SATOOL II: AN IDEF$_0$ CASE WORKBENCH USING ADA AND THE X WINDOW SYSTEM

THESIS

Betty Topp
Captain, USAF

AFIT/GCS/ENG/92M-04

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY
AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT/GCS/ENG/92M-04

SATOOL II: AN IDEF$_0$ CASE WORKBENCH USING ADA AND THE X WINDOW SYSTEM

THESIS

Betty Topp
Captain, USAF

AFIT/GCS/ENG/92M-04

Approved for public release; distribution unlimited

AFIT/GCS/ENG/92M-04

SATOOL II: AN IDEF$_0$ CASE WORKBENCH USING ADA AND THE X WINDOW SYSTEM

THESIS

Presented to the Faculty of the School of Engineering
of the Air Force Institute of Technology
Air University
In Partial Fulfillment of the
Requirements for the Degree of
Master of Science in Computer Systems

Betty Topp, Bachelor of Science in Computer Science
Captain, USAF

March, 1992

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	March 1992	Master's Thesis

4. TITLE AND SUBTITLE
SATOOL II: AN IDEF$_0$ CASE WORKBENCH USING ADA AND THE X WINDOW SYSTEM

5. FUNDING NUMBERS

6. AUTHOR(S)
Betty Topp, Capt, USAF

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Air Force Institute of Technology, WPAFB OH 45433-6583

8. PERFORMING ORGANIZATION REPORT NUMBER
AFIT/GCS/ENG/92M-04

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
LtC James Sweeder
SDIO/SDA, Room 1E149
The Pentagon, Washington D.C. 20301-7100
(202)-693-1826

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

The objective of this research effort is to perform an object oriented analysis, design and implementation of the graphical user interface (GUI) for the SATool II system. SATool II is a Computer Asisted Software Engineering (CASE) workbench developed using Ada and the X Window system. It is designed to serve as an IDEF$_0$ graphical project editor and data dictionary editor. IDEF$_0$ is the ICAM Definition Method Zero graphical notation language adopted by the Air Force to produce a function model of a manufacturing system or environment. The Air Force Institute of Technology is conducting on-going research in the use of IDEF$_0$ in the requirements analysis phase of the software lifecycle. This thesis describes the object oriented design and implementation of the GUI based on an entity-relationship model developed by earlier research efforts for the IDEF$_0$ language. It also describes the integration of the overall SATool II system composed of the essential model, drawing model, machine-independent Ada graphical support environment, and the graphical user interface.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Computer Aided Design, Software Engineering, Ada Programming Language, Object Oriented Design, X Window System, Computer Asisted Software Engineering	180
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

GENERAL INSTRUCTIONS FOR COMPLETING SF 298

The Report Documentation Page (RDP) is used in announcing and cataloging reports. It is important that this information be consistent with the rest of the report, particularly the cover and title page. Instructions for filling in each block of the form follow. It is important to ***stay within the lines*** to meet ***optical scanning requirements.***

Block 1. Agency Use Only *(Leave blank)*.

Block 2. Report Date. Full publication date including day, month, and year, if available (e.g. 1 Jan 88). Must cite at least the year.

Block 3. Type of Report and Dates Covered. State whether report is interim, final, etc. If applicable, enter inclusive report dates (e.g. 10 Jun 87 - 30 Jun 88).

Block 4. Title and Subtitle. A title is taken from the part of the report that provides the most meaningful and complete information. When a report is prepared in more than one volume, repeat the primary title, add volume number, and include subtitle for the specific volume. On classified documents enter the title classification in parentheses.

Block 5. Funding Numbers. To include contract and grant numbers; may include program element number(s), project number(s), task number(s), and work unit number(s). Use the following labels:

C	- Contract	PR	- Project
G	- Grant	TA	- Task
PE	- Program Element	WU	- Work Unit Accession No.

Block 6. Author(s). Name(s) of person(s) responsible for writing the report, performing the research, or credited with the content of the report. If editor or compiler, this should follow the name(s).

Block 7. Performing Organization Name(s) and Address(es). Self-explanatory.

Block 8. Performing Organization Report Number. Enter the unique alphanumeric report number(s) assigned by the organization performing the report.

Block 9. Sponsoring/Monitoring Agency Name(s) and Address(es). Self-explanatory.

Block 10. Sponsoring/Monitoring Agency Report Number. *(If known)*

Block 11. Supplementary Notes. Enter information not included elsewhere such as: Prepared in cooperation with... , Trans. of. , To be published in... . When a report is revised, include a statement whether the new report supersedes or supplements the older report.

Block 12a. Distribution/Availability Statement. Denotes public availability or limitations. Cite any availability to the public. Enter additional limitations or special markings in all capitals (e.g. NOFORN, REL, ITAR).

- **DOD** - See DoDD 5230.24, "Distribution Statements on Technical Documents."
- **DOE** - See authorities.
- **NASA** - See Handbook NHB 2200.2.
- **NTIS** - Leave blank.

Block 12b. Distribution Code.

- **DOD** - Leave blank.
- **DOE** - Enter DOE distribution categories from the Standard Distribution for Unclassified Scientific and Technical Reports.
- **NASA** - Leave blank.
- **NTIS** - Leave blank.

Block 13. Abstract. Include a brief *(Maximum 200 words)* factual summary of the most significant information contained in the report.

Block 14. Subject Terms. Keywords or phrases identifying major subjects in the report.

Block 15. Number of Pages. Enter the total number of pages.

Block 16. Price Code. Enter appropriate price code *(NTIS only)*.

Blocks 17. - 19. Security Classifications. Self-explanatory. Enter U.S. Security Classification in accordance with U.S. Security Regulations (i.e., UNCLASSIFIED). If form contains classified information, stamp classification on the top and bottom of the page.

Block 20. Limitation of Abstract. This block must be completed to assign a limitation to the abstract. Enter either UL (unlimited) or SAR (same as report). An entry in this block is necessary if the abstract is to be limited. If blank, the abstract is assumed to be unlimited.

Preface

This thesis documents the analysis, design, and implementation of SATool II, an object-based $IDEF_0$ Computer Assisted Software Engineering (CASE) workbench. $IDEF_0$ is a methodology used by the Air Force Program for Integrated Computer Aided Manufacturing (ICAM) in an effort to increase the manufacturing productivity of the Aerospace Industry through the application of computer technology (18:1-1). SATool II is the culmination of ongoing research at the Air Force Institute of Technology (AFIT) Department of Electrical Engineering associated with the Strategic Defense Initiative Organization (SDIO) and its interest with the $IDEF_0$ language.

SATool II is a CASE workbench designed to create, modify and preserve $IDEF_0$ diagrams and associated data dictionary used to describe a software or manufacturing system. It is written in Ada and uses an Ada/X Window interface to create the graphical images for the user interface. The Ada/X Window interface allows SATool II to operate independently from any particular type of computer hardware. An object oriented approach was used to analyze, design and implement this workbench. SATool II was designed around an abstract entity relationship model of the $IDEF_0$ methodology consisting of an essential model and a drawing model.

I would like to thank my advisor, Dr Thomas C. Hartrum, whose patience, support and guidance made the successful completion of this research effort possible. I would also like to thank Bruce Clay for his advice on X Windows, Dave Doak for the many hours of help with the Olympus System, and Rick Norris for teaching me the value of the -i option. I especially want to thank my husband Danny and my son Bob for their loving support. And, above all, thank you Lord for guiding me through the longest days.

Betty Topp

Table of Contents

Page

Preface ii

Table of Contents iii

List of Figures ix

List of Tables xii

I. Introduction 1

1.1 Background 2

1.1.1 The $IDEF_0$ Methodology 2

1.1.2 The C Version of SATool 2

1.1.3 The Ada Version of SATool 3

1.2 Problem Statement 4

1.3 Assumptions 4

1.4 Scope and Limitations 5

1.5 Standards 5

1.6 Approach 5

1.7 Equipment and Software 6

1.8 Sequence of Presentation 6

II. Literature Review 7

2.1 Introduction 7

2.2 CASE Tool Enhancements 7

2.3 Highlighting With Color 9

2.4 Online Documentation 10

Page

2.5 Configuration Management 10

2.6 Automatic Diagram Layout 11

2.7 Summary 13

III. Requirements Analysis 14

3.1 Introduction 14

3.2 General Requirements 14

3.3 Requirements Analysis for the Graphical User Interface . . 16

3.4 Essential Model 16

3.4.1 AFIT Data Dictionary Format 21

3.5 Drawing Model 24

3.6 Summary 34

IV. SATool II System Design 35

4.1 Introduction 35

4.2 Generic Multiple Object Manager 35

4.3 Essential Model Design 37

4.4 Drawing Model Design 39

4.5 MAGSE Design 39

4.6 Graphical User Interface Design 41

4.7 Overall SATool II System Design 45

4.8 Summary 45

V. Implementation 47

5.1 Introduction 47

5.2 Overall SATool II System Environment 48

5.3 Window Types 48

5.4 Dialogue Window 52

5.5 Title Window 52

Page

5.6 Drawing Window 53
5.7 Main Menu Window 53
5.7.1 Project Button 53
5.7.2 Diagram Button 56
5.7.3 Dictionary Button 58
5.7.4 Output Button 58
5.7.5 Options Button 58
5.7.6 Utility Button 59
5.8 Tools Title Window 60
5.9 Tools and Objects Windows 60
5.9.1 Create Button 61
5.9.2 Update Button 62
5.9.3 Move Button 62
5.9.4 Delete Button 62
5.9.5 Clear Diagram Button 62
5.9.6 Undo Button 67
5.10 Data Dictionary Editor 67
5.11 Summary 69

VI. Testing and Evaluation 76
6.1 Introduction 76
6.2 Testing 76
6.2.1 Unit Tests 76
6.2.2 External Function Tests 77
6.2.3 Integration Tests 77
6.2.4 System Tests 77
6.2.5 Acceptance Tests 78
6.2.6 Installation Tests 78

Page

6.2.7 Regression Tests 78

6.3 Test Results and Evaluation 78

6.3.1 MAGSE Interface 78

6.3.2 Graphical User Interface 79

6.4 Summary 80

VII. Conclusions and Recommendations 81

7.1 Summary 81

7.2 Conclusions 81

7.2.1 Research Accomplishments 81

7.2.2 SATool II 82

7.3 Recommendations 82

7.3.1 SATool II Menu Selections 82

7.3.2 System Enhancements 83

7.3.3 The MAGSE Subsystem 83

7.3.4 The Essential and Drawing Models 84

Appendix A. Essential Subsystem Implementation Packages 85

Appendix B. Drawing Subsystem Implementation Packages 89

Appendix C. MAGSE Subsystem Implementation 91

Appendix D. SATool II User's Manual 93

D.1 Introduction 93

D.1.1 Background and Purpose 93

D.1.2 Features 93

D.1.3 System Requirements 93

D.1.4 Overview 93

D.2 Getting Started 94

Page

D.2.1 Quick Start 94

D.2.2 Operating Environment 94

D.3 A Guided Tour 96

D.3.1 Introduction 96

D.3.2 The Main Screen 96

D.3.3 The Keyboard and Mouse 97

D.3.4 Using Tools and Objects 97

D.3.5 Creating and Viewing a Project 97

D.3.6 Saving and Loading a Project 99

D.3.7 Error Handling 99

D.3.8 Exiting SATool II 99

D.3.9 Summary 99

D.4 Objects and Tools 100

D.4.1 Introduction 100

D.4.2 The Objects 100

D.4.3 The Tools 102

D.4.4 Summary 103

D.5 Main Screen Menus 103

D.5.1 Introduction 103

D.5.2 PROJECT Menu 103

D.5.3 DIAGRAM Menu 104

D.5.4 DICTIONARY Menu 105

D.5.5 OPTIONS Menu 105

D.5.6 UTILITY Menu 108

D.6 Printing A Window 108

D.6.1 Introduction 108

D.6.2 X Clients for Window Capturing and Printing . . 109

D.6.3 Printing an $IDEF_0$ Diagram and Project 110

Page

Appendix E. SATool II Configuration Guide 112

E.1 Introduction 112

E.2 SATool II Configuration File 112

Appendix F. SATool II Test Cases 117

F.1 Unit Test Cases 117

F.1.1 General Main Screen Windows 117

F.1.2 Main Menu Buttons 117

F.1.3 Tools Window Buttons 120

F.2 Integration Test Case 123

Appendix G. SATool II User Evaluation Form 149

Appendix H. $IDEF_0$ Diagram Syntax Review 157

Bibliography 165

Vita 167

List of Figures

Figure		Page
1.	$IDEF_0$ Activity Essential Data Model	18
2.	$IDEF_0$ Data Element Essential Data Model	19
3.	Original $IDEF_0$ Drawing Model Relationships	25
4.	Original $IDEF_0$ Drawing Model Classes	26
5.	Original $IDEF_0$ Drawing Model Entities and Attributes	27
6.	Revised $IDEF_0$ Drawing Model Entity Relationship Diagram	28
7.	Revised $IDEF_0$ Drawing Model Entities and Attributes	29
8.	Essential Model Design	38
9.	Drawing Model Design	40
10.	MAGSE Design	42
11.	Graphical User Interface Design	44
12.	Overall SATool II System Design	46
13.	Booch Module Symbols	47
14.	Overall SATool II System View	49
15.	SATool II Subprogram Package Dependencies	50
16.	Dialogue Window Package Dependencies	52
17.	Title Window Package Dependencies	53
18.	Drawing Window Package Dependencies	54
19.	Main Menu Window Package Dependencies	55
20.	Project Button Package Dependencies	57
21.	Diagram Button Package Dependencies	58
22.	Output Button Package Dependencies	59
23.	Options Button Package Dependencies	59
24.	Utility Button Package Dependencies	60

Figure Page

25. Tools Title Window Package Dependencies 60
26. Tools Window Package Dependencies 61
27. Objects Window Package Dependencies 62
28. Create Button Package Dependencies 63
29. Update Button Package Dependencies 64
30. Move Button Package Dependencies 65
31. Delete Button Package Dependencies 66
32. Clear Diagram Package Dependencies 67
33. Undo Button Package Dependencies 68
34. Activity Data Dictionary Editor Screen 70
35. Data Element Data Dictionary Editor Screen 71
36. Data Dictionary Button Package Dependencies 72
37. Data Dictionary Main Menu Title Window Package Dependencies . . 73
38. Data Dictionary Main Menu Window Package Dependencies 73
39. Data Dictionary Activity Edit Menu Window Package Dependencies 74
40. Data Dictionary Data Element Edit Menu Window Package Dependencies 74
41. Data Dictionary Text Window Package Dependencies 75
42. Essential Subsystem Package Dependencies 86
43. Essential Fact Utilities Package Dependencies 87
44. CLIPS System Package Dependencies 88
45. Drawing Subsystem Package Dependencies 90
46. MAGSE Subsystem Package Dependencies 92
47. SATool II Main Screen Face 98
48. Activity Data Dictionary Editor Screen 106
49. Data Element Data Dictionary Editor Screen 107
50. $IDEF_0$ Activity and Data Elements 158

Figure Page

51. Sample $IDEF_0$ Hierarchical Decomposition 159

52. A-0 Diagram for 'Control Elevator' 160

53. A0 Diagram for 'Control Elevator' 161

54. A1 Diagram for 'Control Elevator' 162

55. $IDEF_0$ Arrow Types 164

List of Tables

Table		Page
1.	Data Dictionary Entry Format for Activity	22
2.	Data Dictionary Entry Format for Data Element	23

SATool II: AN IDEF$_0$ CASE WORKBENCH USING ADA AND THE X WINDOW SYSTEM

I. Introduction

The IDEF$_0$ methodology was developed by the U. S. Air Force Program for Integrated Computer Aided Manufacturing (ICAM) in an effort to increase the manufacturing productivity of the Aerospace Industry through the application of computer technology (18:1-1). The IDEF$_0$ methodology is part of the IDEF (ICAM Definition) method and is used to produce graphical representations of functions and their interrelationships for a manufacturing system or environment (18:1-1). IDEF$_0$ stands for "ICAM Definition Language Zero" (10:3).

