tbody>
<tr>
<td>INPUT INTERFACE OUTPUT</td>
<td></td>
</tr>
<tr>
<td>PROCESS SET</td>
<td></td>
</tr>
<tr>
<td>input-received-by-proc</td>
<td>process-decomposition</td>
</tr>
<tr>
<td>process-decomposition</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Relation Type Name:</th>
<th>process-decomposition</th>
</tr>
</thead>
<tbody>
<tr>
<td>No. of object types groups: 2</td>
<td></td>
</tr>
<tr>
<td>Equivalent Statements:</td>
<td></td>
</tr>
<tr>
<td>PROCESS consists-of PROCESS</td>
<td></td>
</tr>
<tr>
<td>PROCESS is-part-of PROCESS</td>
<td></td>
</tr>
</tbody>
</table>

Figure 5.3: The Sample Screen Dump of the Language Editor
1. **open Attribute Type Browser.**

   By selecting this operation, an attribute type Browser will be opened so that language definers can define a set of attribute types to describe the currently highlighted object type. The attribute type Browser is discussed in Section 5.1.2.1.

2. **add an object type.**

   Language definers can create a new object type in the current language definition by providing a unique name.

3. **rename an object type.**

   Currently highlighted object type can be renamed.

4. **remove an object type.**

   The highlighted object type can be deleted; relation types, which include the object types, will be also deleted. Objects of the deleted object type in all the existing target system descriptions will also be deleted.

5. **comment.**

   Comments associated with the current object type are displayed in the upper right hand pane.

6. **definition.**

   The definition of current object type is displayed in the upper right hand pane.

7. **relation type involved.**

   Relation types in which the currently highlighted object type is involved are listed. This function can be used by language definers to check the completeness of a language. If an object type is not involved in any relation types, it usually implies that some relation types are missing from the language definition. Eventually, a graphical browser can be used to examine the relation types in which an object type is involved.
The upper right hand pane of the Language Editor is a scrollable pane which contains a list of relation types existing in a system description language. Only the names of the relation types are displayed in this scrollable pane. The lower left hand pane is used to display comments or the definition of the currently highlighted relation type.

The pop-up action menu in the lower left hand pane (i.e., a scrollable pane for a set of relation types defined in a language) has similar operations:

1. open Attribute Type Browser.
   By selecting this operation, an attribute type Browser will be opened so that language definers can define a set of attribute types to describe the currently highlighted relation type.

2. add a relation type.
   Through a series of interactive dialog, the system will collect the information needed to define a new relation type. Language definers have to provide the following information to create a new relation type in the current system description language: a unique relation type name, number of object types groups involved in the relation types, the object types in each object types groups, the sequence of object types groups in the equivalent statements, and the connectors used in the equivalent statements.

3. rename a relation type.
   Currently highlighted relation type can be renamed.

4. remove a relation type.
   The highlighted relation type will be deleted. Relations of the deleted relation type in all the existing target system descriptions will also be deleted.

5. comment.
   Comments about the currently highlighted relation type are displayed in the lower
left hand pane.

6. **definition.**

The definition of the currently highlighted relation type is displayed in the lower left hand pane.

The amount of effort involved in maintaining consistency between system description languages and the system descriptions is tremendous. Once a language has been defined and used, the deletion of object types and relation types should be avoided. However, because the object pointers are used for identifying the types of objects and relations in a system description, changing an object type name and a relation type name will not affect the existing system descriptions. Adding new object types and new relation types into an existing System Description Language will not affect the existing system descriptions in the system description language. The dynamics of changing language constructs allow a system description to evolve without growing pains, while in the traditional database systems, changing the database schema means a major restructuring of the existing database system.

New inference engines can be built by changing the implementation of some of the methods without major changes in other existing systems. For instance, in order to support inheritance of attribute types along the *isa* hierarchy of an object type, only the implementation of the `attributeTypeList` method would have to be changed. This could be accomplished by changing the search method from simply returning a list of attribute types defined for an object type to looking up the *isa* relation to find out all the attribute types defined as the super classes of the object type.
5.1.2.1 Defining Attribute Types

Attribute types are used to describe the characteristics of object types and relation types defined in a system description language. In defining an attribute type, the user is asked to specify attribute name, comment, data type, prompt for user's input, occurrence (either single value or multiple value), legal values, and default value. Currently an attribute type can have the following data types: 1) Description, 2) Integer, 3) Integer number, 4) Real number, 5) String. "Description" is used for a string longer than one line while "String" is used for a single line string. An attribute type with "Description" as its data type can be used to capture procedural knowledge in pseudo codes.

The system is designed with an open architecture which allows users to add new data types for attribute type. Any Smalltalk class can be used as a data type as long as it has the following methods implemented: 1) `addValue`, 2) `updateValue: aValue`, and 3) `removeValue`. These methods are used to add a new value to an attribute, update an existing attribute, and remove an attribute. The polymorphism in Smalltalk allows the system builder to implement different methods for the same message pattern. For example, the `addValue` method for an attribute type "C source program" can be implemented as a C compiler either written in Smalltalk itself or implemented as an external code. A Smalltalk class with these message protocols can be registered as one of the legal data types for an attribute type through the method `addDataType: aDataType` which is a class method of `AttributeType`. Table 5.1 is an example of an attribute type defined for the object type PROCESS in PSL.

The Attribute Type Browser, depicted in Figure 5.4, can be accessed from the Language Editor. The upper half pane contains a scrollable list which displays the names of a list of attribute types for an object type or relation type. The lower half pane is used to display comments or the definition of the current attribute type. The functions to maintain a set of attribute types for an object type or a relation type are as follows:
Table 5.1: An Example of an Attribute Type Definition

1. **add an attribute type.**
   A new attribute type can be added to describe the current object type or relation type. The following information is elicited from the language definer through an interactive dialog: 1) a unique attribute type name, 2) the prompt for user input, 3) data type, 4) legal values, and 5) default value.

2. **rename an attribute type.**
   The currently highlighted relation type can be renamed.

3. **remove an attribute type**
   The highlighted attribute type will be deleted. Attributes of this type used in all the existing target system descriptions will also be deleted.

4. **comment.**
   Comments of currently highlighted relation type are displayed in the lower left hand pane.

5. **definition.**
   The definition of currently highlighted relation type is displayed in the lower left hand pane.
Definition of Attribute Type

name: typa
propmt: What is the process type of this PROCESS object?
data type: String
occurrence: #one
legal values: ('system' ' subsystem' 'program' 'module')
default values: 'program'

Figure 5.4: Attribute Type Browser
5.1.3 Creating Sublanguages

In most database management systems, the structure of the database is represented at three levels: internal model, conceptual model, and external model [McFadden and Hoffer, 1988]. The internal model, also called a physical model, describes the internal structures of a database implementation, such as the use of hashing or indexing. The external model represents the users' views of database applications of concern to them. An external model is specified by using a database definition language to define a subschema for a specific user application. The conceptual model represents the integration (or union) of users' views on various applications with a database system. The integration of various external views into a conceptual model enforces consolidation, completeness, and consistency in the database design [McFadden and Hoffer, 1988]. A similar concept has been used in designing the sublanguage creation facility in MetaPlex.

Why do users need to have the ability to define a sublanguage to interact with a system description? Complex language constructs in CASE tools such as PSL/PSA, which has about 17 object types and 66 relation types available to describe a target system, makes the definition of a system description very difficult, if not impossible. This is why Teichroew and Hershey [1977] suggest dividing system descriptions in PSL/PSA into eight major system aspects: 1) System Input/Output Flow, 2) System Structure, 3) Data Structure, 4) Data Derivation, 5) System Size and Volume, 6) System Dynamics, 7) System Properties, and 8) Project Management.

For example, the System Input/Output Flow aspect of a system looks only at the interaction between a target system and its environment, i.e., the top level information flow crossing the system boundary. In terms of language construct, users are allowed to view and manipulate only the relations among the following five types of objects: INPUT, OUTPUT, INTERFACE, SET, and PROCESS.

A sublanguage in MetaPlex is a subset of a complete language. A sublanguage's
relationship to a language in MetaPlex is comparable to that of a subschema to a schema in database systems. The creation of sublanguages provides a mechanism for allowing users (both as individuals and as groups) to describe the system from certain aspects without being bothered by the complexity of a complete language.

There are two ways to define a sublanguage. First, it can be defined by language definers. A language definer first extracts object types from an existing language and then selects from the existing language relation types among the extracted object types. The sublanguage defined will be made available to the system definer, who can use the sublanguage to interact (i.e., view, add, update, and delete objects and relations) with a system description defined by the language from which the sublanguage is derived. Second, a system definer can define a sublanguage dynamically while interacting with a system description. Once the sublanguage has been defined, the system will open up a target system browser which allows the system definer to interact with the underlying system description by means of the dynamically generated sublanguage. The system does not save dynamically generated sublanguages.

Using a sublanguage in MetaPlex simplifies the individual description definer's work in defining a system. The sublanguage capability in MetaPlex makes it easier for a group of description definers to work together on the same target system. Using sublanguages which describe the same system from different aspects, members of a group of description definers can contribute their knowledge about the system from their own perspectives.

5.2 System Description System

MetaPlex can automatically generate a generic environment for specifying descriptions of target systems, i.e., the System Description System. Customized user interface tools for system specification can also be built. The System Description System is used by the
description definer to specify descriptions of target systems according to a certain system description language. A system description consists of a set of objects and relations defined in terms of object types and relation types in the system description language used.

The System Description System has two major functions: 1) managing a list of target system descriptions for each existing system description language and 2) defining a system description according to a certain system description language.

5.2.1 Managing Target System Description

Through the MetaPlex Language Browser, the description definer can open the Target System Browser, which contains a list of target system descriptions in a certain description language.

Target System Browser, as depicted in Figure 5.5, is a browser designed to allow the description definer to access all the system descriptions defined in a description language. The upper pane of the Target System Browser is a scrollable pane which displays the names of system descriptions in a description language. The lower pane displays comments on the system description highlighted in the upper pane.

All the high level managing functions of system descriptions are available to the description definer through the pop-up action menu of the upper pane in the Target System Browser which makes it possible to:

1. add a target system.
   Description definers can create a new target system description by providing a unique name to the new system. Comments can be entered in the lower pane of the browser.
Figure 5.5: Target System Browser
2. **rename a target system.**

The name of currently highlighted system description can be renamed.

3. **remove a target system.**

A highlighted system description can be deleted. The system should confirm with the description definer that the delete action is desired.

4. **report.**

By selecting this operation, description definers can ask the system to generate either the System Description Report or the Completeness Checking Report. The report functions are discussed in Section 5.2.3.1.

5. **open Target System Editor.**

The Target System Editor will be opened for the currently highlighted target system description in the upper pane of the Target System Browser.

Description definers can use the Target System Editor to access all the functions needed to define a system description. These functions and the Target System Editor are discussed in the next subsection.

### 5.2.2 Defining System Descriptions

The basic functions involved in defining a system description are maintaining the definitions of a set of objects and of a set of relations. Both objects and relations can be further described by a set of attributes, which is used to describe the characteristics of objects and relations in a target system description. The functions to specify a system description are made available to the description definers through the Target System Editor.

Target System Editor is a tool used to specify system descriptions in MetaPlex. The layout of the Target System Editor window is depicted in Figure 5.6. The screen
Figure 5.6: The Layout of the Target System Editor
dump of a Target System Editor is shown in Figure 5.7. The upper left hand pane of the Target System Editor is a scrollable pane which contains a list of objects existing in a system description. Only the names of the objects are displayed in this scrollable pane. The upper right hand pane displays comments or the definition of the currently highlighted object.

The functions for defining a set of objects of a system description are available through the pop-up action menu of the upper left hand pane that makes it possible to:

1. open Attribute Browser.

Selecting this operation opens an Attribute Browser so that the description definer can define a set of attributes to describe the currently highlighted object. The Attribute Browser is discussed in Section 5.2.2.1.
2. *add an object.*

Description definers can create a new object in the current system description by giving a unique name. The system will ask the description definer to select from a set of object types defined in the description language the object type of the new object. There will be no undefined object in the system. The determination of an object type for a new object is not a trivial task, especially when there are many object types in a system description language. A rule-based expert system has been designed to help the description definer in this regard. The discussion of this rule-based system design can be found in Section 5.2.6.

3. *remove an object.*

The highlighted object can be deleted. Relations which include the object also will be deleted.

4. *rename an object.*

The currently highlighted object can be renamed.

5. *comment.*

Comments on the currently highlighted object in the upper right hand pane can be displayed.

6. *definition.*

The definition of currently highlighted object in the upper right hand pane can be displayed.

7. *add a new relation.*

The system allows a relation of one of the relation types with which the currently highlighted object can be involved to be added.

8. *query on objects.*

Description definers can perform on-line queries based on the object type of objects and constraints on attributes’ values. This function is discussed in detailed in
Section 5.2.4.


The relations in which the currently highlighted object is involved can be browsed through by using the Structure Browser. Details of the Structure Browser are discussed in Section 5.2.5.

10. *completeness checking.*

The completeness checking function can be accessed by selecting this operation. Detailed design and operation are discussed in Section 5.4

The lower left hand pane of the Target System Editor is a scrollable pane which contains a list of the names of relations existing in a system description. The lower right hand pane is used to display comments or the definition of the currently highlighted relation. The pop-up action menu in the lower right hand pane (i.e., a scrollable pane displaying a set of relations defined in a system description) has similar operations:

1. *open Attribute Browser.*

By selecting this operation, an Attribute Browser will be opened so that description definers can define a set of attributes for the currently highlighted relation.

2. *add a relation.*

The system will first ask the description definer to select the type of relation to create. The system will show the description of the relation type to the description definer for confirmation. The description definer will then be asked to specify a set of objects in each object group involved in a relation according to the relation type of this relation. There are two ways to enter the objects involved in a relation. When all the objects of the new relation already exist in the system, a menu-driven mode will be used. In this mode, the system will show only objects of the correct object type(s) in each group and ask the description definer to select one or more objects according to the occurrence of objects in the group (i.e., #one or #many).
In a command mode, the system will display a template and ask the description definer to enter appropriate object names. The system will check to make sure that all the objects entered are of correct object types. If there is more than one object type possible for a new object entered in a command mode, the system will ask the user to select one object type from all the legal object types. If there is only one object type corresponding to a newly entered object, MetaPlex will automatically assign that object type.

3. rename a relation
   The currently highlighted relation can be renamed.

4. remove a relation.
   The currently highlighted relation can be deleted.

5. comment.
   Comments on the currently highlighted relation in the lower right hand pane will be displayed.

6. definition.
   The definition of the relation currently highlighted in the lower right hand pane will be displayed.

5.2.2.1 Attribute Browser

The upper pane in the Attribute Browser window displays a scrollable list of attributes of an object or a relation selected from the Target System Editor. Users can switch the contents of the lower pane in the Attribute Browser to show either comments on the attribute type or the value of the selected attribute. A question mark (?) is concatenated to the end of an attribute name to remind users that the value of the attribute has not yet been defined.
Figure 5.8: Attribute Browser
5.2.3 Report and On-line Query Facilities

Various report and query facilities have been designed and implemented to help description definers understand a target system description.

5.2.3.1 Report Facilities

Report facilities in System Description System can be used to generate the following reports:

1. The System Description Report.
   All the relations in which an object is involved will be cross-referenced and listed under the object in the System Description Report. The System Description Report is similar to the Formatted Problem Statements used in PSL/PSA. This report can be used to check the contents of the system description. Eventually, MetaPlex can file in a partial System Description Report generated by one description definer to integrate it with another system description. However, conflict resolution strategies should be used to detect redundant and inconsistent information in two merged system descriptions. Consideration of this issue is not within the scope of this dissertation. A sample System Description Report can be found in Appendix B.

   The Completeness Checking Report can be generated according to incompleteness checking rules defined for the system description language.

   All the free-standing objects (i.e., objects not involved in any relation) will be reported in the Free-standing Object Report. Listing as a free-standing object usually implies that some relations are missing in the system description. Completeness
Checking Report and Free-standing Objects Report are discussed further in Section 5.4.

Besides these standard reports, language definers who know how to program in Smalltalk can also write code in Smalltalk to generate a customized report. The Smalltalk program can be saved in conjunction with a system description language and can be called by description definers to generate the report.

An instance method which has been defined for all the target system descriptions (i.e., all the instances of TargetDB Class) can execute a piece of Smalltalk code to generate a report. Language definers can use this feature to define Smalltalk methods to generate customized reports for target systems of a specific language. Language definers who do not know the Smalltalk programming environment or understand the detailed internal implementation of MetaPlex can still access all of the class and instance variables as well as instance methods defined for the TargetDB.

5.2.4 Query Function

On-line query of objects and relations is available through a menu-driven dialog. Objects and relations in a system description are displayed in two different panes in the Target System Editor window. They are listed in alphabetical order in a scrollable subview. However, when a user wants to find objects or relations of a certain object type or relation type, an on-line query function is available through the pop-up action menu from these panes in the Target System Editor, called object query and relation query, respectively.

To formulate a query, users can specify object types or relation types as well as certain conditions on the attributes of objects and relations. The system will list the objects or relations that satisfy these criteria in a hierarchy menu. The user can select one
of these objects, and then choose to open the Attribute Browser of the selected object or highlight the chosen object in the Target System Browser. The on-line query function will help users find the objects or relations they want. A special Structure Browser has been designed to help description definers examine relations associated with an object. This Structure Browser provides an alternative for on-line query. The functions of Structure Browser are discussed in Section 5.2.5.

5.2.5 Structure Browser for Relations

MetaPlex provides a generic graphical browser (i.e., Structure Browser) which supports a user's navigation of relations among connected objects within a system description. Description definers need not know the syntax and operations of any query language. Although Structure Browser provides the viewing of one type of relation at a time, it is considered to deal with relations better than other types of systems, such as relational database systems and hypertext systems. In relational database systems, foreign keys and primary keys are used together to connect relations between entities. Because the relations between foreign keys and primary keys are not represented explicitly, the navigation through relations among entities has to be handled by programming. In most hypertext systems, only binary relations are allowed.

The Structure Browser function is available within the System Description Editor (also called Target System Editor). A system definer first moves the cursor to the objects list pane in the System Description Editor and highlights one object by clicking the mouse on it. Clicking the middle button of the mouse will then cause an action menu to appear. The system will show the system definer all the possible relation types that the current object can be involved in and ask the system definer to select one type of relation for further navigation. The Structure Browser program will then open a window and show all the instances of the selected relation type in a graphical format as depicted in Figure 5.9.
process-decomposition => PROCESS has-subparts: PROCESS

order-processing-system(PROCESS) => order-entry-program(PROCESS) => enter-order(PROCESS) => order-validation(PROCESS)

order-processing-system(PROCESS) => update-order-db(PROCESS)

order-processing-system(PROCESS) => reporting-program(PROCESS) => generate-customer-blol(PROCESS) => generate-customer-report

input-receiving => PROCESS receives: INPUT

order-entry-program(PROCESS) => customer-order(INPUT)

order-entry-program(PROCESS) => order-adjustment-form(INPUT)

Figure 5.9: Structure Browser
Each object displayed in the Structure Browser will be annotated with its Object Types and surrounded by a rectangle (i.e., a bounding box). A scrolling function available within the Structure Browser allows the user to view the global structure of the relations and to select certain parts of it for examination in the Structure Browser window. Users can select any object in the Structure Browser window and open up another Structure Browser window to examine relations of the selected object. Navigation through a system description can be continued in this fashion.

5.2.6 Using Rules to Determine Object Types

Whenever users want to create a new object, they first must identify the type of object it is to be. Once the type of the object is known to the system, users of the system will be able to use the information defined at the language level to guide the rest of the system definition process.

However, for a complicated system description where many object types are used in describing a system, users may be unable to decide the object type of the object in mind. For instance, 17 object types are available in PSL and it is sometimes difficult to differentiate among them. SET is only differentiated from ENTITY in that SET is used to define a physically existing data set and ENTITY is used to describe a logical entity which could be described by some data. The object types INPUT and OUTPUT are used to describe data that flow across the system boundary while data within the system boundary are called either SET or ENTITY.