To help promote the IDEF$_0$ methodology, the Air Force Institute of Technology (AFIT) Department of Electrical and Computer Engineering developed the predecessor to SATool II, a Computer-Aided Software Engineering (CASE) tool known as SATool (15:1-1). That tool is a graphical editor written in the C programming language and uses the window and graphics features provided by the Sun Microsystems workstations. Several enhancements were suggested for this tool which included the suggestion to make it a portable tool. These suggestions led to the creation of SATool II, an improved CASE tool for the IDEF$_0$ methodology (8:5-12) (27:1-2). SATool II falls into the category of CASE workbench since it not only provides a diagram editor but also other functions within its environment like a data dictionary editor and a syntax checking capability.

The following section provides a description of the background information associated with the development and implementation of SATool II. A brief description

of the $IDEF_0$ methodology is presented. Then, a discussion of the rationale that led to the creation of SATool II.

1.1 Background

1.1.1 The $IDEF_0$ Methodology $IDEF_0$ is based on SofTech's Structured Analysis and Design Technique (SADT) (18:iii). It is used to produce a function model via the graphical representation of a manufacturing system's functions and the interrelationships of those functions with each other (18:1-1). The $IDEF_0$ methodology is also useful in describing software systems and has been introduced as a graphical language for modeling software system requirements (17:3).

AFIT tailored the $IDEF_0$ methodology to better describe software systems by adding the requirement of a data dictionary (10:3). The data dictionary is a structured analysis modeling tool used to "organize the data elements that are pertinent to a system, with precise, rigorous definitions so that both the user and the system analyst will have a common understanding of all inputs, outputs, components of stores, and intermediate calculations" (28:189). The modified $IDEF_0$ methodology uses the data dictionary to describe the objects found in each $IDEF_0$ diagram. The data dictionary and the objects it defines are described further in the Requirements Analysis chapter. A review of the $IDEF_0$ diagram syntax can be found in Appendix H.

1.1.2 The C Version of SATool In 1987 Johnson completed a thesis at AFIT for which he built a computer-aided graphics editor called SATool written in the programming language C (15:1-8). SATool provides a graphics editor used to create $IDEF_0$ structured analysis diagrams and the capability to create and manipulate data dictionaries in accordance with the $IDEF_0$ requirements (10:3). However, SATool relied on the Sun workstation's graphics features to implement the user interface which reduced the portability of the tool to other systems (27:1-2). There were also several deficiencies observed in the user interface (27:1-2). These included,

among others, the difficulty to modify existing diagrams, the inability to create a comprehensive data dictionary, and the lack of a hierarchical structure for the creation, manipulation and deletion of the diagrams. In subsequent research, several changes to the SATool system were suggested. These included added features like portability, an improved user interface, and the ability to handle more than one diagram per session (15:5-12).

1.1.3 The Ada Version of SATool The suggested improvements to the SATool system brought about the development of SATool II. Portability of the system and a more user friendly interface were the two major improvements suggested. Additional features, like diagram syntax checking and automatic diagram layout, were also identified. This section provides an overview of the groundwork for the present design of SATool II based on these suggested improvements.

The portability issue was addressed by proposing a system written in Ada using the X Window System for the user interface. The X Window system does not mandate a particular user interface (23:79). "The purpose of the X Window system is to provide a network-transparent and vendor-independent operating environment for workstation software"(16:4). Network transparency implies that the application can run on whatever CPU is most convenient (16:5). In contrast, the windowing systems offered by Apple's MacIntosh and Microsoft Windows do require that the application adhere to a particular interface style (27:1-4).

After the completion of the C version of SATool, the structure of the system was re-evaluated by a group of graduate students at AFIT who took an object oriented approach to redesign of the system (1:641). An $IDEF_0$ diagram was now viewed as an entity composed of objects from two distinct models. This approach was used by Smith (25) who laid the groundwork to convert the user interface of SATool to the X Window System and developed an early version of what is now the essential model (17:1-2).

In 1990-91, Tevis (27), Kitchen (17), and Shyong (24) completed thesis work which created several building blocks for the updated version of SATool. Kitchen developed the essential model objects and operations and started work on the Ada based expert system used for diagram syntax checking. This work was later competed by Shyong. Tevis developed the drawing model objects and operations as well as the X Window/Ada Interface called the Machine-Independent Ada Graphical Support Environment (MAGSE). The essential model contains all the data dictionary information required to describe the objects identified in the $IDEF_0$ model (all object descriptions, relationship information, etc). The drawing model contains all the graphics information that is part of what the $IDEF_0$ diagram looks like. This information includes attributes such as the location and size of all boxes and lines on the diagram. The drawing model uses the X Window system as an interface between SATool II and its users. A detailed description of these models is provided in the Requirements Analysis chapter.

1.2 Problem Statement

The purpose of this thesis was to develop and implement the SATool II system by creating a graphical user interface (GUI) that tied together the essential and drawing models as well as the X Window/Ada interface created by Tevis, Kitchen, and Shyong. The result is a system that allows an entire $IDEF_0$ project to be created or loaded into the SATool II system, viewed, manipulated and stored with a user interface that is network-transparent and easy to use.

1.3 Assumptions

1. Kitchen's essential model and Tevis' drawing model are complete with respect to the types of objects, attributes and relationships among the objects they have developed.

2. The Ada interface source code supplied by Science Applications International Corporation (SAIC) works correctly when used for calling X Windows library functions from an Ada program (27:1-5).
3. The functions implemented in the X Window programming library perform as described in the X Window documentation (27:1-5).
4. Tevis' X Window/Ada interface provides all necessary functions and procedures to implement the SATool II graphical user interface and works correctly.

1.4 Scope and Limitations

This thesis concentrates specifically on the development of the graphical user interface for the SATool II system. This tool provides the required interfaces and system enhancements to facilitate the work performed by a user in the development of applications using the $IDEF_0$ methodology.

1.5 Standards

The SATool II application source code is documented using the guidelines and standards written in (11). The actual Ada coding practices used were object-oriented analysis, design, and implementation, loosely-coupled packages, consistent indentation, and consistent naming of packages, procedures, functions, and variable names (27:1-6).

1.6 Approach

The research was approached in four phases:

1. A review of work done by Tevis and Kitchen and research into the possible enhancements to the system.
2. Analysis and design of all necessary components of the graphical user interface that tie the essential and drawing model together.

3. Implementation of all the graphical user interface components.
4. Unit tests, overall system integration tests, and user evaluations to ensure the proper function of the final product.

1.7 Equipment and Software

The following equipment and software were used during this research:

1. Sun Microsystems Sun 3/110 workstation
2. AT clone personal computer.
3. BSD UNIX operating system (version 4) and MS-DOS 3.3
4. Verdix Ada compiler (version 5)
5. SAIC source code modules for the Ada interface to X
6. X Windows System library (version 11, release 4) (14)

1.8 Sequence of Presentation

This thesis is divided into seven chapters. The first chapter is the thesis introduction. Chapter 2 presents the findings of the literature review performed for this research effort. Chapters 3, 4, and 5 present the Requirements Analysis, Design and Implementation of the SATool II system. The SATool II test suite and results are presented in Chapter 6. Chapter 7 summarizes the thesis findings and presents several recommendations for further work to be done with the SATool II system project.

II. Literature Review

2.1 Introduction

This chapter presents several CASE tool enhancement options that could be used to improve the overall performance of the SATool II user interface. A better user interface was one of the suggestions given for the original tool. The following sections discuss the criteria used to select the proposed enhancements to the user interface as well as discuss several possible enhancements for the present system.

2.2 CASE Tool Enhancements

As computer hardware becomes more powerful and less expensive, the software developed for use in computer systems is becoming more complex. The traditional methods and tools used to develop smaller applications have become inadequate for use in the development of the larger, more complex software systems. "Historically, the most significant productivity increases in manufacturing or building processes have come about when human skills have been augmented by powerful tools. For example, one man and a bulldozer can probably shift more earth in a day than 50 men working with hand tools" (26:362). In the Software Engineer's world, one such class of tool is known as Computer-Aided Software Engineering (CASE) tools which can be used to extend the capabilities of Software Engineers by aiding the software development process. There are several types of CASE tools available, but the one of most interest to this research is the CASE workbench. A CASE workbench is a software engineering support tool which assists the analysis and design stages of the software development process by means of multipurpose diagram editing, design analysis and checking, query language facilities, data dictionary facilities, and report/forms generation (26:363).

Following is a review of current literature on CASE workbenches which identifies several features that can be used to enhance the CASE workbench environment.

The enhancements identified were selected based on a set of criteria compiled by Matingly (19:2-26 - 2-30). The purpose of this set of criteria is to be used in the evaluation of CASE tools. In a similar light, it can also serve as a set of guidelines for selecting enhancements that will enrich a CASE workbench environment. There are five categories identified that useful to this research effort:

- *Ease of Use* - Refers to the "user friendliness" of the tool. Does it have an online help facility? Does the graphical user interface have an easy to use menu system? Does it have safeguards for error prevention, and in case one occurs can the tool recover gracefully from it? Can the tool environment be controlled so as to shut off unwanted features?
- *Power* - Refers to performance features like the capability to easily modify diagrams and save the current work and be able to reload it into the system. This category also includes the ease of adding macros systematically to the system to increase the entire collection of possible operations.
- *Robustness* - Includes such criteria as consistency, adaptability and maintainability. Can objects be stored and retrieved consistently? Can the tool evolve with changing requirements? Can the tool be easily repaired once bugs are found in it?
- *Functionality* - Does the tool support all aspects of the methodology? Does the tool operate correctly and produce correct output?
- *Ease of Insertion* - How long does it take to learn the tool? How easy is it to integrate into the overall software development process.

This set of criteria was used to identify four enhancements that could increase the performance of the SATool II system in three of the above categories. The first two sections present features that can be useful in the design of the user interface to enhance ease of use and power. The last two sections present features that can be used to increase the functionality of the CASE tool itself.

2.3 Highlighting With Color

Highlighting is used to display information in special formats to emphasize its importance and to set aside special areas of the screen. Some of the most common forms of highlighting are blinking, sounds, color, and boxing (7:98-100). Color will be the focal point of discussion in this section.

Research in the management information systems and the reference disciplines have made several major findings (12:121):

- Color improves performance in a recall task.
- Color improves performance in a search-and-locate task.
- Color improves performance in a retention task.
- Color improves comprehension of instructional materials.
- Color improves performance in decision judgment task.

However, care must be taken to avoid the overuse of color. For example, it is easy to misuse color in the interface design and end up with displays that are error prone and unpleasing to look at. There are two common mistakes made when designing color interfaces (26:283-285):

- The interface designer tries to use color to communicate meaning.
- Too many colors are used in the display and/or the colors are used in inconsistent ways.

The problem with trying to communicate meaning through color is that there are no standard conventions on what the meaning of colors are. So, the meaning the designer wants a color to have may be misinterpreted. There is also the problem of dealing with color blind users. They may not be able to perceive that a particular

color is displayed. Therefore, color should be used only for highlighting purposes, to draw a user's attention to a certain part of the display.

The designer must also avoid the over-use of color as well as using colors that are too bright, and saturating the diagram with too many colors. Sommerville suggests a set of guidelines for the effective use of color in any system interface. The bottom line of these guidelines is that the designer should try to be as conservative as possible when designing color displays (6:67-77).

2.4 Online Documentation

Along with written documentation, a well designed online help facility can improve the productivity of users and increase their satisfaction with the software system (7:54-55). Online help includes user invoked help messages, status messages, prompts and error messages. The purpose of this facility is to allow the user to have greater control over the system by providing him with as much information as he needs to understand the system. Help messages should be tailored to the user's current context. The help message should be related to the action the user is currently performing. The help facility should also allow the user to pick the verbosity level of help messages to allow for various levels of expertise as users become more familiar with the system. Finally, messages should be positive rather than negative. They should use an active mode of address and should never be insulting or attempt humor (26:277).

2.5 Configuration Management

Configuration management is concerned with the development of procedures and standards for managing an evolving software system (26:552). A configuration management facility in a CASE workbench can be particularly helpful when controlling multiple design versions of a given project. Configuration management tools allow individual versions of a system to be retrieved support system building from

components, and maintain relationships between components, and their documentation (26:552). This type of control is difficult to achieve. However, if implemented appropriately, it can facilitate the implementation of other features like rapid prototyping (26:552).

2.6 Automatic Diagram Layout

One of the goals of this research was to determine an automatic layout algorithm for $IDEF_0$ diagrams. Following is a discussion of layout algorithms for data flow diagrams, which are similar in several respects to $IDEF_0$ diagrams.

Drawing data flow diagrams can be very time consuming even in the most user friendly of systems (21:11). An alternative to drawing the diagram from scratch is to have the user input the project information into a requirements database or repository and have the CASE tool draw out the diagrams based on that information. The overall objective when creating data flow diagrams whether manually or automatically is clarity (readability) of the diagram. Clarity is not easy to achieve because it is difficult to identify the important characteristics of the diagram. A balance must be achieved between the symbols, text, and white space in the diagram layout (21:11).

The approach taken by Prototzko et al (21) to solve the diagram layout problem is to access a project database to extract the system flow information. The data is converted to a directed-graph-like internal representation that allows the CASE tool to follow the system flow to and from any system flow object. This is followed by the placement of each of the data flow objects in a grid structure. A placement algorithm is then used to place all data flow objects in a grid structure. Finally the data flow arcs are placed between the objects by means of a routing algorithm (21:11-12).

Batini et al (2) takes a similar approach to produce a data flow diagram layout algorithm. Two types of graphic standards are identified for data flow diagrams. The straight line standard creates DFD's where all processes are connected by straight

lines. In this case the DFD is also called a bubble chart and the processes are identified with circles instead of boxes with rounded edges. The second standard is the grid standard that makes all process connections run along the lines of a rectangular grid in which the diagram is embedded (2:538-539).

The grid standard was chosen for the layout algorithm since it creates diagrams with high regularity and modularity (2:539). The placement algorithm takes into consideration the number of connections for each process and embeds each one into the grid by first placing the ones that have only one connection in the innermost part of the grid. The grid is viewed as a set of arrays of grid cells whose perimeter grows as the number of connections per process grows (2:539). This description is a simplistic view of the placement algorithm described in (2). The layout algorithm uses the following strategy (2:540-541):

- Find a two dimensional or planar representation for the DFD grid that tries to stay within five prescribed guidelines (2:539):

    1. Minimize the crossings between connections.
    2. Minimize the global number of bends in connection lines.
    3. Minimize the global length of connections.
    4. Minimize the area of the smallest rectangle covering the diagram.
    5. Place external boundary symbols so as to minimize crossings.

- Then give an orthogonal shape to the planar representation finding an orthogonal representation. This follows the second guideline.
- Finally, the grid embedding is completed by assigning integer lengths to line segments, according to guidelines three and four.

The layout algorithms discussed so far have been directed towards data flow diagrams. Even though there are several similarities between DFDs and $IDEF_0$

diagrams, the restrictive syntax rules of the $IDEF_0$ methodology may provide a more direct approach to laying out $IDEF_0$ diagrams. Further research is suggested in this area.

2.7 Summary

This chapter presented several enhancement options for the SATool II system. A set of CASE tool evaluation criteria was identified to aid in the selection of these enhancement options. SATool II must be made attractive to the user if it is going to be used in a software development process that includes the $IDEF_0$ methodology. The evaluation criteria was used to determine what enhancements would make SATool II more attractive. The enhancement features presented in this literature review addressed this concern by advocating a good user interface design by means of appropriate highlighting and help facilities. It also presented the options of automatic diagram layout and configuration control as functional enhancements to allow the speedup of the software development process.

III. Requirements Analysis

3.1 Introduction

This chapter presents an analysis of the requirements for the design and implementation of the graphical user interface (GUI) for the SATool II system. It also presents a review of the requirement models created to capture the information found in the $IDEF_0$ diagrams.

3.2 General Requirements

Following is a summary of the system requirements identified for SATool II (17:34):

1. All parts of SATool II must be implemented in Ada.
2. The tool must have a graphical user interface (GUI).
3. The tool must be implemented on a workstation supporting X-Windows and Ada.
4. The tool must provide for the creation, editing, and output of $IDEF_0$ diagrams (i.e., the manipulation of $IDEF_0$ syntax).
5. The tool must provide for the creation, editing, and output of the AFIT Data Dictionary formats.
6. The tool must provide for the storage of the essential data model information of an $IDEF_0$ model that is separated from the stored drawing data model information.
7. The tool must provide for the storage or automatic generation of the drawing data model information (i.e., the diagrams) of an $IDEF_0$ model that is separate from the stored essential data model information.

8. The tool must be integrated with an Ada based expert system for the purpose of identifying $IDEF_0$ syntax and modeling errors.
9. The tool must allow for the user to terminate work on an $IDEF_0$ model, leaving it in an unfinished state. For example creating an activity with no connecting data elements leaves the $IDEF_0$ model in an incomplete state.
10. The tool must be developed using an object oriented design methodology in order to assess its potential in the construction of an Ada based CASE tool.

An analysis of these requirements suggests five subproblems to be solved:

1. The development and implementation of an object model to create, retrieve and restore $IDEF_0$ essential model information (requirements 1, 4, 6, 9, and 10) (17:35).
2. The development and implementation of an interface to an Ada based expert system (requirement 8) (17:35).
3. The development and implementation of an object model to create, retrieve and restore $IDEF_0$ drawing model information (requirements 1, 4, 7, 9, and 10).
4. The development and implementation of a method to create, retrieve and output AFIT Data Dictionary information (requirement 5) (17:35).
5. The development and implementation of a graphical user interface using the X Window system and Ada that is capable of manipulating the essential and drawing model information in order to present a complete, homogeneous picture of an $IDEF_0$ project to the user (requirements 2, 3 and 4).

The first three subproblems were addressed and satisfied by Tevis (27), Kitchen (17), and Shyong (24) during their research. The work they completed became the foundation for the final design and implementation of the GUI system. Kitchen

provided the essential model that would capture the data dictionary information of the $IDEF_0$ diagrams. Tevis developed the drawing model for the $IDEF_0$ diagrams as well as the X-Window-Ada interface. Shyong implemented the expert system specified by requirement 9. The fourth and fifth subproblems defined the major thrust on requirements for the design of the GUI for the SATool II system.

3.3 Requirements Analysis for the Graphical User Interface

Based on the requirements specified in the previous section, the SATool II system must be capable of handling an entire $IDEF_0$ project during any given session. This implies that the system must be able to maintain information on the project's hierarchical decomposition. These requirements also specify that the data stored for each project must be easily created, modified, stored, and reloaded. The SATool II system must maintain a data dictionary for any project that is loaded into the system as well as show the project information in a graphical format similar to the format given for the manual drawings. The requirements also suggest that an automatic diagram layout feature by a part of the final product.

Since the GUI design is based on the two models designed to capture the information found in the $IDEF_0$ diagrams, the following two sections review the essential and drawing model composition. There are two basic types of objects present in an $IDEF_0$ diagram, activities and data elements (see Appendix H for details on $IDEF_0$ syntax). The essential model maintains the logical relationships between activities and data elements as well as the data dictionary information for each data element and activity. The drawing model maintains the physical (location, shape, etc.) information on data elements and activities.

3.4 Essential Model

The present version of the essential model description was designed by Kitchen (17). Figure 1 and Figure 2 show the entity relationship diagrams that describe this

model. Figure 1 describes the attributes of an activity and its relationship with other activities and data elements. Figure 2 completes the picture by showing the attributes of the data elements and its relationship with other data elements and activities. Overall, there are six entities and twelve relationships defined by the essential model.

The six objects defined by the essential model are:

1. Project: This entity refers to the project the activity or data element is associated with. Its one identifying attribute is *Pname* and contains the project name.
2. Activity: An activity represents a function performed by a given system. There are seven attributes associated with the activity:
   - *Name*: Used as the unique identifier.
   - *Activity Number*: Used to determine the location of the activity in the diagram hierarchy.
   - *Description*: Used to describe the activity's function.
   - *Version*: records the current version number of the activity.
   - *Date*: indicates the creation date of the activity.
   - *Changes*: captures what has changed between this activity and the previous version.
   - *Author*: The creator of the activity.
3. Historical Activity: This is an activity that belongs to another project and is "called" by the activity in this project. Its two identifying attributes are *Project* and *Activity Number*. *Project* represents the project the activity comes from and *Activity Number* shows which activity in the project to address.

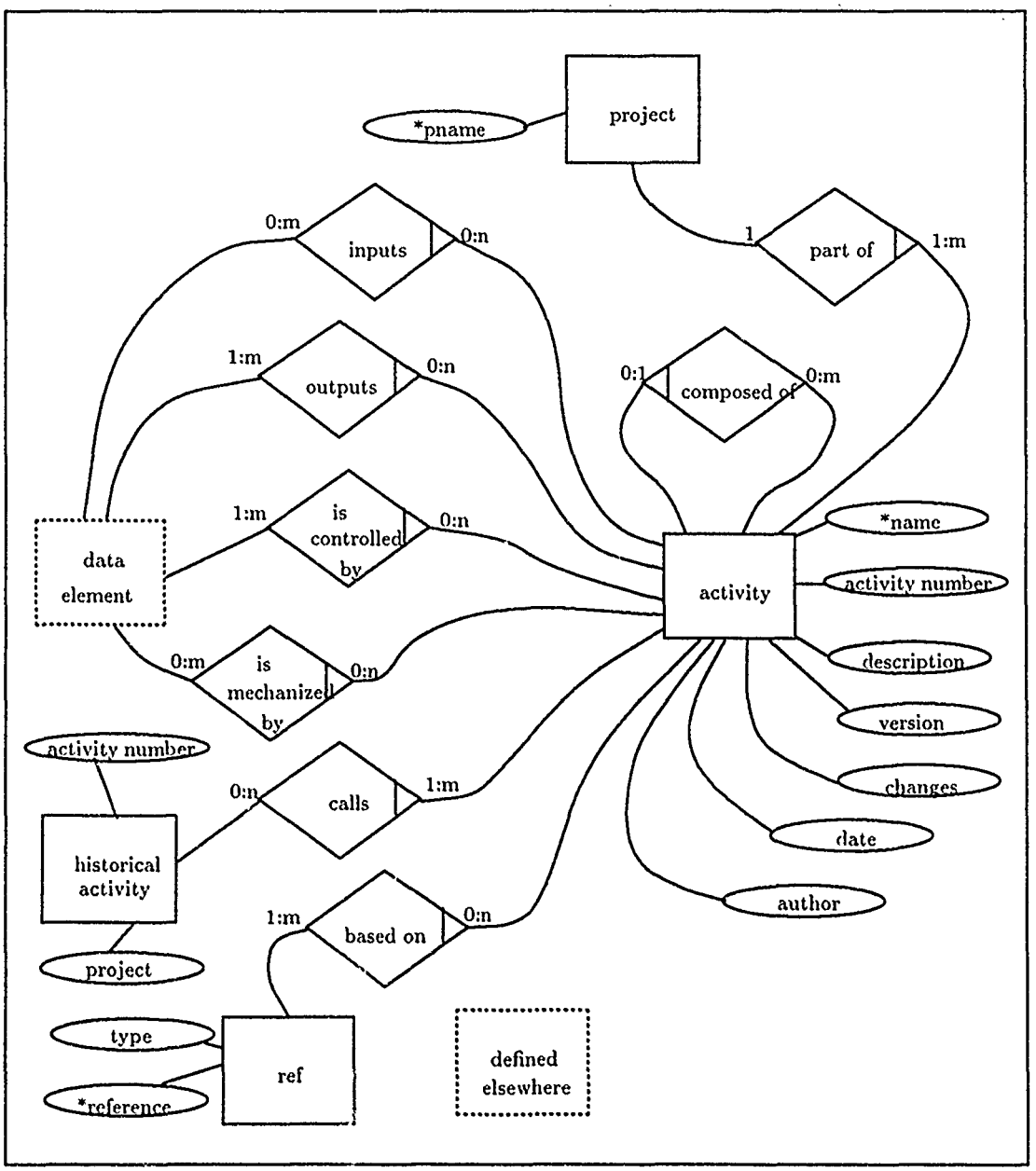


Figure 1. IDEF$_0$ Activity Essential Data Model

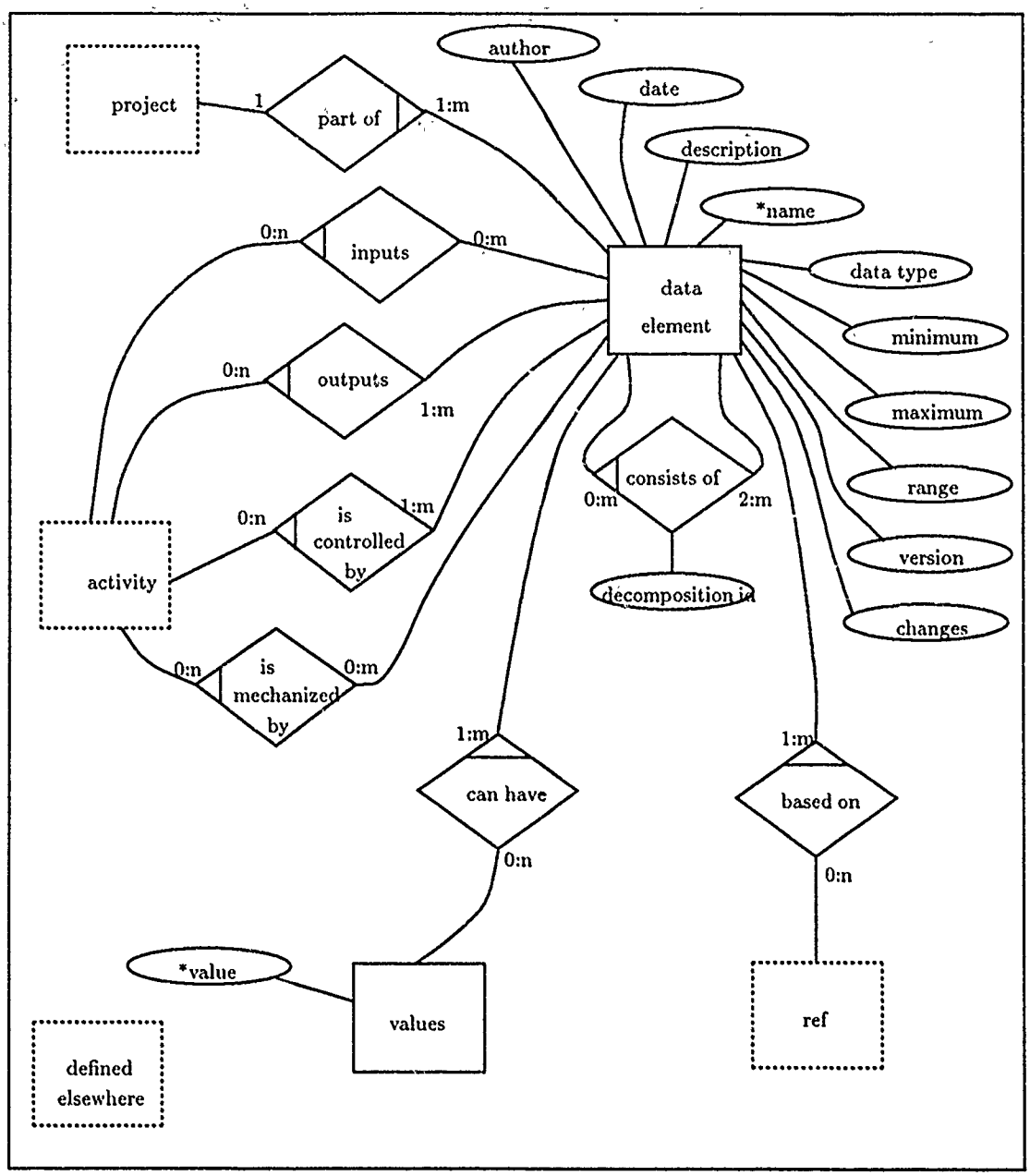


Figure 2. IDEF$_0$ Data Element Essential Data Model

4. Ref: Captures any reference associated with an activity or data element. The identifying attributes are *Reference* and *Type*. The reference entity allows an activity or data element to be associated to written documents other than those defined by the $IDEF_0$ methodology. This allows for a better description of the object and its function.