A backward chaining rule-based system therefore has been designed to help MetaPlex users determine the object types of objects. Using Data Flow Diagram as an example, the set of rules listed in Table 5.2 can be used to determine the object types of an object in DFD.
IF the object is data = true AND
the object is stored in a file or database = true
THEN the object type of the object = DATA STORE

IF the object is data = true AND
the object is stored in a file or database = false
THEN the object type of the object = DATA FLOW

IF the object exists outside the system boundary = true AND
the object can generate or receive data
from the system = true
THEN the object type of the object = EXTERNAL ENTITY

IF the object exists inside the system boundary = true AND
the object can generate or receive data
from the system = true
THEN the object type of the object = PROCESS

Table 5.2: Sample Rules for Determining the Object Types of an Object in a DFD Language
5.3 Transformation System

Language definers of the MetaPlex system may create system description languages to describe a target system in each phase of the system lifecycle. A transformation system is needed to convert a target system description defined in one language into another. Since MetaPlex languages are defined in terms of the same knowledge primitives at the axiomatic level, the specifications of transformation languages can be defined in a descriptive rule format. The transformation mechanism provides a common ground for information sharing of system descriptions defined in various languages and for the integration of GDSS and CASE.

5.3.1 The Necessity of a Transformation System

Top managers are concerned with organizational goals and objectives, competitors and strategic assumptions; information systems developers are concerned with the detailed information required for the design and implementation of target systems. Information captured in one phase should be able to be carried over at least partially to another phase in which a different language is used. This creates the need to develop a transformation system.

MetaPlex allows users to define a transformation language between any two languages (e.g., Language A and B) existing in the system. For example, a transformation language “A→B” can be defined to convert a system description in language A into a system description in language B. Language A is the source language and language B is the target language. A transformation language “B→A” also can be defined to transform a system description in language B into a system description in language A. The ability to define a transformation language between two languages in both directions allows MetaPlex to support both forward engineering and reverse engineering in the system de-
velopment process. The system can use an instance variable of a newly created object to store an object pointer to the object in the source system description so that the system can trace back to the source of a transformed object. This provides a basis for traceability in system descriptions throughout the system life cycle.

5.3.2 The Architecture of the Transformation System

Figure 5.10 shows the architecture of Transformation System, which consists of the Transformation Language Definition System and the System Description Transformation System. Language definers use the Transformation Language Definition System to define a transformation language between any two existing system description languages. For example, a transformation language which transforms a BSP system description into a PSL system description can be defined. BSP is the source language and PSL is the target language. By default the system will name the transformation language as “BSP→PSL”.

A transformation language can be used by the System Description Transformation System, which transforms objects and relations in a system description of the source language into a system description of the target language. For example, BSP has been used to define a target system called Order Department. The user can use the System Description Transformation System to transform the system description into a new system description in PSL by using the “BSP→PSL” transformation language defined.

5.3.3 Transformation Languages

A system description in MetaPlex consists of objects and relations of certain types determined by the language it uses. Since the structures and the transformation algorithms of objects and relations are different, two types of rules are used to specify the transformation of objects and relations. Language definers should define object transformation
Figure 5.10: The Architecture of the Transformation System
IF

Object Type of an object is INFORMATION SYSTEM and
this object’s attributes’ values meets the following conditions:
mode IN OrderedCollection (’on-line’)

THEN

convert the object into an instance of PROCESS
with attribute-value pair(s) as:
on-line = true

Table 5.3: An Example of Object Transformation Rule

rules and relation transformation rules in a transformation language separately. In the
condition part of an object transformation rule, the language definer specifies the object
type of objects to be transformed from system descriptions in the source language and
then specifies the criteria of the values of an object’s attributes for the transformation.
In the action part of the rule, the language definer specifies the object type of the new
object to be created in the target language as well as the attribute values that the new
object should have. Table 5.3 is an object transformation rule which converts an ‘on-line’
INFORMATION SYSTEM object defined in BSP into an ‘on-line’ PROCESS object in
PSL.

The syntax for a relation transformation rule is similar to that of the object transfor-
mation rule. Table 5.4 shows an example of a relation transformation rule so defined as to
convert an information-system-decomposition relation in BSP to a process-decomposition
relation in PSL.
IF

relation type of a relation is information-system-decomposition:

INFORMATION SYSTEM consists-of: INFORMATION SYSTEM

THEN

convert the relation into an instance of process-decomposition:

PROCESS has: PROCESS

Table 5.4: An Example of Relation Transformation Rule

5.3.4 The Design of the Transformation System

Figure 5.11 is the Transformation Language Editor in which a list of object translation rules (shown in the upper left hand pane) and relation translation rules (shown in the lower left hand pane) have been defined to translate a BSP system description into a PSL system description.

The creation of both object transformation rules and relation transformation rules is accomplished through a menu-driven dialog that not only helps the language definers by preventing their misspelling of names but also relieves them from having to remember the exact syntax of rules. The procedures for defining an object transformation rule are as follows:

1. Select the operation *add object transformation rule* from the pop-up action menu of the upper left hand pane in the Transformation Language Editor.

2. The system will display a list of all the object types of the source language in a menu. The language definer then can select one object type from the menu.
Figure 5.11: Transformation Language Editor
3. Once an object type has been selected, the system will ask the language definer to specify constraints imposed on the attributes' values. The language definer will first select one attribute type from a list of attribute types for this object type. The system will query the condition on the value of the selected attribute types. If the data type of the attribute type is Boolean, the system will ask the language definer to specify whether the value is true or false. If the data type of the attribute type is Integer or Real number, the system will ask the language definer to specify the relation operator first (i.e., =, <>, >, <, >=, <=) and then specify the range of its value. If the data type of the attribute type is "String" with legal values then the system will ask the language definer to select values from its legal values as part of the condition (e.g., `system-type IN ('program', 'system'). Constraints on more than one attribute type can be used for the condition and action parts of a rule.

4. To construct the action part of a rule, the system will display all the object types of the target language in a menu. The language definer can then select an object type for the new object in the target language.

5. A procedure similar to procedure 2 is then used to specify the values of the attributes that the newly converted objects should have. This completes the specification of the action part of the rule.

Figure 5.12 displays a user's interaction with the Transformation Language Editor in defining an object transformation rule as shown in Table 5.3. Currently the system does not check the consistency among rules, for which the language definer must be responsible.

5.3.5 The Design of System Description Transformation System

The System Description Transformation System function is available to the description definer through the Target System Browser. The user first selects the target system
Select an Object Type from the source language: BSP

(a) BUSINESS SUBSYSTEM  DATA SET  INFORMATION SYSTEM  ORGANIZATIONAL ENTITY

Select an attribute for the condition of transforming an INFORMATION SYSTEM

(b) importance  mode  stage

Choose the value(s) of importance

(c) batch  manual  on-line

Select an Object Type from the target language: PSL

(d) INPUT  INTERFACE  OUTPUT  PROCESS  SET

Select an attribute for the action of transforming to a PROCESS

(e) on-line  response-time  type

Is the value of on-line true?

(f) YES  NO

Figure 5.12: Menu-Driven Dialogs for Defining an Object Transformation Rule
to be transformed from the upper pane of the Target System Browser. By clicking the middle button of the mouse to activate an action menu and selecting the operation called transform, the user can start the transformation function. The System Description Transformation System will first check all the transformation languages and list the transformation languages whose source language is the same as the language used for the highlighted system description. If more than one transformation language is applicable, the description definer will be asked to select one of the languages from a menu and provide a name for the new system description to be created. The System Description Transformation System will start using the transformation rules in the selected transformation language for the transformation. Object transformation rules will first be applied to transform all objects which satisfy the condition part of a rule to new objects in the new system description according to the action part of the rule. The name of the old object will be preserved in the transformation process.

The transformation of relations is more difficult than the transformation of objects. System Description Transformation System first checks whether a relation's type and the values of its attributes satisfy one of the relation transformation rules defined in the transformation language. If the condition part of a relation transformation rule is evaluated to be true, the system will further check to see whether all the objects involved in the original relation have been transformed to the new system description. Only when all the objects involved in the relation have been converted will the system proceed with creation of a new relation in the new system description according to the action part of the relation transformation rule. For example, the system would not transform a relation "A consists-of: B, C, D" defined in one language into a relation "A' decompose-into: B', C', D'" in another language if any one of the objects in the relation (i.e., A, B, C, D) is not converted by the object transformation rules. The reason for the checking is to make sure that the relation transformation is consistent with the object transformation.
5.4 Consistency and Completeness Checking in MetaPlex

Consistency checking and completeness checking are functions which help description definers identify potential errors or missing information in a system description. As described in Section 4.6, MetaPlex handles these functions at two levels. Language definers specify rules to check consistency and completeness of system descriptions defined in a system description language. Description definers can apply these rules to the consistency and completeness checking inference engine to generate consistency and completeness checking reports. The design of consistency checking and completeness checking functions in MetaPlex is presented in this section.

5.4.1 Consistency Checking

A system specification must be internally consistent. An inconsistent specification is not caused by syntax errors of the description language but by semantic errors that occur under certain situations. Level balancing is an example of consistency properties of a Data Flow Diagram specification [Howden, 1982; Index Technology Co., 1986]. Consistency checking can be handled in various ways:

1. Static Analysis: The structural validity of a specification that has been created in a batch mode can be checked. Examples of illegal connections in a data flow diagram are: DATA STORE and EXTERNAL ENTITY can not connect to themselves; DATA STORE can not directly connect to EXTERNAL ENTITY; EXTERNAL ENTITY generates a DATA FLOW that is directly received by DATA STORE [Index Technology Co., 1986]. The Verification Report in Excelerator can be used to examine the legality of connections in a data flow diagram [Index Technology Co., 1986].
IF the object type of the object X = INPUT AND
the volume (unit/day) of X >= 1000 AND
the object type of the object Y = PROCESS AND
Y receives: X AND
the response-time (second) of Y >= 30
THEN inconsistency type = 'Unreasonable response time of a
PROCESS Y with a high volume INPUT X'

Table 5.5: An Example Rule for Consistency Checking

2. Dynamic Checking: Some CASE tools, such as SYLVA [CADWARE, 1988], can
perform "dynamic checking" of user input when they enter a relation. These tools
can detect an inconsistent relation, give users an error message, and drop the
incorrect entry or make suggestions for correction.

In MetaPlex, the author took a further step, so that most inconsistencies are pre­
vented by allowing users to enter only legal statements. For instance, a menu-driven
dialog is used to enter a relation of an existing object. The MetaPlex system lists only
objects of correct object types involved in this type of relation for users to choose from
so that entries of illegal statements are prevented.