5. Data Element: Data Elements are used to define the data that is passed from activity to activity. This data can be simply information passed, yet, it can also be defined as a control mechanism for the activity. There are ten attributes associated with the data element:

   - *Name*: Identifying Attribute is the name of the data element.
   - *Description*: Gives a description of the data element.
   - *Data Type*: Indicates the type of data. For example, is it a character string or an integer.
   - *Minimum*: Holds the minimum value the data element can take.
   - *Maximum*: Holds the maximum value the data element can take.
   - *Range*: The data value range (if applicable).
   - *Version*: The current version of the data element.
   - *Changes*: How is this data element version different from previous version.
   - *Date*: Date of creation of present version.
   - *Author*: Name of person who created this entity.

6. Values: Used for data elements that do not consist of other data elements and have enumerated values. Each value object is identified by a single attribute *Value* which contains one of the values the data element can take on.

Of the twelve relationships defined by the ER diagram, two share the same name and definition (*Part_Of* and *Based_On*). So, they are only described once here. The relationships are defined in the following manner:

1. Part_Of: A data element or an activity are part of exactly one project. A project may contain one or more activities or data elements.
2. Inputs: An activity can have zero or more data elements as input.
3. Outputs: An activity must have at least one output data element.
4. Is_Controlled_By: An activity must have at least one control data element.
5. Is_Mechanized_By: An activity can have zero or more mechanism data elements.
6. Composed_Of: Each activity has one parent activity, except for the A-0 activity which has no parent. On the other hand an activity can have zero or more children activities.
7. Consists_Of: This relationship is used for building pipelines (bundles/joins and forks). It shows that a data element can be broken down into other data elements (fork) and at the same time a data element can be part of a group of data elements that feed into a parent data element (bundle/join).
8. Calls: An activity can call zero to many historical activities and at the same time a historical activity can be called by zero or more activities.
9. Based_On: Shows that a data element or activity can have zero or more references and at the same time a reference can be related to zero or more activities or data elements.
10. Can_Have: Associates a data element that consists of enumerated values to those values. Each value is associated with one or more data elements.

3.4.1 AFIT Data Dictionary Format Even though the $IDEF_0$ methodology does not explicitly require a data dictionary, it does require a glossary (10:73). AFIT introduced the use of the data dictionary to support this requirement. The present version of the AFIT data dictionary format was established by Kitchen (17). A data

dictionary entry is required for each activity and data element. Table 1 shows the data dictionary format for an activity. Table 2 shows the format for a data element.

Format	Field	Description	Size
(S)	Name	activity name	C25
(S)	Type	defaults to ACTIVITY	N/A
(S)	Project	project name	C25
(S)	Number	activity number of this activity	C20
(ML)	Description	text description	C60
(MF)	Inputs	data element name	C25
(MF)	Outputs	data element name	C25
(MF)	Controls	data element name	C25
(MF)	Mechanisms	data element name	C25
(G)	Calls:		N/A
(S)	...Project	different or same project name	C25
(S)	...Activity Number	activity number of called activity	C25
(S)	Parent	name of parent activity	C25
(ML)	Reference	reference cite	C60
(S)	Ref Type	type of the reference	C25
(S)	Version	version of this entry	C10
(ML)	Changes	a history of the changes	C60
(S)	Date	mm/dd/yy (date of creation)	C8
(S)	Author	author's name	C25

Table 1. Data Dictionary Entry Format for Activity

The field classifications for the AFIT data dictionary formats shown are as follows (17:52):

- (S) - The field consists of a single field that appears on a single line.
- (ML) - The field consists of a single field that appears on one or more lines.
- (MF) - The field consists of one or more fields, and each is a single field that appears on a single line.
- (G) - The field consists of two or more fields grouped together and multiple groups are allowed. However, each group member is still a single field that can only appear on a single line.

Format	Field	Description	Size
(S)	Name	data element name	C25
(S)	Type	defaults to DATA ELEMENT	N/A
(S)	Project	project name	C25
(ML)	Description	text description	C60
(S)	Data Type	type of the data, if known	C15
(S)	MIn Value	minimum data value, if known	C15
(S)	Max Value	maximum data value, if known	C15
(S)	Range	range of values, if applicable	C60
(MF)	Values	enumeration values, if applicable	C25
(MG)	Decomposition:		N/A
(S)	...Part Of	name of parent data element	C25
(MF)	...Composition	subcomponent data element names	C25
(MG)	Sources/Destinations:		N/A
(MF)	...Outputs	activity(s) where output	C25
(MF)	...Inputs	activity(s) where input	C25
(MF)	...Controls	activity(s) where a control	C25
(ML)	Reference	reference cite	C60
(S)	Ref Type	type of the reference	C25
(S)	Version	version of this entry	C10
(ML)	Changes	a history of the changes	C60
(S)	Date	mm/dd/yy (date of creation)	C8
(S)	Author	author's name	C25

Table 2. Data Dictionary Entry Format for Data Element

- (MG) - Two or more fields are grouped together and multiple groups are allowed. Each group member is permitted to be a single field, a single field of multiple lines, or multiple fields. Therefore, each group member must be classified with either a 'S', 'ML', or 'MF' field classification.

3.5 Drawing Model

The present version of the drawing model is a modified version of the drawing model established by Tevis (27) (see Figure 3 through Figure 5). While reviewing the drawing model several inadequacies were found in the type of attributes each object was given to store necessary information. All previously identified objects are still found in the present model (Figure 6). However, the terminator object is decomposed into its subclass objects (simple turn, junctor, and arrow) since they each have a different set of attributes (Figure 7). The relationships between the objects were simplified by deleting the relationship between a label and a historical activity. A label now identifies a data element, a squiggle, or a footnote. The historical activity relationship is taken care of by the essential model. It has no physical attribute that needs to be shown in the $IDEF_0$ diagram, and therefore, need not be repeated in the drawing model. The relationship between the verbal addition objects and the squiggle were also changed. Footnotes are now identified by labels. Squiggles can either point to a label or a note.

Following is a description of the objects defined by the revised drawing model:

1. Diagram: A diagram can be composed of zero or more drawable objects. It can be derived from a single box, but does not have to be. This allows for the creation of the A-0 diagram which has no parent activity (or box). It is not considered a drawable object. Instead it is a template for objects to be drawn on. Its attributes are:

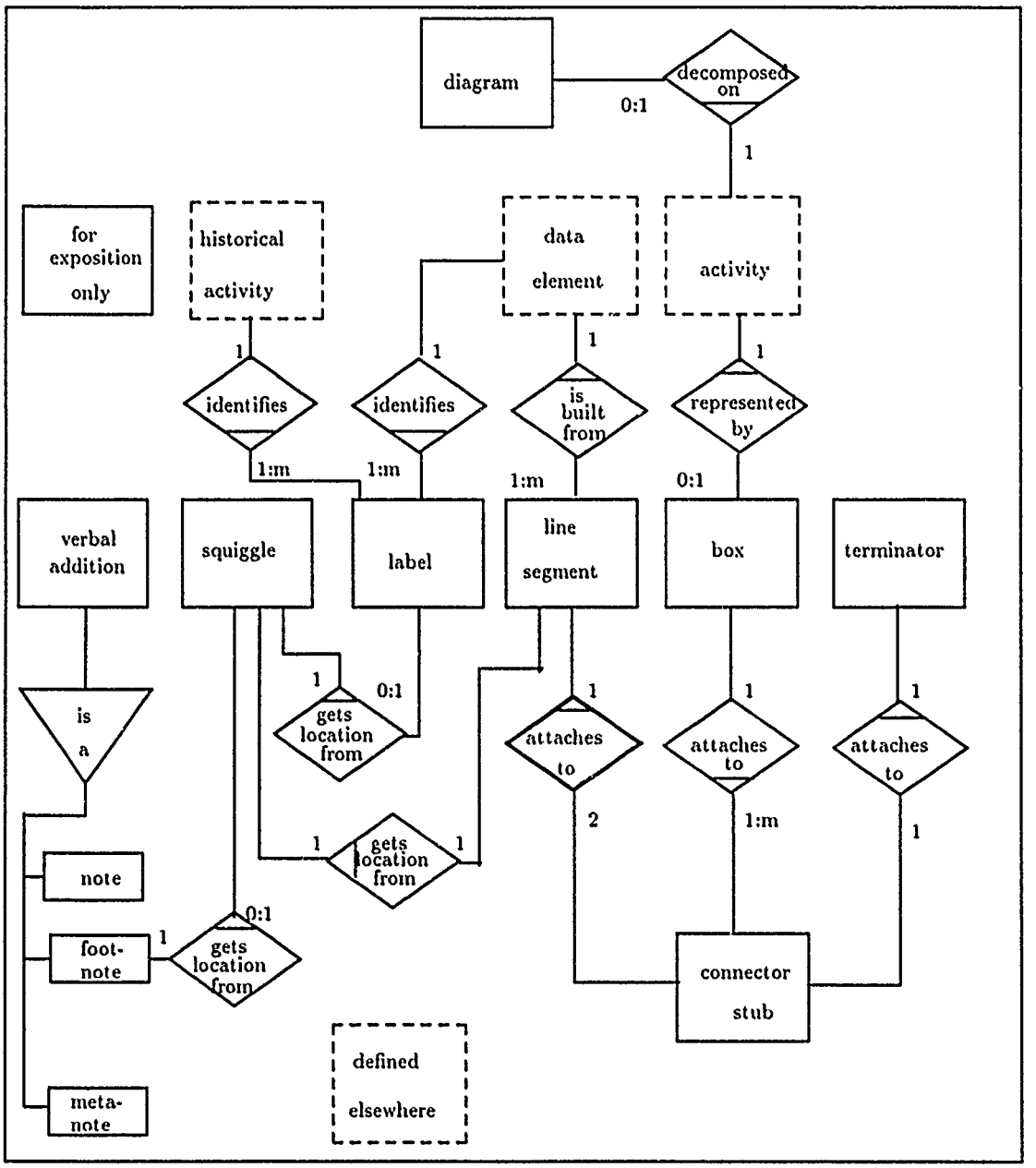


Figure 3. Original IDEF$_0$ Drawing Model Relationships

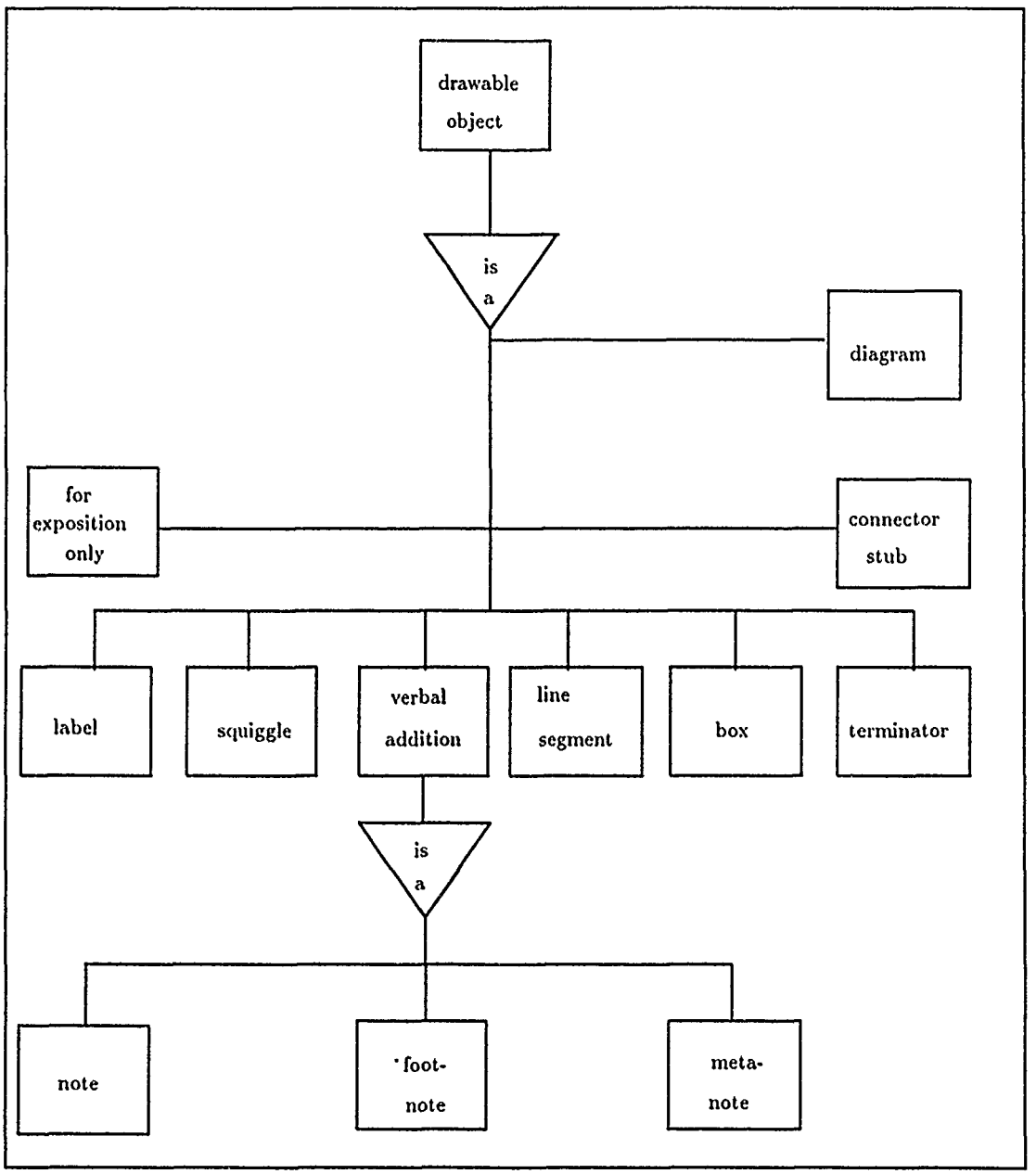


Figure 4. Original IDEF$_0$ Drawing Model Classes

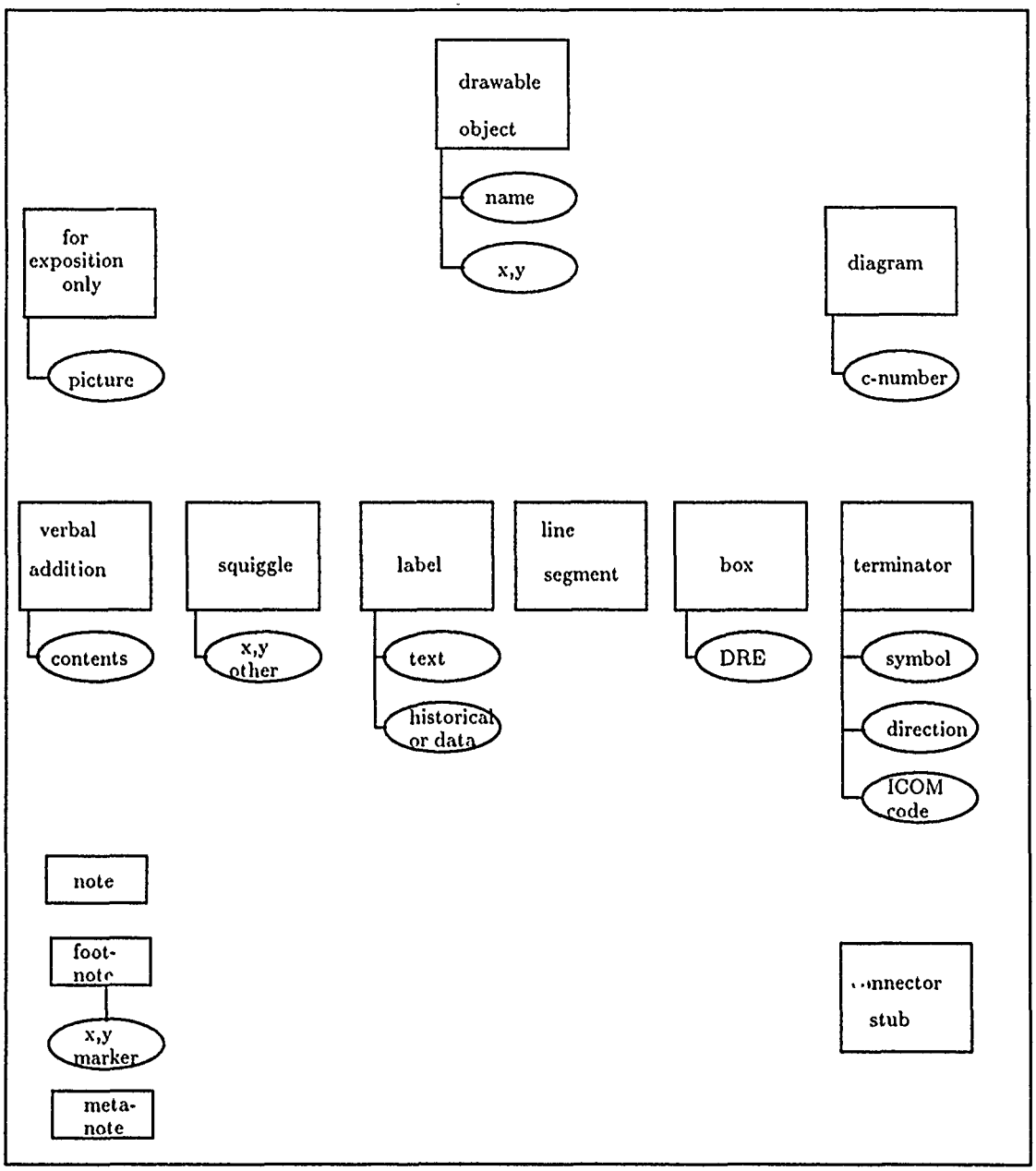


Figure 5. Original IDEF$_0$ Drawing Model Entities and Attributes

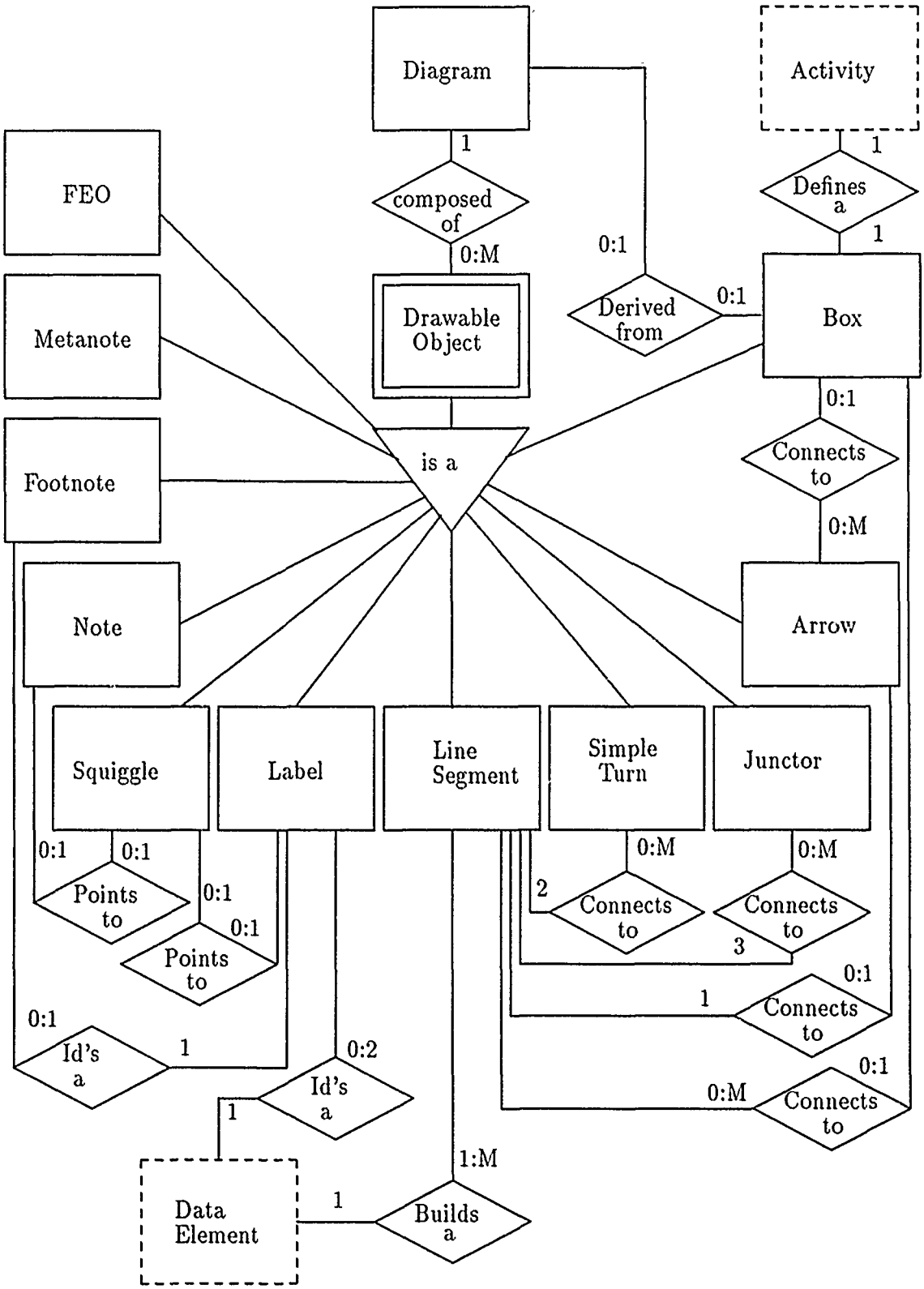


Figure 6. Revised IDEF$_0$ Drawing Model Entity Relationship Diagram

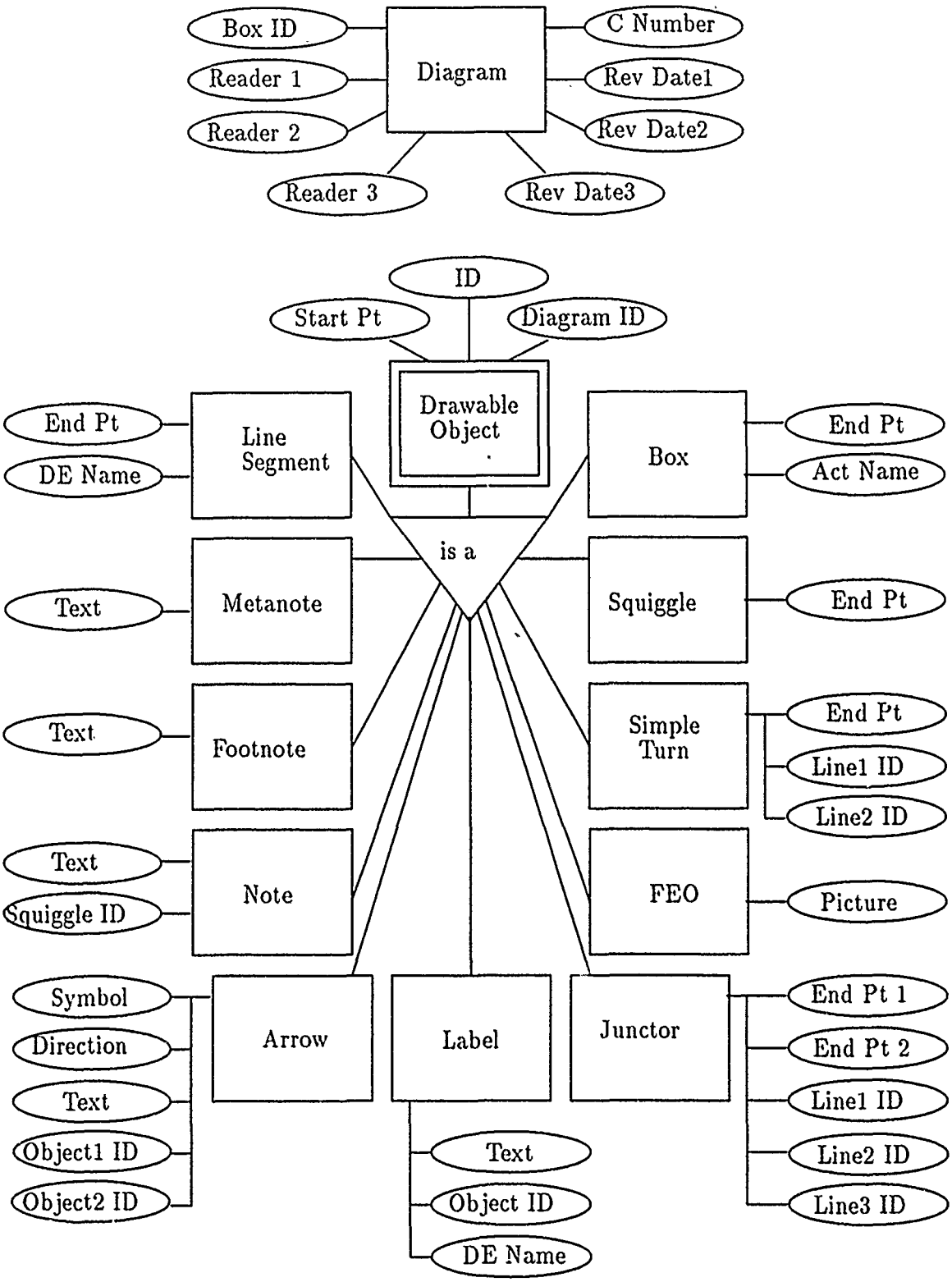


Figure 7. Revised IDEF$_0$ Drawing Model Entities and Attributes

- *Box_ID* - The unique identification string of the activity box this diagram is derived from.
- *C_Number* - It is the activity number of the parent activity.
- *Reader_1* - Name of the first person that reviewed the diagram.
- *Reader_2* - Name of the second person that reviewed the diagram.
- *Reader_3* - Name of the third person that reviewed the diagram.
- *Rev_Date1* - Date of first review.
- *Rev_Date2* - Date of second review.
- *Rev_Date3* - Date of third review.

2. Drawable_Object: A drawable object is any one of the other eleven objects shown in the ER diagram: FEO, metanote, footnote, note, squiggle, label, line segment, simple turn, junctor, arrow, box. It has three attributes that are globally shared by all drawable objects:

   - *ID* - The unique identification string for each object in the system.
   - *Start_Pt* - Location of the object on a diagram. This is the only location information needed for some objects, like FEO's, notes, and arrow heads. Other objects, like line segments and squiggles require an end point also. Those attributes are identified for each object individually.
   - *Diagram_ID* - The unique id string for the diagram that contains a particular object.

3. FEO: For Exposition Only (FEO) is an indicator that there is separate figure associated with this diagram. For example, a context diagram. Its only attribute is *Picture*. This is a reference id to that other figure associated with the diagram.