For more complicated situations, consistency checking rules can be used to check
the inconsistency of a system description. Table 5.5 is an example rule for consistency
checking on a system description in PSL. An Inconsistency Report can be generated by
applying such rules against a system description.
5.4.2 Completeness Checking

Completeness checking of a system description can identify any incompleteness of a system specification. In MetaPlex, the completeness checking rules are defined at the language definition level. The language definer can define a set of completeness checking rules which check the possible incompleteness of system descriptions in a description language. Table 5.6 presents a set of completeness checking rules for a DFD specification.

For the first rule, the system checks whether an EXTERNAL ENTITY is either generating or receiving a DATA FLOW. It is possible for an EXTERNAL ENTITY only to generate DATA FLOW or only to receive a DATA FLOW. For the second rule, the system will check to make sure that a DATA STORE has both incoming DATA FLOW and outgoing DATA FLOW to be logically complete. A DATA STORE which only receives DATA FLOW but does not generate any DATA FLOW to be used by a PROCESS is incomplete. An example of a Completeness Checking Report is shown in Figure 5.13 in which incomplete objects are listed under each completeness checking rule. Users may use this report to identify information missing from a system description.

The Target System Editor also has an on-line completeness checking function which can display all objects in the system description that are not involved in any relation with other objects, i.e., are free-standing objects. The system will list these "free standing" objects in a menu and users can select one of them to create a relation with the selected object.

Completeness checking in the Target System Editor has the following functions: 1) check incomplete objects on-line, 2) generate incomplete objects reports, 3) check free-standing objects on-line, and 4) generate free standing objects report.

The procedure for checking the incomplete objects on-line is: 1) the system first asks the user to select an object type from a list of object types for completeness checking,
1. IF

the object type of an object = EXTERNAL ENTITY AND

the object is not involved in relations

[EXTERNAL ENTITY] generate: [DATA FLOW] OR

[EXTERNAL ENTITY] received: [DATA FLOW]

THEN

incompleteness type = 'EXTERNAL ENTITY is not generating or receiving any DATA FLOW'

2. IF

the object type of an object = DATA STORE

the object is not involved in relations

[DATA STORE] generates: [DATA FLOW] received-by:

[PROCESS] AND

[DATA STORE] received: [DATA FLOW] generated-by:

[PROCESS]

THEN

incompleteness type = 'DATA STORE is not generating and receiving any DATA FLOW'

Table 5.6: Examples of Completeness Checking Rules of a Data Flow Diagram
Target System Name: ORDER PROCESSING SYSTEM
System Description Language Name: PSL

Incomplete Type: PROCESS object does not involve the relation -->
PROCESS receives: INPUT

Description: : A PROCESS object does not receive any INPUT object.

Objects of this type of incompleteness are as follows:
order-file-update
order-reporting-programs

Incomplete Type: PROCESS object does not involve the relation ==> 
PROCESS has: PROCESS

Description: : A PROCESS object has no decomposition (i.e., no subprocess).

Objects of this type of incompleteness are as follows:
order-entry-program
order-file-update
order-reporting-programs

Figure 5.13: A Completeness Checking Report of a System Description
2) all the relation types in which the selected object type is involved are listed for selection, 3) all the objects of the selected object type which do not have an instance of the selected relation type are listed as incomplete objects, and 4) the system will initiate an operation to generate an instance of the relation type.

Figure 5.14 displays a series of menu-driven dialogs for checking completeness on-line. After the user has selected the complete checking operation from the action pop-up menu in the target system editor, the menu shown in Figure 5.14(a) will show up. The user is asked to select one of the following completeness checking functions: check incompleteness on-line, generate incompleteness checking report, check free standing objects on-line, or generate free standing objects report. If the user chooses to check the completeness on-line, the pop-up menu depicted in Figure 5.14(b) will show up. The user is asked to choose one of object types defined in the system description language used by the target system. If the user chooses “PROCESS”, the menu depicted in Figure 5.14(c) will pop up. The user is asked to select one of the relation types in which “PROCESS” objects are not involved. A list of incomplete objects specified from this series of menu-driven dialogs will be displayed in a menu as depicted in Figure 5.14(d). Eventually the user will be able to select one of the incomplete objects and to instantiate a proper relation to remove the incompleteness of the selected object.

5.5 Implementation Issues

In this section, the author first discusses the reasons for selecting the Smalltalk programming environment as the software platform for the prototype system built, after which the limitations of the implementation environment will be discussed.
Select one of the completeness checking functions

<table>
<thead>
<tr>
<th>Check Completeness On-line</th>
</tr>
</thead>
<tbody>
<tr>
<td>Generate Completeness Checking Report</td>
</tr>
<tr>
<td>Check Free Standing Objects On-line</td>
</tr>
<tr>
<td>Generate Free Standing Objects Report</td>
</tr>
</tbody>
</table>

(a)

Select one Object Type for completeness checking

- INPUT
- INTERFACE
- OUTPUT
- PROCESS
- SET

(b)

Check a relation in which PROCESS objects are not involved

- input-received-by-process
- output-generated-by-process
- process-decomposition

(c)

PROCESS objects do not involve the relation: 'PROCESS receives: INPUT'

- order-entry-program
- order-file-update
- order-reporting-program

(d)

Figure 5.14: Menu-Driven Dialogs for On-Line Completeness Checking
5.5.1 The Selection of Software Platform for Implementation

Selecting a software platform for a system development research project is an important implementation decision. The basic criteria for such a decision are:

1. The semantic gap between a software platform and the underlying system design. The software platform which has the most direct mapping to the design of the system should be chosen.

2. Flexibility and power. Usually there is a trade-off between the flexibility and the power of a software platform. Language systems are flexible but using them to implement a system takes a long time. Software environments or tools, such as an expert system shell, are powerful in terms of their having shorter implementation time. However, these tools are designed with specific types of applications in mind so that they are less flexible than language systems in dealing with specific implementation requirements. A high level software platform should be selected, provided it can be used to implement all the functions needed.

3. Transportability and performance. Depending on the focus of the research, the transportability and performance issues of the software platform must sometimes be considered. These issues become very important when the system has to handle a large amount of data or many calculations or when it has to be run on various hardware platforms.

4. Availability. Practically, the selection of a software platform is also determined by its availability. This includes the cost and time involved in setting up and learning about the system.

The Smalltalk programming environment has been chosen to implement the current MetaPlex system because: 1) Smalltalk has been suggested as an ideal prototyping tool [Diederich and Milton, 1987], 2) Smalltalk allows dynamic binding, which makes changes
of data structures design less painful, 3) the symbolic debugging facility and the inspect browser make the debugging process much easier [Goldberg, 1984], 4) the object-oriented programming style provides a basis for enforcing software engineering principles, such as information hiding, modularity, and abstraction, 5) the generic user interface framework in Smalltalk provides a solid ground for the interface design and implementation in MetaPlex, 6) the memory management function is managed by the garbage collection mechanism in the Smalltalk environment so less effort is required to manage memory, and 7) the Smalltalk system is built on top of a Smalltalk virtual machine. Applications built in Smalltalk can be easily transported to Sun workstation, IBM PC/AT, Apple Macintosh and other hardware platforms.

The following discussion explains how Smalltalk as an object-oriented language supports software engineering principles in the implementation of a software system:

1. Information hiding. Information hiding is a technique of “encapsulating software design decisions in modules in such a way that the module’s interfaces reveal as little as possible about the module’s inner workings” [IEEE, 1983]. In Smalltalk language, the data structures of an object (i.e., instance variables) are hidden from the external entities (i.e., message senders) because external entities are allowed to access or update the status of the object only through sending messages. The internal implementation complexity (i.e., the implementation of methods) is hidden away from the external entities. Users of an object have to know only the interface of the object (i.e., message protocol) in order to use it.

2. Abstraction. The abstraction concept applies to both data and procedures. In an object-oriented language, abstraction is supported by the class hierarchy. Objects or subclasses can inherit data (i.e., instance variables) and procedures (i.e., methods) from their classes or super classes. Abstraction promotes the reusability of code and facilitates top-down design.
5.5.2 Limitation of the Implementation Environment

Using Smalltalk does not come without cost. For objects to remain persistent in the Smalltalk system, they have to be stored as objects in the system image. The system uses up main memory very quickly when the system starts to grow. There are two major approaches to dealing with the storage of large amount of information: 1) developing fileIn and fileOut utilities for language definitions and target system descriptions so that only the information actually required is loaded into the system, 2) using an object-oriented database to store objects and retrieve them as needed. GemStone is such an object-oriented database system which is built on top of the Smalltalk programming environment. Issues involved in making Smalltalk a database system are discussed in [Coperland and Maier, 1984]. As a prototype system, MetaPlex does not deal with this problem.

5.6 Issues in Using MetaPlex

There are two major issues that need to be addressed in using the MetaPlex system: how to define a system description language and how to select a system development method for the specification of a target system.

5.6.1 Methods in Defining MetaPlex Languages

MetaPlex has a user interface that enables language definers to define languages at the meta level; no intelligent support in defining languages is provided. However, defining a language in MetaPlex is not a trivial task. Knowledge of a domain and an understanding of methodologies applicable in a domain are essential for defining a useful language. There are three methods for defining languages in MetaPlex: 1) analyzing structured methods, 2) conducting domain analysis, 3) deriving from a prototypical system descrip-
tion. Applications of these methods are not mutually exclusive. Each method is discussed in detail from Section 5.6.1.1 to Section 5.6.1.3.

5.6.1.1 Analyzing Structured Methods

Most structured methodologies for information system planning, analysis, and design, as well as methods for group argumentation, use certain structured representations to capture information generated. By analyzing the basic representation schemes, the language definer can map the representation constructs used by a method to the object types and relation types as well as to attribute types used in describing object types and relation types.

For example, in Structured System Analysis [Gane and Sarson, 1980], there are four basic symbols: DATA STORE, DATA FLOW, PROCESS, and EXTERNAL ENTITY. They can be defined as object types in MetaPlex language. The explosion or leveling of PROCESS can be defined as a relation type called processing-decomposition. The composition of DATA STORE can be defined by data-store-decomposition and group-item-decomposition. There are also relation types which define the generation and reception relations of DATA FLOW to PROCESS, EXTERNAL ENTITY, and DATA STORES. Table 5.7 lists some of the relation types for Structured System Analysis.