4. Metanote: A note that provides information about the diagram as a whole. Its only attribute, *Text*, holds the string of information the user wishes to convey about the diagram.

5. Footnote: A note that is placed at the bottom of the diagram. It too has only one attribute, *Text*. It is used to clarify any ambiguities that may be perceived with the objects and their relationships within the diagram.

6. Note: A small message used to help clarify a particular part of the diagram drawing. Its purpose is the same as that of a footnote. The only difference is that a footnote is placed at the bottom of the diagram and a note is placed among the diagram objects. There is usually a squiggle associated with a note. Its used to show what area the note is referring to in the diagram. Its attributes are:

    - *Text* - Holds a descriptive string about an area of the diagram.
    - *Squiggle_ID* - Unique squiggle identifier that associates the note with a given area of the diagram.

7. Squiggle: used to assist readability in a crowded part of the diagram. It usually points to a label or a note. Its only attribute is *End_Pt*. Indicates how long the squiggle is.

8. Label: Used to identify a data element by providing its name. It is usually associated with a line segment; but, in the case of crowding, a squiggle can be used as a link between the line segment and the label. A label can also be used to identify a footnote by providing the footnote number. A squiggle is always used to point to the label when the label is associated with a footnote.

    - *Text* - Contains the label text, it could be a data element name or a footnote number.

- *Object_ID* - The unique id of the line segment or squiggle it is associated with.
- *DE_Name* - Name of the data element associated with this label. This could be a blank field if the label is associated with a footnote instead.

9. Line Segment: A data element consists of one or more line segments that are connected to each other by simple turns or junctors and are finalized by an arrow. Its attributes are:

   - *End_Pt* - Determines the length of the line segment.
   - *DE_Name* - The data element this line segment is associated with.

10. Simple Turn: A data element can be constructed of one or more line segments. These line segments can be connected to each other by simple turns. All the line segments connected by simple turns define the same data element.

    - *End_Pt* - Determines direction of the simple turn.
    - *Line1_ID* - Line segment id of line attached at the start point of the simple turn.
    - *Line2_ID* - Line segment id of the line attached at the end point of the simple turn.

11. Junctor: Line segments can also be connected to each other via junctors. These differ from simple turns in that they are usually used to define the members of a pipeline. So, a junctor can actually be connecting three different data elements.

    - *End_Pt_1* - Determines direction of the junctor.
    - *End_Pt_2* - Determines direction of the junctor.

- *Line1_ID* - Line segment id of line attached at the start point of the junctor.
- *Line2_ID* - Line segment id of the line attached at the first end point of the junctor.
- *Line3_ID* - Line segment id of the line attached at the second end point of the junctor.

12. Arrow: An arrow is what should really be called a terminator because the presence of an arrow determines whether or not a data element has been completely defined. It is always attached to a line segment on one end. The other end can be attached to an activity box or left blank.

    - *Symbol* - The arrow can have several shapes (see Figure 55 in chapter 2). It can be a simple arrow-head, a tunnel arrow, a to_all or a from_all.
    - *Direction* - Determines which way the arrow is pointed.
    - *Text* - Used to identify the single character allowed in the to_all and from_all arrows.
    - *Object1_ID* - Unique id of a line segment, a box or a null object.
    - *Object2_ID* - Unique id of a line segment, a box or a null object.

13. Box: Defines the location of an . ctivity in the diagram. It has two attributes:

    - *End_Pt* - Determines the size of the box.
    - *Act_Name* - Unique activity id of the associated activity.

There are thirteen relationships shown in the drawing model ER diagram. Of these thirteen, there are only seven distinct relationships:

1 Defines: For every activity there is one box defined.

2. Derived_From: A diagram may be derived from an activity box, except for the A-0 diagram which has no parent activity. On the other hand a box can have zero or one diagram associated with it.

3. Points_To: A squiggle points to a label or a note.

4. Id's: A label id's a data element by keeping its name or a footnote by keeping the number of the particular footnote at the bottom of the diagram.

5. Connects_To: All the subcomponents of a data element are associated to each other via the connector stubs.

6. Builds_A: A data element is composed of one or more line segments. In turn each line segment must have only one data element associated with it.

3.6 Summary

This chapter presented the SATool II system requirements and the $IDEF_0$ models that were created to satisfy those requirements. One of the goals of these requirements was to create an object based system. Research done by Kitchen (17) and Tevis (27) identified the essential and drawing models needed to construct an object based system for SATool II. Shyong (24) completed work that fulfilled the requirement for an expert system. The drawing model was revised by redefining the relationships between objects and the attributes of those objects. The essential model and the AFIT data dictionary format remain as defined in (17). The requirements specified for the SATool II system as well as the structure of the essential and drawing models are the primary factors influencing the GUI design and implementation.

IV. SATool II System Design

4.1 Introduction

The design of the SATool II system is divided into four major parts: the Essential Subsystem, the Drawing Subsystem, the Machine-Independent Ada Graphical Support Environment (MAGSE) Subsystem, and the Graphical User Interface (GUI) Subsystem. The object oriented design (OOD) process methodology used throughout the development of SATool II is based on (5). The process is divided into four steps (5:190):

- Identify the classes and objects at a given level of abstraction.
- Identify the semantics of these classes and objects.
- Identify the relationships among these classes and objects.
- Implement these classes and objects.

The identification of the object classes, the objects and the relationship between them was accomplished via the analysis of the Entity-Relationship Diagrams described in chapter 3 for the essential and drawing models. The semantics of the classes were defined via the use of the Generic Multiple Object Manager. This manager will be discussed in the next section. The implementation of the object classes and objects is discussed in the next chapter.

4.2 Generic Multiple Object Manager

The Generic Multiple Object Manager is a modified version of the Booch component *Queue Nonpriority Balking Sequential Unbounded Unmanaged Iterator* (4:166-169). Its function is to maintain a sequential list of items. The abstract data type (ADT) for this manager describes the operations that can be performed on

objects that are managed by it. Following is the abstract data type for the Generic Multiple Object Manager:

```
Structure Generic_Multiple_Object_Manager (manager,
                                           object,
                                           boolean,
                                           pointer)
Declare
    Create()  =>                                  manager
    Clear(manager) =>                             manager
    Add_Object(item, manager, pointer) =>         manager', pointer'
    Set_Object(pointer, item) =>                  manager'
    Remove_Object(manager, pointer) =>            manager', pointer'
    Is_Equal(manager, manager') =>                boolean
    Is_Empty(manager) =>                          boolean
    Length_Of(manager) =>                         natural
    Initialize_Iterator(pointer, manager) =>      pointer'
    Get_Next(pointer) =>                          pointer'
    Value_Of_Object(pointer) =>                   object
    Is_Done(pointer) =>                           boolean
end Generic Multiple Object Manager
```

There is a separate manager declared for each type of object identified in the Essential and Drawing Subsystems. The manager can add an object, modify it once it's been added, view the contents of the object, or delete it from the list.

4.3 Essential Model Design

The Essential Subsystem is composed of seven object managers (Figure 8). They are based on the essential model Entity-Relationship diagrams shown in Chapter 3 (see Figure 1 and Figure 2). These figures identified 6 entities and 10 types of relationships among the entities. Based on the domain analysis done by Kitchen (17), those entities and relationships were reduced to seven object classes (types), six of which have to be tracked via an object manager like the one described in the previous section.

The project manager maintains the name of the project being held in the SATool II environment. Since there is only one project in the SATool II system at any time, a multiple object manager was not required. The activity, historical activity, and data element object classes are a reflection of the entities described in the previous chapter.

Three relation classes were identified as requiring a manager. The Calls relation manager handles the relationships that are identified between an activity and a historical activity. The Consists-Of relation manager handles the relationships that are identified between an activity and other activities or between a data element and other data elements. The final manager identified was the ICOM relation manager. This manager maintains all the objects from the ICOM relation class. This class is a summary of the Input, Control, Output, and Mechanism relationships identified between an activity and a data element in the essential model ER diagram.

An integral part of the Essential Subsystem is the Expert subsystem designed by Kitchen (17) and Shyong (24). This subsystem is composed of four object classes whose compound function is to determine if the essential model objects created for a particular project follow the rules set by the $IDEF_0$ methodology. Its operations answer questions about the logical relationships between the essential model objects and questions about the syntax adherence of those objects to those prescribed by the $IDEF_0$ methodology.

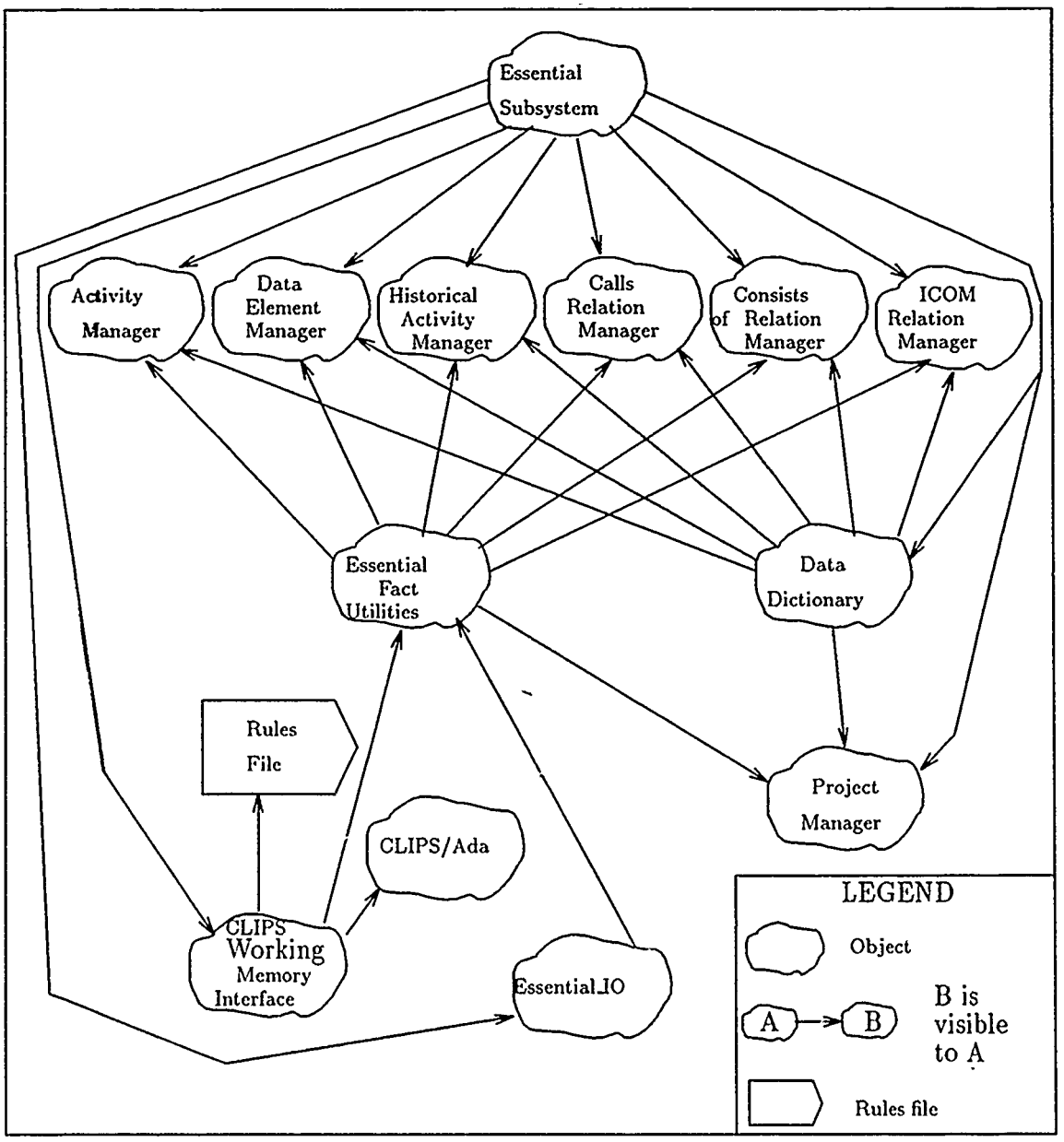


Figure 8. Essential Model Design

4.4 Drawing Model Design

The Drawing Subsystem is composed of twelve object managers (Figure 9). They are based on the drawing model Entity-Relationship diagram shown in Figure 6 (chapter 3). Each object class in the subsystem corresponds to a drawing entity identified in the ER diagram. Each manager is an instantiation of the Generic Multiple Object Manager. The general operation of this generic manager was described in 4.2. The Drawing Utilities object shown in Figure 9 is used to manage input and output of a given project's drawing model. The operations of this manager are to store and retrieve the drawing objects to and from a storage file into the SATool II editing environment.

4.5 MAGSE Design

The MAGSE was designed and implemented by Tevis (27). It was designed to serve as a general purpose interface between an Ada application and the X-Window system. Its present function is to serve as an interface between the SATool II system and the X Window System. Figure 10 shows the overall design of the MAGSE subsystem. There are seven object classes in the subsystem (27:3-20 - 3-22):

1. *Drawing Primitive* - This class contains several basic objects that can be placed on an X-Window screen: lines, boxes, circles, pixels, and text strings. The operations that can be performed on these objects are: Draw_Object and Erase_Object. These basic objects are used by the SATool II system to build the $IDEF_0$ diagram objects.
2. *2-D Plane* - This class contains a two dimensional plane. The operations that can be performed on it are: Set_X_Y_Dimensions, Clip_Complex_Primitive on the plane, and Render_Complex_Primitive on the plane.
3. *2-D Matrix Stack* - Contains a stack that stores matrices used to perform two-dimensional transformations. It's operations include: Push_Matrix, Pop_Matrix,

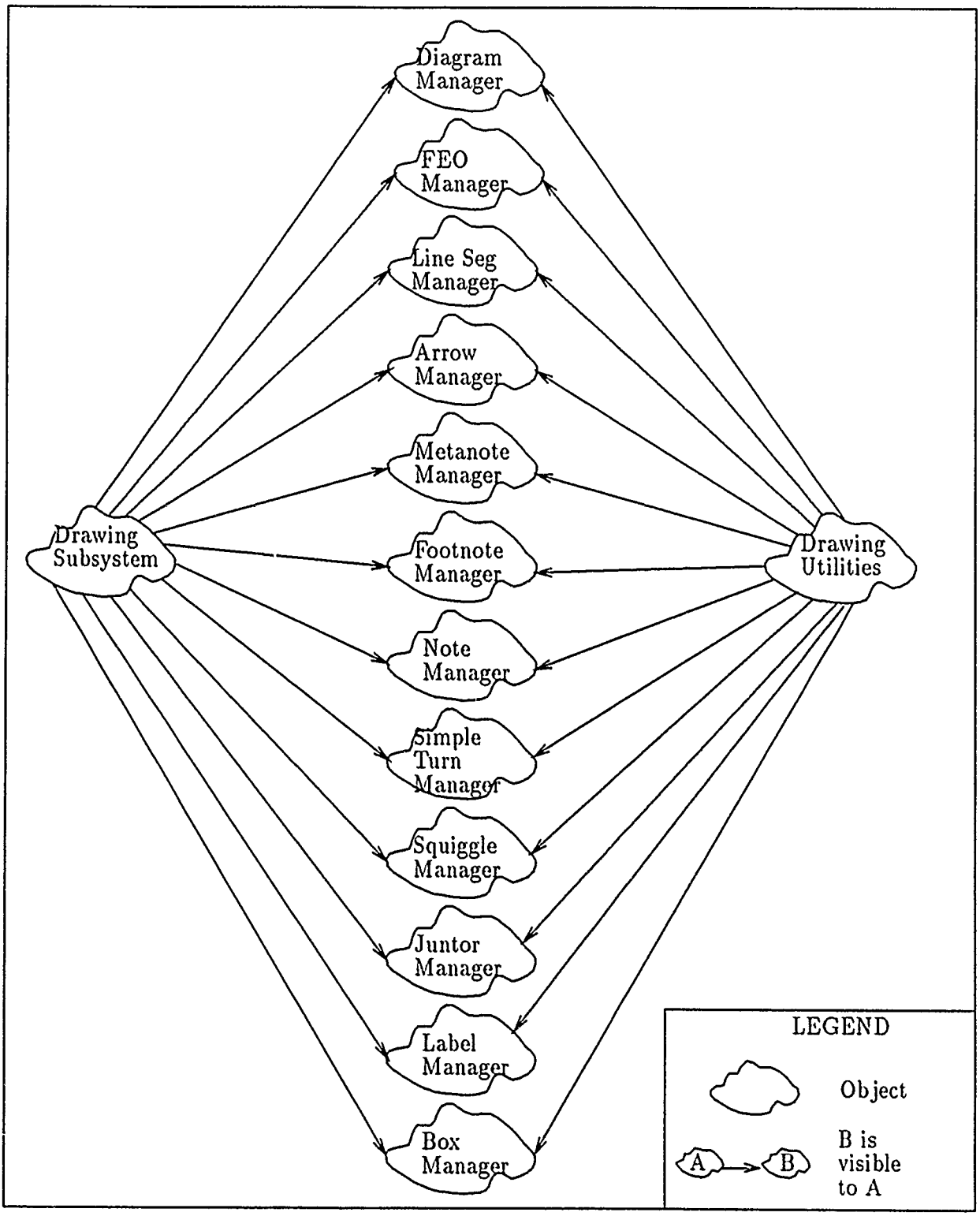


Figure 9. Drawing Model Design

Multiply_Matrix with the matrix on the top of the stack, Rotate, Scale, and Translate with a matrix at the top of the stack.

4. *3-D Pyramid* - Contains a three-dimensional perspective pyramid. It sets the X,Y,Z dimensions of the pyramid, the viewing location and perspective of the pyramid, and clips and renders complex primitives in the pyramid.

5. *3-D Matrix Stack* - Contains a stack for storing matrices used in performing three dimensional transformations. The operations are the same as for a 2-D Matrix Stack.

6. *Input Device* - The class contains a keyboard a cursor and a three button mouse. It's operations include: Read_Keyboard_Input, Get_Cursor_Position, Detect_Mouse_Movement, Detect_A_Button_Click, and Detect_Window_Events.

7. *Window Manager* - Contains a window manager object. There are seven window types: drawing window, acknowledge window, confirm window, dialogue window, column menu window, sign window, and text window. The manager can create, move, hide, display, store and delete these window types.

4.6 Graphical User Interface Design

The design of the GUI takes into consideration the requirements specified in 3.2 for the fourth and fifth subproblems, the design of the graphical user interface for the original version of SATool, and a review of several other graphical user interfaces (27:3-24). The result is a design that has four separate managers (Figure 11):

1. *Project Manager* - Contains the one project object that is present in the SATool II system environment. This manager is responsible for ensuring that an existing project can be loaded for modifications and saved. It also allows the user to create a project from scratch. This manager also has the capability of transforming a project that exists only in its essential model form (i.e. no

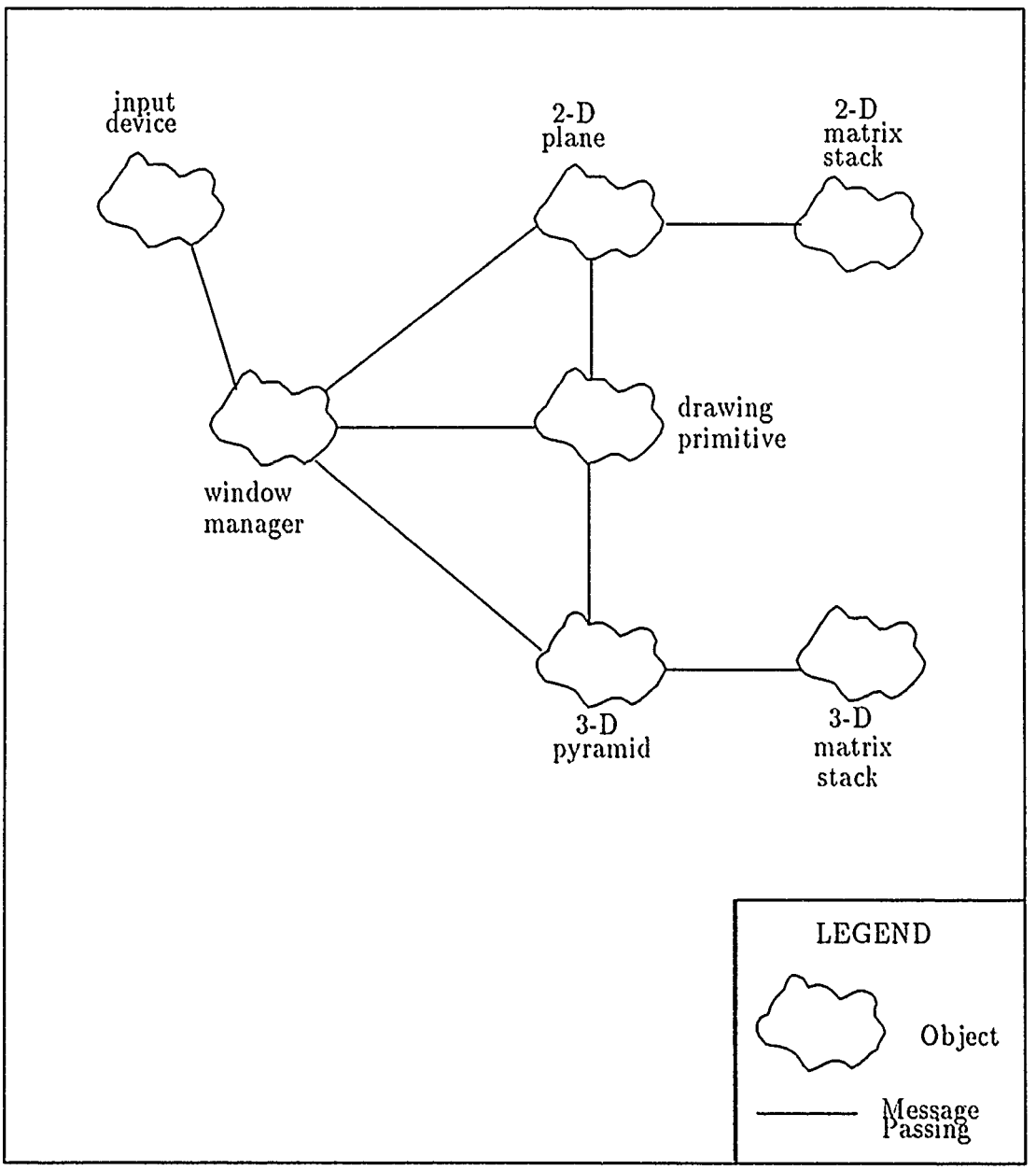


Figure 10. MAGSE Design

drawing objects defined) and creates the diagram model. This project manager is not the same as the project manager defined for the essential subsystem. This project manager is in charge of handling the essential subsystem objects as well as the drawing subsystem objects.

2. *Diagram Manager* - Manages the diagrams that are graphically presented to the user. This manager ensures that the user can create and delete $IDEF_0$ diagrams in the prescribed hierarchical format. It allows the user to view and modify diagrams in a hierarchical order as well as allowing the user to access a specific diagram without having to go through the entire hierarchy. Like the project manager above, this diagram manager views a diagram as consisting of both the essential and drawing model objects.

3. *Data Dictionary Manager* - The data dictionary manager allows the user to view the data dictionary information and perform modification operations on the attributes of essential model objects. Only the modification of existing objects is allowed. Creation and deletion of objects must be done via the diagram manager. This manager does not deal with the drawing model objects at all.

4. *Environment Manager* - Allows the user to control the SATool II editing environment via the following operations:

   - *Turn Grid On/Off* - Allows the user to create his diagrams with or without a guiding grid.
   - *Change Drawing Font* - Controls the type of character that is used in the diagrams.
   - *Change Line Thickness* - Controls the thickness of the drawing lines.
   - *Change Object Dimensions* - Controls the size of objects in the diagram.

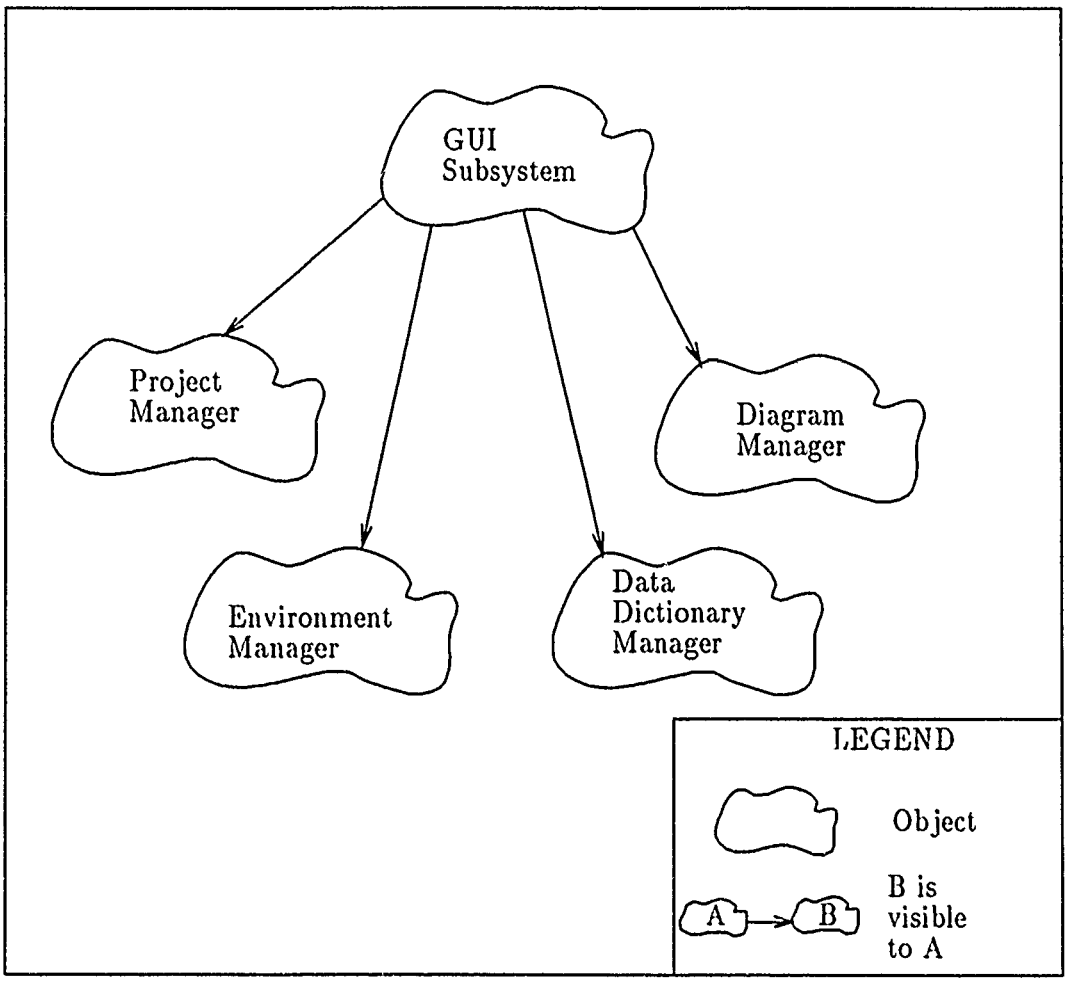


Figure 11. Graphical User Interface Design

4.7 Overall SATool II System Design

The overall SATool II System design is shown in Figure 12. This system is composed of the Graphical User Interface subsystem, the Essential Model subsystem, the Drawing Model subsystem, and the MAGSE subsystem. The MAGSE subsystem works as the interface between the SATool II system and the X-Window System. The GUI performs the calls to the CLIPS Expert system operations to perform the syntax and logical checks on the Essential subsystem objects. This design evolved as a result of the design of the GUI which ties together all the other managers created for the SATool II system.

4.8 Summary

This chapter described the design of all the components of the SATool II system. There are four subsystems that contain managers to handle all the objects found in the system. The essential model subsystem manages all logical objects that contain the data dictionary information associated with activities and data elements within an $IDEF_0$ project. It also manages the expert system used for syntax checking. The drawing model subsystem manages the drawable objects that define where the activities and data elements appear on a diagram within an $IDEF_0$ project. The MAGSE subsystem is used to manage the interface between the X-Window system and the Ada based SATool II system. Finally, the GUI is the part of the SATool II system that allows a user to view an $IDEF_0$ project and check its conformance with $IDEF_0$ rules by tieing together the essential subsystem, the drawing subsystem, and the X-Window System.