5.6.1.2 Conducting Domain Analysis

One of main thrusts in reusability is to expand the reusability concept beyond programs to the whole system lifecycle, especially to the reuse of expertise in design and analysis. The domain analysis method used in the Draco Project has been identified as a major approach that promotes reusability in analysis and design phases. When no structured method appropriate for defining target systems is available in a domain, carefully selected
<table>
<thead>
<tr>
<th>Name: process-decomposition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Equivalent statements:</td>
</tr>
<tr>
<td>[PROCESS] consists-of:</td>
</tr>
<tr>
<td>[PROCESS] is-part-of:</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name: data-store-decomposition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Equivalent statements:</td>
</tr>
<tr>
<td>[DATA STORE] consists-of:</td>
</tr>
<tr>
<td>[GROUP ITEM, DATA ITEM] is-part-of:</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name: group-item-decomposition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Equivalent statements:</td>
</tr>
<tr>
<td>[GROUP ITEM] STORE] consists-of:</td>
</tr>
<tr>
<td>[GROUP ITEM, DATA ITEM] is-part-of:</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name: process-generate-data-flow</th>
</tr>
</thead>
<tbody>
<tr>
<td>Equivalent statements:</td>
</tr>
<tr>
<td>[PROCESS] generate:</td>
</tr>
<tr>
<td>[DATA FLOW] generated-by:</td>
</tr>
</tbody>
</table>

Table 5.7: Sample Relation Types for Structured System Analysis
experts in the domain can be used to define a domain-specific system description language.

The author suggests that a group decision support system, such as Plexsys Planning Tools, can be used by such domain experts to define a language. The procedure in defining a language by using the Plexsys Planning Tools is: 1) Identify experts in the domain of analysis. 2) In a face-to-face language definition session, ask these experts to list concepts, things, events, etc., in the domain by using the Electronic Brainstorming System. An EBS comment file will be generated. 3) Use Issue Analysis tools to identify and consolidate the EBS comments into a list of object types. 4) Based on the list of identified object types, ask the domain experts to use the EBS tool again to generate possible relation types among object types. 5) Through the Issue Analysis process, relation types will be identified and consolidated. 6) Have the experts use the Topic Commenter to generate attribute types for both object types and relation types. 7) Once these steps have been completed the language definer can use the output (i.e., a list of object types and a list of relation types, as well as their attribute types) from these GDSS sessions to create a MetaPlex language.

Through domain analysis, the language definer can create domain-specific languages for the analysis or design of systems in the domain. Capturing experts' analysis and design knowledge in a domain-specific language facilitates the reuse of analysis and design expertise. A domain-specific language should work better for the description definer in defining a target system.

5.6.1.3 Deriving from a Prototypical System Description

When no structured method is appropriate and no expert is available for defining a specification language, a prototypical system description can initially be described by a description definer without using a language. The language definers can then work with the description definer to classify the objects and relations used in the system description
into object types and relation types. Currently MetaPlex does not directly support this prototypical approach to deriving a language.

5.6.2 Selection of System Development Methods

It is important that proper system development methods are selected to be encoded into system description languages. The following criteria are suggested for determining what system development methods should be used in defining new system description languages:

1. The knowledge representation requirements. There are more than a hundred structured methods for analyzing and designing systems. However, there are very few guidelines for selecting an appropriate method for a task. In this research, the author has found that the underlying knowledge representation scheme of a method is the determinant of a method's expressive power.

2. The user’s experience. User experience and the standard procedures used in an organization will affect users' attitudes toward the use of a certain method.

3. Types of applications. A method which is natural for applications to a domain should be used. For example, Structured Design works well for the design of COBOL programs, Data Flow Diagram works well for traditional data processing applications, and the object-oriented approach works well for distributed systems.

4. Automatic support. Applying a structured method without automated support is counterproductive. Some methods lend themselves to the building of automatic tools, some don’t. Selection of an appropriate method for representation in the MetaPlex system description language was also an important factor.

Rules can be formulated according to the above criteria so that a consultation expert system can be developed to help the description definer in selecting what description
language to use. Information System Expert [Nunamaker, et al., 1988] provides a framework for building such an expert system to select a language for the specification of an application. The managing of language definition requires tremendous effort. Appointing a designated system administrator who is responsible for coordinating the definition and use of system description languages is suggested.
Chapter 6

Integration of GDSS and CASE: A Metasystem Approach

In this chapter, the need to support collaboration in the system development process is first addressed. Methods and tools that have been used for coordinating the collaborative effort of a system development team are discussed. It is suggested that GDSS tools be used by system development teams in their face-to-face system development meetings, and the advantages of using GDSS tools in the context of information system development are presented. Because a customized GDSS is needed to support group deliberation in system development, the use of a metasystem such as MetaPlex is proposed to accomplish this customization. The design of a specific tool, called Structured Electronic Brainstorming System (SEBS), is discussed to demonstrate this concept. Since the GDSS and CASE can both be supported by the same metasystem, the integration of GDSS and CASE becomes possible. A framework for such integration is presented.

6.1 System Development As a Collaborative Effort

System development is a collaborative effort among managers, users, and system developers. It has been recognized as a process which acquires expertise from multiple experts [Henderson, 1987]. Domain specific knowledge about an application and its environment as well as technical knowledge about system development have to be incorporated into
a cohesive framework to make the target system work. The system development process therefore involves many group activities. The complexity of the dialog and poor communication among managers, users, and developers has always been identified as one of the major factors resulted in failure of information system projects [Chen, Nunamaker, and Konsynski, 1987].

A study has shown that face-to-face meetings are important in the software development process [Cook, et al., 1987]. Group meetings among users, managers, and information system personnel can be used to communicate and elicit information and knowledge, to create alternatives, to resolve disagreements, to reach consensus, and to make design decisions. Unfortunately, system development meetings are usually out of focus and often fail to reach conclusions because of the mis-communication caused by the diverse backgrounds and viewpoints of the participants. For a complex and large software project, the participation of users and managers in a project is critical to its success. Because having project members interact with each other one-on-one is very time-consuming, if not infeasible, support for software project meetings are essential for full participation by project members. As shown in Figure 6.1, most CASE tools are designed for individuals rather than for groups; therefore information systems personnel have to interview users and managers one-on-one and enter partial views of a system from each individual into CASE tools. The integration of multiple viewpoints must be done by the information systems personnel. Since users and managers communicate only indirectly with each other and information systems personnel rarely coordinate their efforts, the resulting system description usually reflects a broken picture of a target system.

"Effectiveness is a social concept: it applies to groups. Efficiency is an individual concept: it applies to isolated acts" [Strassmann, 1985; p. 117]. Using this statement as a measure to evaluate efforts to develop tools for increasing software productivity, the author found that most such tools are designed for individual work and emphasize efficiency of software development. Since there are almost no tools or methods that
Figure 6.1: The Use of Individual-Based CASE Tools
support group activities in a software project, effectiveness has rarely been addressed in the current system development environment.

While a need exists, most system development methods do not directly support group interaction in a face-to-face setting. Joint Application Design (JAD) developed by IBM is an exception [IBM, 1987]. CASE tools, which are usually based on structured methods, do not explicitly provide mechanisms to facilitate meetings about a software project. A software project meeting support system which can be incorporated into a software development environment is a critical component that is missing from CASE products.

6.2 A Method for Collaborative System Development: JAD

Collaboration for system development has only recently received much attention [Walz, Elam, Krasner, and Curtis, 1987]. Among the very few methods that do address collaboration in a system development project, Joint Application Design (JAD) "is a technique developed by IBM to accelerate the system design process. JAD provides a technique for documenting business requirements from the user's viewpoint, rapidly and accurately, in a language the user understands. JAD accomplished this through a series of design sessions where users, management and data processing personnel work together – guided by an experienced JAD session leader – to produce a set of user-approved design specifications" [IBM, 1986]. Several features make the JAD method unique [IBM, 1988]:

1. Emphasis on the involvement of users, managers, and system personnel. All of these participate in a design workshop to which each group contributes its knowledge to the project. This is critical to the success of the project.

2. Use of a multimedia room. Visual aids, such as graphics, help people to understand a complex problem. A multimedia room setting is used in a JAD session to facilitate
the discussion and presentation.

3. Having planning and design meetings conducted by a session leader. A JAD session leader is responsible for guiding each JAD meeting toward predefined objectives and makes sure that the required documents are generated. The facilitation of a JAD session requires very high human and communication skills since participants are from different backgrounds and hold different positions.

Currently some CASE tools, such as Excelerator, have been used in JAD sessions [Sobnovski, 1988], but only in a stand-alone mode to record design information generated from a JAD session. No automatic tool has been used to support the interactions among JAD session participants. However, ideas and concepts in JAD are helpful in developing a face-to-face GDSS for use in facilitating group interactions and capturing the group dynamics in the system development process, leading to the design of a customized system for collaborative system development.

6.3 Collaborative System Development Tools

There are very few tools that support an information system project at the group level. Some of the tools that are available, such as version and configuration control systems, are used primarily to coordinate the software components generated by a programming team at the implementation phase. In this research, the focus was on the support of communication among project members in the planning, analysis, and design phases. Currently only some generic tools, such as electronic mail and group decision support systems, are available to support the group activities of an information system project team. Studies have shown that electronic mail is an effective way to coordinate project members’ activities by announcing meetings and checking and reporting project status, but there is evidence that major design decisions are still made in face-to-face meetings
A GDSS environment, such as Plexsys Planning Tools, is designed to support face-to-face meetings for planning and designing.

Group Decision Support Systems (GDSS) research has provided researchers with insights into techniques for facilitating software project meetings. A GDSS consists of a set of software, hardware, and language components, as well as procedures that support a group of people engaged in a decision-related meeting [Huber, 1984]. Group support systems, especially when used by large groups for complicated tasks, often increase group member participation, reduce meeting time, and improve the quality of meeting results [Dennis, George, and Nunamaker, 1987]. The advantages of using a GDSS in the context of system development include:

1. Avoidance of direct confrontation of participants.
   A new information system usually changes the power structure of the organization for which it is developed. Politics is involved and confrontation occurs in the development process of information systems. Anonymity is guaranteed in GDSS-supported meetings so that direct confrontation among participants can be prevented.

2. Increased creativity in the development process.
   Creativity is one of the critical components in the system development process [Weber, Weinstein, and Chen, 1987]. This is particularly true in applying information systems strategically and in defining the requirements of a system. Experiments have shown that a GDSS is effective in facilitating creativity in planning and design types of meetings [Applegate, Kosynski, and Nunamaker, 1986]. The electronic brainstorming system in Plexsys Planning Tools was designed to facilitate creativity in group meetings.

3. Increased productivity of project meetings.
   Several mechanisms in GDSS are useful for increasing meeting productivity. First,
by using computer terminals, participants can enter their knowledge and opinions of the target systems in parallel. Second, the information entered will be automatically captured by the computer system and stored in a database so that no information will be lost. Third, the session management system and the structuring techniques embedded in the GDSS force participants to focus on the most important issues without getting off the track.

6.4 A Comparison Between GDSS and CASE

After examining the concepts and components in GDSS and CASE, the author has found some similarities and differences between them. A comparison of the two suggests the possibility of integrating GDSS and CASE because they have in common.

1. Automated support.
   Structured group management techniques (e.g., nominal group technique and Delphi technique) have been shown to be superior to less structured processes for group meetings [Huber, 1982]. A GDSS provides automated tools to support these structured techniques for conducting group decision meetings. Information system development research has shown that structured techniques (e.g., Structured Analysis and Structured Design) are effective for communication among team members, for recording information gathered, and for analyzing system documents. CASE provides automated tools to support the use of structured methods in the development of information systems [Chikofsky and Rubenstein, 1988].

2. An information repository.
   A GDSS uses database and knowledge-base systems to store information from meeting sessions in the form of organizational memory. Organizational memory can be used to integrate various tools used in meetings and to facilitate information sharing across meetings of the same group or different groups. Likewise, CASE
tools use data dictionary systems to store information about projects in order to integrate tools used in various phases in the system life cycle.

3. Use of graphics.

Graphics tools have been used extensively in CASE for capturing and representing system development information. Researchers in hypertext systems for group deliberation have just started to explore the use of graphics tools in a GDSS [Conklin, 1987], but GDSS tools haven't used graphics to their full potential.

4. Language components.

CASE tools such as PSL/PSA and Excelerator provide a language component that the designer can use to define a target system so that structured information can be captured in a data dictionary or a knowledge base. Existing GDSS tools do not offer a language component for group argumentation and deliberation so that a GDSS tool (e.g., an electronic brainstorming system [Applegate, Konsynski, and Nunamaker, 1986]) often generates "messy data." Messy data need to be identified and consolidated by meeting participants and the facilitator to make the underlying structures and semantics more apparent.

A GDSS can be used as a front end tool to elicit information and knowledge from project members in face-to-face design meetings. Information captured by a GDSS tool can be used to populate a data dictionary of CASE tools. At the same time, the structured methods and graphic tools used in CASE tools can be adapted and employed in a GDSS. This provides meeting participants a structured representation through which to consider issues as well as to gather and present information in graphic form. Just as Problem Statement Analyzer (PSA) can be used to analyze information in a PSA data base [Teichroew and Hershey, 1977], similar analysis functions can be applied to meeting information once it has been captured in a structured format. In the following sections the author presents a metasystem approach which makes the integration of GDSS and CASE possible.
6.5 Using Plexsys Planning Tools for System Development Meetings

Plexsys Planning Tools constitute a GDSS developed at the University of Arizona Management Information Systems Department. Functionalities of some of the frequently used processes and tools in the context of supporting software development meetings include:

1. Session Preparation.
   In the session preparation stage the session leader meets with key participants to decide the length of the session, the participants, the topics to be discussed, the EBS question, and the objectives to be achieved. It is the session leader who will create the stimulus for the electronic brainstorming session. The standard approach is to post a question on the participants' screens at the beginning of the session. Alternative approaches that can be used for stimulating a brainstorming session are presentation of a sample case or demonstration of a similar software system. Participants will be asked to react to the stimulus provided by expressing their opinions and reactions to the situation presented.

2. Electronic Brainstorming (EBS).
   The EBS session is initiated by giving the participants a start-off point, usually introduced in the form of a question. Participants are encouraged to use free association to generate as many ideas as possible. Experience with EBS has demonstrated that each participant becomes the initiator of a thought trail. Subsequent iterations allow participants to comment on ideas generated by other participants. This helps clarify trails of thought by introducing different points of view for each of the thought trails. EBS generates a session file that contains all the comments generated during the session. Questions such as, "What are the objectives of the system ABC?" (used in the system planning phase), "What are the input and output of the system ABC?" (used in the requirement definition phase), "What are the data elements in the customer database?" (used in the logical design phase), can
be used at an EBS session. EBS questions should be carefully phrased so that the required information can be elicited from participants easily. After a EBS session, the EBS comments will be analyzed in the Issue Analysis process. Issue Analysis consists of Issue Identification and Issue Consolidation.

3. Issue Identification.

The issue identification tool allows meeting participants to classify comments generated in the EBS session into issues. Each issue is supported by EBS comments. Depending on the EBS question posted, issues may mean input, output, system functions, data items, etc. in the context of information system development.

4. Issue Consolidation.

The issue consolidation process is usually directed by the facilitator with the help of participants. The process of issue consolidation entails creating structural information for the target system. For example, in the EBS and issue identification phases, a set of data structure components, such as data item, group items, as well input and output records has been generated. The issue consolidation tool can be used to group data items into group items, and assemble group items into input or output records. However, relations created are represented only in textual format.

5. Voting Mechanism.

The voting mechanism allows participants to prioritize the issues identified. The tool supports voting methods such as yes/no, agree/disagree, 1 to 10 scaling, multiple choice, and preference ordering. Statistics are gathered for a follow-up discussion. The voting mechanism is particularly useful in resolving opinion conflicts among participants, determining major functions of a system, and choosing solutions among design alternatives.

Collaboration among project team members is a continuous process. A consolidated issue may call for another design meeting and will be used to formulate an EBS question
for further deliberation at that GDSS session. Figure 6.2 shows a model of Plexsys Planning Tools. Each tool is represented as an instance of a TOOL object type. The sequence in using these tools is specified by the come-before relation type. Data generated by tools are identified by the generate relation type. Relations among tools and data generated by tools are also defined. A MetaPlex system description language can be defined based on this model. This language can be used to capture data generated from a series of GDSS sessions in a system description. Browsing tools such as Target System Editor and Structure Browser can be used to allow the users to browse the system description to review all the information collected in project meetings.

6.6 An Example of Generating GDSS Tools from MetaPlex: Structured Electronic Brainstorming System

Christakis [1987] noted that "inter-disciplinary teams cannot work productively and efficiently in designing complex systems unless their work is supported and argued by methodologies that have been invented specifically for this task." This concern is especially true for large software projects in which members with diverse backgrounds need to work together. A metasystem approach has been taken to provide the flexibility to customize methods for both group deliberation in GDSS and system definition in CASE.

Groups who use Plexsys in meetings generally use the following process: A meeting agenda is set before the meeting. Meeting participants first use the Electronic Brainstorming (EBS) to generate ideas on the question posted on the screen from their PC workstations. Ideas created (also called EBS comments) are randomly sent out through a local area network to other participants to stimulate them to generate more ideas. An EBS session usually takes 45 minutes. Following the EBS session, an Issue Analyzer (IA) tool is used to identify and consolidate the EBS comments into major issues. A callable voting program can be used to prioritize the consolidated issues. The use of
Figure 6.2: Representation Model of Plexsys Planning Tools: Process and Data
Issue Analyzer usually takes about 60 minutes. Since ideas are entered as free format text, an idea generated by one participant can be interpreted by another out of its original context. Because the structures and relations embedded in the original ideas might be lost in the process, Structured Electronic Brainstorming System (SEBS) has been designed to eliminate this drawback.

Software development meetings usually are held for specific purposes, such as to define the system objectives, to identify all the required reports and queries, or to determine system functions. These meetings often employ some structured methods. Before running an SEBS session, the session leader will choose or define a MetaPlex system description language for the meeting discussion. A language for SEBS is defined based on the nature of tasks, the common terminology used in an organization, theories and methods in a specific domain.

The flexibility of the metasystem has expanded Plexsys Planning Tools into a GDSS generator which can generate customized GDSS environments to support a wide-range of tasks. This extension to GDSS capability, as Huber suggested, will lead to the success of adoption of a GDSS in an organization [Huber, 1984]. The session leader should define "the language of business" to reduce the communication barrier among meeting participants [Martin, 1987]. Since users and managers are allowed to "speak their own language," they will be able to participate fully in the discussion.

The language specified by the session leader will be loaded into the SEBS tool to systematically direct the contents and structures of group discussion. Whenever users want to generate a new idea, the SEBS tool asks them to categorize it according to the object types defined in the language. Users can also create relations among ideas, such as specifying "Group Item A can be decomposed into Data Item B, C, and D," "Report E should include Group Item A and F," etc. Detailed attributes about objects and relations, such as "the data format of Data Item is 9(X)," can be collected through the Attribute Browser. Additional advantages of using the SEBS tool are its capability to:
1. Merge the brainstorming and issue identification processes. By merging the original brainstorming and issues analyzing processes SEBS helps to reduce meeting time.

2. Allow users to generate ideas on several related issues at one brainstorming session. Instead of brainstorming on a single question, such as “What are the critical success factors of the application ABC?”, the system allows users to generate ideas on several categories. An EBS question can be as complex as “What are the Input, Report, and Process in the application XYZ?” and “What are the relationships among Input, Process, and Report in the application XYZ?”

3. Help users focus on the issues under discussion. Current Plexsys tools are adequate for exploration type of meetings. However, to collect specific information for system development in a meeting, the SEBS will be an ideal tool.

4. Facilitate the acquisition knowledge concerning complex relations. In the current Plexsys Planning Tools, Issue Analysis supports the identification and consolidation of issues from an EBS file. Relations among issues usually get lost in the consolidation process. SEBS allows participants to enter relations among objects in a structured format to prevent this process loss.

5. Integrate SEBS results into CASE tools. Information captured has been categorized in a structured format so that SEBS meeting results can be easily transformed and exported to CASE tools.

6. Apply completeness and consistency checking on meeting information. Just as Problem Statement Analyzer can be used to analyze information in a PSA data base captured through Problem Statement Language [Teichroew and Hershey, 1977], similar analysis functions can be applied to meeting information, since it is captured in structured form. The meeting participants will be informed of missing information and of inconsistencies in their argumentation.

Other tools can also be designed by using MetaPlex to define languages to drive
the interface of the tools. The SEBS tool is one example of this potential. Currently Topic Commenter in Plexsys [1988] has very limited metasystem functions. A facilitator can create a list of topics so that participants can make comments on each topic. Each topic considered in a session is like an object type. The current Topic Commenter does not have the capability to elicit attributes for instances in each topic and does not support the description of relationships.

6.7 A Metasystem Approach to the Generation of Computer-Supported Collaborative Work Tools

Webster [1987] pointed out the importance of processability and expressiveness in design languages. The higher the processability of a system description the easier it is for computers to process it. The higher the expressiveness of the representation in a software system the easier it is for users to describe a target system. The author has applied some of his ideas about both to computer-supported collaborative work. Most computer-supported collaborative work tools such as The Coordinator and Information Lens, use type attachment to categorize and to clarify a message which is in natural language. Imposing some structures reduces the ambiguity of the messages interchanged so that these messages can be easily processed by computers. However, a system with a rigid structure will inhibit users from expressing their ideas flexibly. The tradeoff between structuralization (processability) and flexibility (expressiveness) of computer-supported collaborative work (CSCW) tools is obvious (see Figure 6.3).

CSCW tools discussed in Section 2.3, Plexsys Planning Tools, and MetaPlex tools (e.g., SEBS) are positioned in the figure to show their relative structuralization and flexibility. Looking back to the development of Plexsys Planning Tools, EBS was developed first, then Issue Analysis, and most recently Topic Commenter. The trend is moving towards capturing information in a more structured format. In order to keep the flexi-
Figure 6.3: The Effect of a Metasystem Approach to the Computer-Supported Collaborative Work Tools
bility and generality of Plexsys Planning Tools and to enhance the structuralization of these tools, a metasystem approach has been proposed in this research so that both the structuralization and flexibility of CSCW tools can be increased. At the meta level of the metasystem, language definers have the flexibility to define a description language according to a specific domain, while at the instance level, description definers will conform to the structure defined by the language. MetaPlex is designed to provide extra flexibility for the description definer at the instance level: 1) some meta level functions are available on-line to the description definer and 2) an undefined object type and an undefined relation type can be used to represent an unexpected knowledge structure in a target system.

6.8 Defining a System Description Language for Information Systems Planning: Critical Success Factors

Critical Success Factors is an information systems planning method used to identify the key factors for a manager to be successful in the business. The original CSF method provides only procedures for conducting the CSF study and focuses only on information requirements for an individual manager's decision [Henderson et al., 1984]. The binding of CSF to an individual manager's personal needs causes frequent changes in the resulting system. The author has designed a collaborative CSF tool, generated by MetaPlex, which allows managers and IS personnel to develop an information system from the organization's and the manager's perspectives. Object types in CSF method are: CRITICAL SUCCESS FACTOR, MEASURE (to the CSF), REPORT or QUERY (to reflect these measurements), INFORMATION SYSTEM (which generates the reports and queries), BUSINESS GOAL (behind these critical factors), and ORGANIZATION ENTITY (involved). All these object types and relation types among them have been defined in MetaPlex as a language which can be used to guide users in using the CSF method.
The CSF method for information systems planning follows a four-step procedure [Martin, 1983]: 1) Identify the critical success factors. Managers as a group are asked to identify the critical factors which will affect the success of their business. 2) Determine the measurements for identified critical success factors. Critical success factors are sometimes difficult to quantify so that soft measures have to be used. 3) Design reports or on-line queries to inform managers of the status of the CSF measured or changes that have been made. Reports or queries have to be generated from an existing or a new information system. Managers who will use these reports or queries are not necessarily the same managers who identify CSF or define the measures for CSF. The original CSF method is designed to be used by individual executive officers to define their own information needs. However, this method can be extended for use by a group of managers to determine the information system requirements of an organization as a whole. A GDSS can be used in supporting groups who use CSF to define their information needs.

Figure 6.4 is a representation model for the Critical Success Factors method. A representation model is a graphical representation of the object types and relation types defined by a method. In such a representation model, object types are enclosed in rectangles, relation types among object types are represented as labeled arrow lines (only binary relationships can be represented in a representation model and only one direction of a relation is shown in Figure 6.4). Attribute types that describe object types and relation types are not shown to simplify the presentation of the model. As shown in this figure, the CSF method contains object types in both information systems and organization systems domains. It has been used to bridge the gap between information systems planning and business planning. Language definers can extend the language to cover other related object types and relation types in both organization and information systems domains.
Figure 6.4: A Representation Model of Critical Success Factors Method
6.9 Using MetaPlex to Generate Integrated GDSS and CASE Environments

As was discussed in Chapter 5, MetaPlex can be used to generate CASE tools, such as tools based on the data flow diagram technique or Problem Statement Language. In this chapter the author has presented the use of MetaPlex to generate a customized structured electronic brainstorming system. The SEBS can be used by an IS project team as a front end tool to elicit information required by the project in a face-to-face group meeting. Since the information is captured in a structured format, it can be used directly by a CASE tool defined in the same language (or a super set of the SEBS language). Through the use of the transformation system in MetaPlex, the system description captured in a SEBS session can also be transformed into a system description of a CASE tool defined in a different language.

A framework for the integration of GDSS and CASE is depicted in Figure 6.5. At the top of the figure is the architecture of MetaPlex which has been discussed in Section 4.4. MetaPlex can be used to generate some CASE tools which can be used in information system planning, analysis, and design. Some other implementation and maintenance CASE tools, such as fourth generation languages and application generators, can work with CASE tools generated by MetaPlex. MetaPlex can also be used to generate GDSS tools such as SEBS which can be used in business and information systems planning, as well as system analysis and design. The key to the integration of GDSS and CASE is the sharing of knowledge bases in MetaPlex, i.e., the description languages definitions and system descriptions.

Figure 6.6 shows how a GDSS driven by a metasystem can be used as a front end to CASE tools. The use of GDSS tools allows us to capture strategic assumptions and business plans, as well as models of an organization, from managers through group meetings. The application of the metasystem concepts in GDSS tools, such as SEBS, makes it possible to customize a GDSS environment to acquire information from multiple
Figure 6.5: The Integration of GDSS and CASE
experts involved in a software project in a structured format to be integrated with CASE tools. With an integrated GDSS and CASE environment, an organization can manage knowledge about its business and information systems consistently. Therefore, the alignment of organization structures and business strategies with information systems can be supported by an integrated system development environment such as MetaPlex.

The points made in this chapter can be summarized as follows:

1. The degree of collaboration in a software development project determines the effectiveness and success of the project.

2. GDSS tools can be used as a front end to CASE tools to facilitate the collaboration between managers, users, and information systems personnel.

3. CASE technologies and the metasystem approach can enhance existing functions of current GDSS and provide new features in future GDSS.

4. A metasystem approach provides the basis for integration of GDSS and CASE which paves the way for the integration of organization and information systems modeling.
GDSS Driven by a Metasystem

CASE Tools Generated by a Metasystem

Structured Information

Organization and Information Systems Models

Integrative models of organization and information systems

Figure 6.6: The Use of a GDSS as a Front End to CASE Tools: A Metasystem Approach
Chapter 7

Conclusion

The need for the integration of the strategies and policies of an organization with its information systems and the alignment of organization structures with information system structures are critical to the survival of a business. In this research, the author studied how automated tools can be designed and used to support the integration of organization and information system modeling. This dissertation is concluded with discussions of the contributions and of the future directions for related research.

7.1 The Contributions of the Dissertation

This research made contributions in the following areas:

1. Identifying the needs and features of the metasystem approach to the development of CASE tools.

2. Building a conceptual framework for integrating GDSS and CASE for organization and information system modeling through a metasystem approach.

3. Defining a system development research methodology and elaborating on the significance of system development in management information systems.
Each of these contributions is summarized in turn in the following subsections.

7.1.1 Contributions to the Metasystem Approach

Contributions to the design, implementation, and use of metasystems through the development of MetaPlex include:

1. *Flexibility at instance level.*
   The availability of some metalevel functions to the end users of a metasystem makes the system much more flexible.

2. *Incremental strategies for implementation.*
   Layered architecture and object-oriented implementation provide a basis for the development of *incremental CASE* and *integrated CASE*.

3. *Uniform user interfaces for ease of use.*
   Uniform user interfaces from language definition to target system description are designed to make use of the system easier.

4. *Managing complex relations.*
   Relations are defined as first class objects so that complex relations can be represented and the navigation of relations is supported.

5. *Extendible description languages.*
   New object types and relation types can be added to an existing description language without affecting an existing system description defined in the description language.

   Sublanguages can be generated from a language and used as filtering mechanisms that permit description definers to interact with only a subset of a system description through the system description system.
7. *Transforming system descriptions.*

A rule-based transformation system allows a system description defined in one language to be converted to another.

7.1.2 Contributions to CASE and GDSS Areas

Contributions to CASE and GDSS areas include:

1. The importance of integrating organization and information systems modeling is addressed.

2. A metasystem to generate CASE tools for both organization and information system modeling is employed.

3. The ability of GDSS and CASE to benefit each other is suggested by comparing the similarities and differences between CASE and GDSS.

4. Use of a metasystem approach to the generation of GDSS tools is shown to increase both the structure and flexibility of such tools.

5. The use of GDSS tools driven by a metasystem to support collaborative efforts in systems development so that information captured can be seamlessly integrated with existing CASE tools is demonstrated.

7.1.3 Contribution to the System Development Research Methodology

System development is one of the major thrusts in MIS research. In Chapter 3 the author used a simple model to describe relationships among research process, research methodology, research domain, and the body of knowledge. A review of some of the evolution that has taken place in certain areas makes the importance of system development efforts obvious. The author also has described the system development research process
from a methodology perspective. The discussion presented in Chapter 3, even though very preliminary, constitutes one of the first long overdue efforts to formalize system development research methodology in management information systems research.

7.2 Future Research

Two major areas for extending the research for this dissertation are extending the MetaPlex system and conducting empirical studies to validate the MetaPlex system.

7.2.1 Extensions to MetaPlex

Several potential extensions to MetaPlex already have been identified:

1. To extend the knowledge base of MetaPlex as a server for concurrent accessing and to apply version control on it.

2. To build an inference mechanism that will allow inheritance in both object types and relation types.

3. To extend the data types of an attribute in MetaPlex to any Smalltalk class.

4. To incorporate procedures attachment to both object types and relation types.

5. To implement the sublanguage creation function.

7.2.2 Validation of MetaPlex

Several empirical studies could be conducted using MetaPlex or comparing it with other CASE tools:
1. Comparison between generic and domain-specific requirements definition languages. Peter Freeman [1987] has stated that a domain-specific language should work more effectively than a generic requirement definition language. This statement can be tested by using MetaPlex to generate both a domain specific language and a generic requirement language for a task. A laboratory experiment can be conducted to test how well subjects perform in analyzing a given case. User satisfaction in using two different requirement languages can also be measured and compared. The advantages of using a metasystem like MetaPlex to conduct experiments are that: 1) MetaPlex allows the generation of both a domain-specific and a generic requirement definition language, and that 2) the ability of MetaPlex to use the same user interface tools and functions makes the user interface a control variable instead of an independent variable.

2. Comparison between a language-driven and a graphic-driven interface.
MetaPlex is still a language-based system, while other CASE tools such as Excelerator are using graphics intensively. It would be interesting to study users' performance and preferences in using these two different type of tools. Although using graphics in representing a system specification certainly has the advantage of better visual effect, users probably spend more time in laying out the graphics icons on the screen than in specifying a system they know. Comparing the use of text, graphics, and matrix to gather input and generate output of a system description will be an interesting exercise and the results of such comparisons may contribute to the design and implementation of future CASE and GDSS tools.

3. Comparison between menu-driven and command driven modes.
There are two methods to enter a relation into MetaPlex: using menu-driven dialog to select objects from a menu and using a template to enter a relation statement by typing in objects' names. The former method should reduce typing errors and increase the system developer's productivity. However, this claim needs to be tested through laboratory experiments.
4. Comparison among various analysis and design methods.

Recently, tremendous interest has been shown in conducting empirical studies of competing system analysis and design methods [Hoffer, 1982; Mantha, 1987; Shoval and Even-Chaime, 1987]. MetaPlex can be used to generate tools based on various analysis and design methodologies so that empirical studies of automatic tools based on various methods can be conducted.

7.3 Final Remarks

A GDSS provides a mechanism to capture the dynamic of organization decision making. The rationale of an organization’s strategies and policies can thus be obtained through the use of the GDSS and stored in the GDSS knowledge base. Information systems have been used to support an organization’s strategies and policies more and more frequently. The foundation of integrating strategic planning and information system technologies must be the integration of organization and information system modeling. GDSS and CASE tools together provide a computer-supported environment for the modeling of organization and information systems. The metasystem, MetaPlex, presented in this dissertation has demonstrated the possibility of integrating organization and information system modeling through the generation of group decision support systems and computer-aided software engineering tools.
Appendix A

A Sample Language Syntax Report

*********************************************************************************************************************
* The Language Syntax Report                                      *
* Date: 27 May 1988 Time: 9:01:19 pm                                *
*********************************************************************************************************************

Language Name: BSP
Comment: Business Systems Planning

Object Types are as follows:

Name: BUSINESS PROCESS
Comment: None
No attribute type defined.
Involved in the following Relation Types: None

Name: DATA SET
Comment: None
No attribute type defined.
Involved in the following Relation Types:
Relation Type Name: IS-USE-DS
Statement:
DATA SET used-by: INFORMATION SYSTEM

Relation Type Name: OWNS
Statement:
DATA SET, INFORMATION SYSTEM owned-by: ORGANIZATION ENTITY

Name: INFORMATION SYSTEM
Comment: None
INFORMATION SYSTEM has the following Attribute Types:
  Attribute Type Name: importance
  Comment:
  Propmt: What is the value of the importance?
  Data type: String
  Occurrence: onlyOne
  Legal values: OrderedCollection ('very important' 'important' 'trivial')
  Default values: No default value

  Attribute Type Name: cost
  Comment:
  Propmt: What is the value of the cost?
  Data type: Integer
  Occurrence: onlyOne
  Legal values: No legal value
  Default values: No default value

  Attribute Type Name: mode
  Comment:
  Propmt: What is the value of the mode?
  Data type: String
  Occurrence: onlyOne
  Legal values: OrderedCollection ('online' 'batch' 'manual')
  Default values: 'online'
Involved in the following Relation Types:
Relation Type Name: INFORMATION SYSTEM DECOMPOSITION
Statement:
INFORMATION SYSTEM consists-of: INFORMATION SYSTEM
INFORMATION SYSTEM is-part-of: INFORMATION SYSTEM
Relation Type Name: IS-USE-DS
Statement:
INFORMATION SYSTEM uses: DATA SET
Relation Type Name: OWNS
Statement:
DATA SET, INFORMATION SYSTEM owned-by: ORGANIZATION ENTITY

Name: ORGANIZATION ENTITY
Comment: None
Involved in the following Relation Types:
Relation Type: OWNS
Statement:
ORGANIZATION ENTITY owns: DATA SET INFORMATION SYSTEM

There are total 4 Object Types defined.

Relation Types are as follows:

Name: INFORMATION SYSTEM DECOMPOSITION
Comment: None
Number of Object Type Groups: 2
Occurrence of Object Type Groups: one, many
Equivalent statements for the relation:
INFORMATION SYSTEM consists-of: INFORMATION SYSTEM
INFORMATION SYSTEM is-part-of: INFORMATION SYSTEM
No attribute type defined.

Name: IS-USE-DS
Comment: None
Number of Object Type Groups: 2
Occurrence of Object Type Groups: many, many
Equivalent statements for the relation:
INFORMATION SYSTEM uses: DATA SET
DATA SET used-by: INFORMATION SYSTEM
IS-USE-DS has the following Attribute Types:
Attribute Type Name: type
Comment:
Prompt: What is the value of the type?
Data type: String
Occurrence: onlyOne
Legal values: OrderedCollection ('update' 'access' 'delete'
Default values: 'access'

Name: OWNS
Comment: None
Number of Object Type Groups: 2
Occurrence of Object Type Groups: many, many
Equivalent statements for the relation:
ORGANIZATION ENTITY owns: DATA SET, INFORMATION SYSTEM
DATA SET, INFORMATION SYSTEM owned-by: ORGANIZATION ENTITY
No attribute type defined.

There are total 3 Relation Types defined.
Appendix B

A Sample of System Description Report

***************************************************************************
* The System Description Report                                      *
* Date: 5 July 1988     Time: 11:40:54 pm                               *
***************************************************************************

Target System Name: ORDER PROCESSING SYSTEM
Comment: An order processing system

Objects defined are as follows:

Name: customer-order
Object Type: INPUT
Comment: None
No attribute defined.
Involved in the following relations:
Relation Name: receive-customer-order
Statement:
  customer-order received-by: order-processing-system

Name: customer-report
Object Type: OUTPUT
Comment: None
customer-report has the following attributes:
  Attribute Name: purpose
    Value: 'The purpose of the customer-report is ....'
Involved in the following relations: None

Name: monthly-report
Object Type: OUTPUT
Comment: None
No attribute defined.
Involved in the following relations: None

Name: order-entry-program
Object Type: PROCESS
Comment: None
order-entry-program has the following attributes:
  Attribute Name: online
    Value: true
  Attribute Name: type
    Value: 'program'
Involved in the following relations:
Relation Name: order-processing-system-decomposition
Statement:
  order-entry-program order-file-update order-reporting-programs is-part-of: order-processing-system
Relation Name: process-decomposition
Statement:
  order-entry-program order-reporting-programs is-part-of: order-processing-system
Relation Name: receiving-customer-order
Statement:
  order-entry-program receives: customer-order
Name: order-file-update  
Object Type: PROCESS  
Comment: None  
order-file-update has the following attributes:  
  Attribute Name: online  
  Value: true  
  Attribute Name: type  
  Value: 'module'  
Involved in the following relations:  
Relation Name: order-processing-system-decomposition  
Statement:  
  order-entry-program order-file-update order-reporting-programs is-part-of:  
  order-processing-system  

Name: order-processing-system  
Object Type: PROCESS  
Comment: None  
order-processing-system has the following attributes:  
  Attribute Name: type  
  Value: 'system'  
  Attribute Name: online  
  Value: false  
Involved in the following relations:  
Relation Name: order-processing-system-decomposition  
Statement:  
  order-processing-system has: order-entry-program, order-file-update,  
  order-reporting-programs  
Relation Name: process-decomposition  
Statement:  
  order-processing-system has: order-entry-program, order-reporting-programs  
Relation Name: receive-customer-order  
Statement:  
  order-processing-system receives: customer-order  

Name: order-reporting-programs  
Object Type: PROCESS  
Comment: None  
No attribute defined.  
Involved in the following relations:  
Relation Name: order-processing-system-decomposition  
Statement:  
  order-entry-program, order-file-update, order-reporting-programs  
  is-part-of: order-processing-system  
Relation Name: process-decomposition  
Statement:  
  order-entry-program, order-reporting-programs is-part-of:  
  order-processing-system  
There are 7 objects defined.  
***************************************************************************
Relations defined are as follows:

Name: process-decomposition
Type of relation: process-decomposition
Comment: None
No attribute has been defined.
Equivalent statements for the relation:
order-processing-system has: order-entry-program, order-reporting-programs
order-entry-program, order-reporting-programs is-part-of: order-processing-system

Name: order-processing-system-decomposition
Type of relation: process-decomposition
Comment: None
No attribute has been defined.
Equivalent statements for the relation:
order-processing-system has: order-entry-program, order-file-update, order-reporting-programs
order-entry-program, order-file-update, order-reporting-programs is-part-of: order-processing-system

Name: receive-customer-order
Type of relation: input-received-by-process
Comment: None
No attribute has been defined.
Equivalent statements for the relation:
customer-order received-by: order-processing-system
order-processing-system receives: customer-order

Name: receiving-customer-order
Type of relation: input-received-by-process
Comment: None
No attribute has been defined.
Equivalent statements for the relation:
customer-order received-by: order-entry-program
order-entry-program receives: customer-order

There are 4 relations defined.
List of References