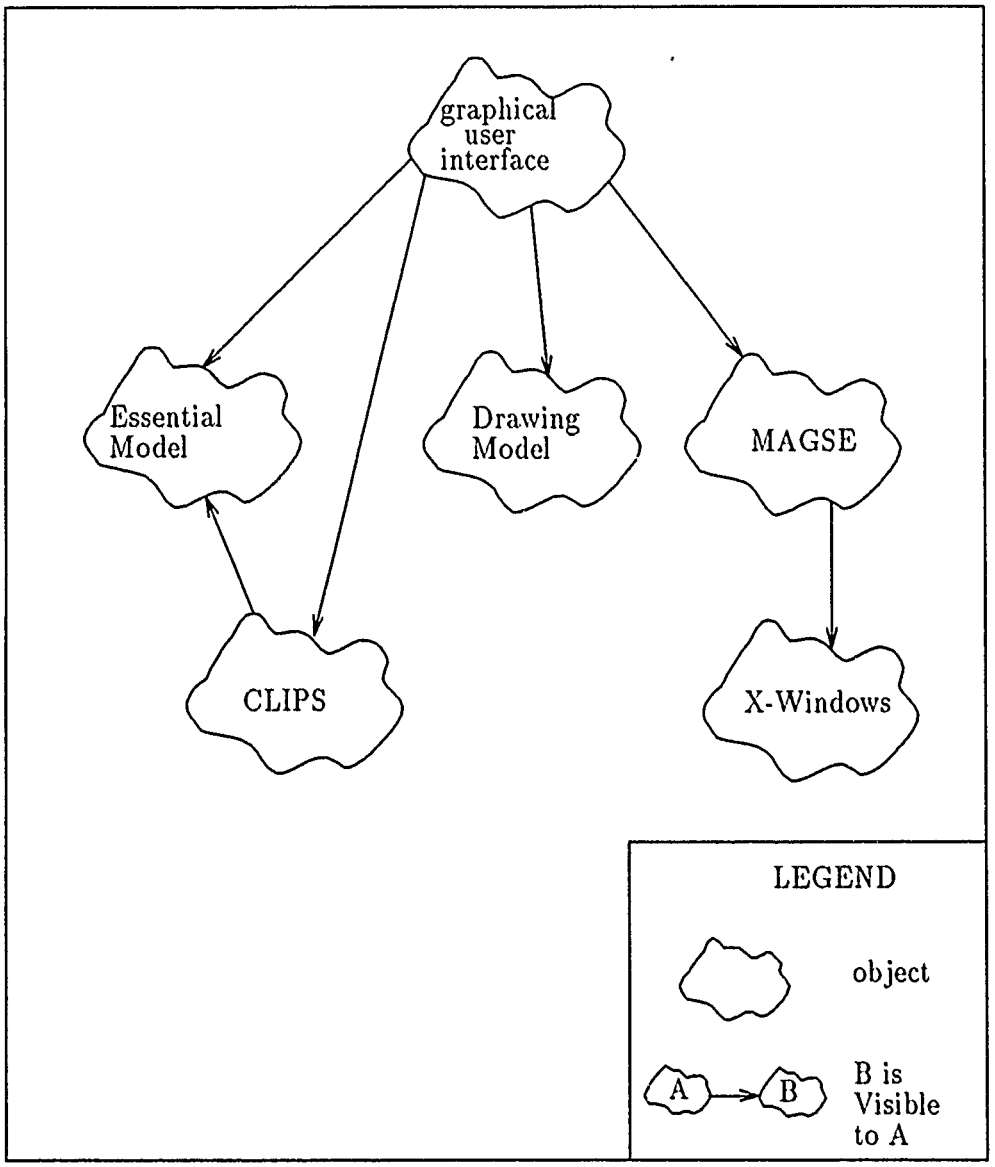


Figure 12. Overall SATool II System Design

V. *Implementation*

5.1 Introduction

This chapter describes the implementation of the Graphical User Interface (GUI) subsystem identified in the previous chapter. The purpose of the GUI is to bind all other subsystems into one homogeneous, user friendly environment. The package dependencies for the Essential, Drawing, and MAGSE subsystems are described in Appendices A through C. The package dependencies for each module in the GUI will be shown via diagrams composed of a modified version of module symbols presented in (3:55-59). Figure 13 shows the three types of module symbols used in this chapter to describe the SATool II system packages. The first module is used to represent the main SATool II subprogram. The second module represents the packages that encapsulate the object operations in the system. The third symbol represents all the packages in an entire subsystem. This third symbol is used as a space saver since there are some packages in the GUI system that require access to all objects and operations from the essential subsystem and the drawing subsystem.

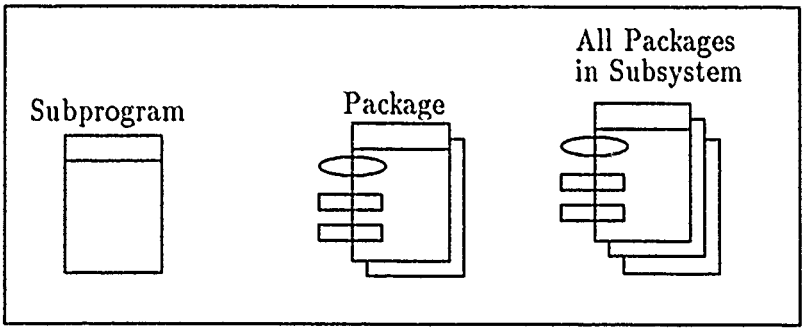


Figure 13. Booch Module Symbols

The following sections describe how the GUI subsystem is logically divided. As described in the previous section, the GUI is composed of the project manager, diagram manager, data dictionary manager, and the environment manager. The

implementation of these managers is viewed from a window perspective. First, the overall SATool II system environment is described, then each of the GUI windows and their function are described along with their package dependencies.

5.2 Overall SATool II System Environment

Figure 14 shows the main SATool II window system presented at the start of program execution. There are six distinct windows shown in this figure. Of these six, there are two windows which hold the menus that implement the four GUI managers. How each manager was implemented is discussed in the sections that present a description of the menu windows.

The first window shown in Figure 14 is the SATool II title window. Below it are the main menu window and the diagram edit menu title window used to identify the tools window. The tools window sits on top of the objects window. Both the main menu window and the tools window encapsulate the operations defined for the four GUI managers. The diagram window is next to the tools and objects windows. There are several other types of windows managed by the GUI that fall into the category of popup windows that do not appear in this figure.

Figure 15 presents the package dependencies of the SATool II application. The SATool II subprogram is used as a control mechanism for the GUI components. The arrows going from the packages to the main subprogram indicate that the subprogram can view the objects contained in each package and can only perform the operations allowed by those packages. The following section describes the window types contained in the GUI.

5.3 Window Types

The GUI is implemented as a subsystem based on window objects. Each type of window has its own set of operations that can be performed on it by any given application. In general, the operations encapsulated by each window package

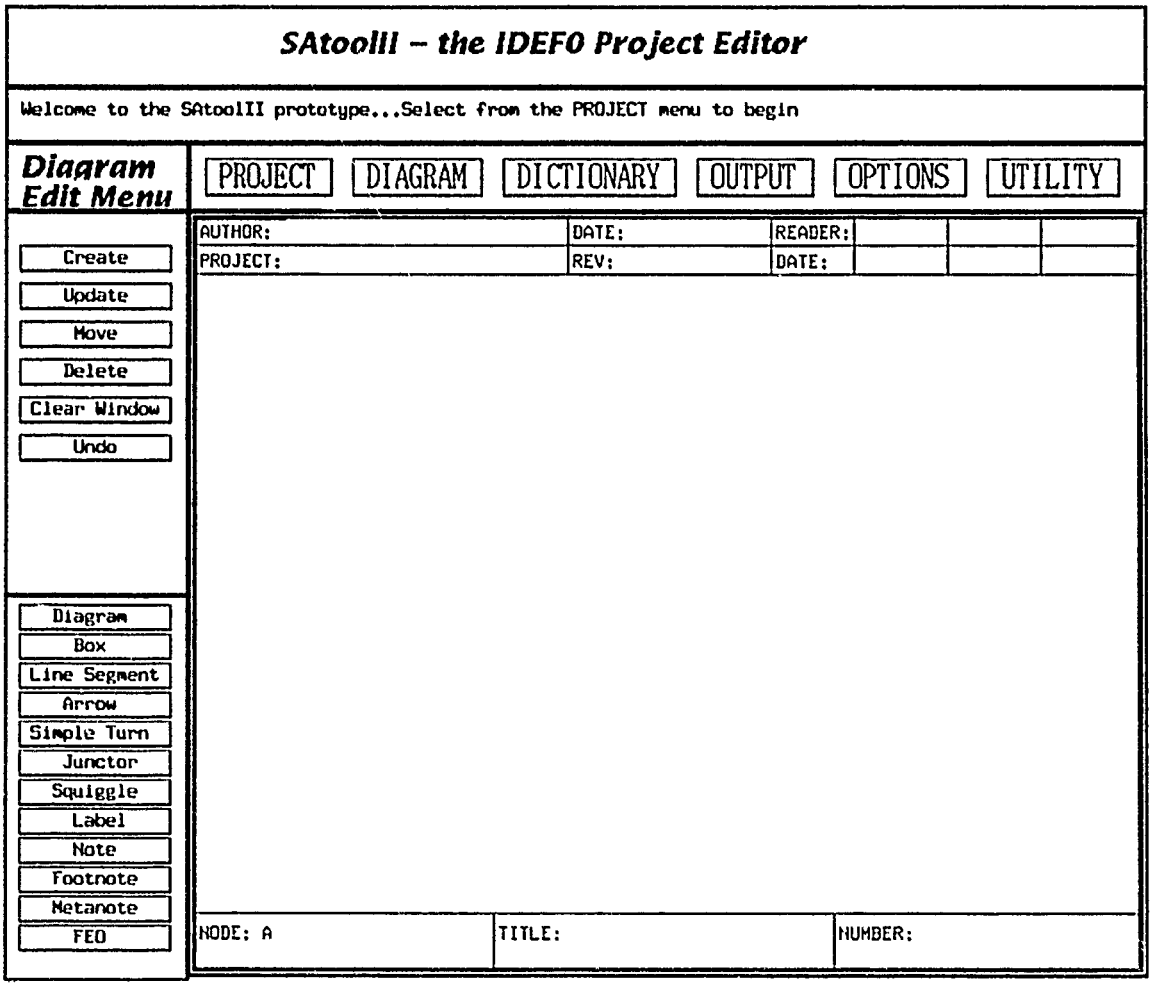


Figure 14. Overall SATool II System View

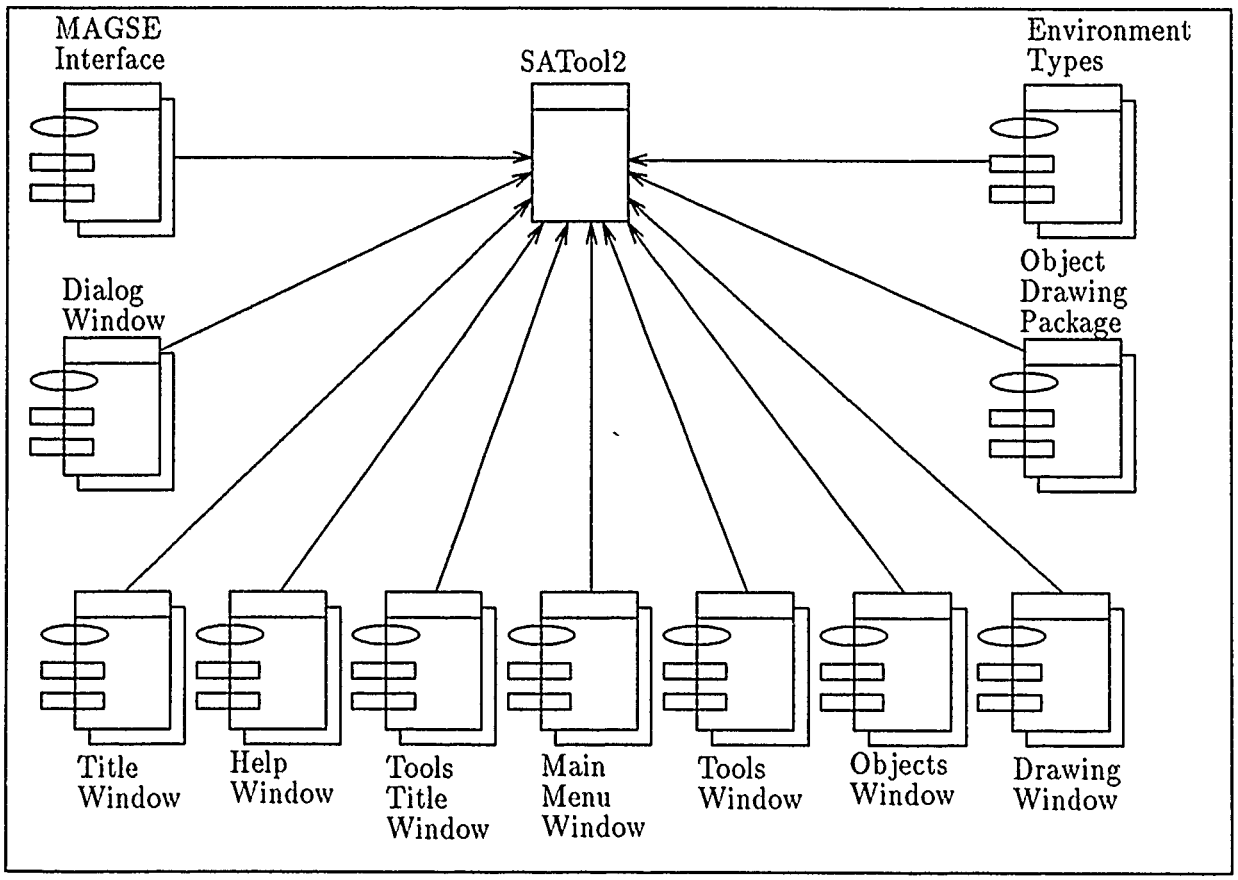


Figure 15. SATool II Subprogram Package Dependencies

are create_window, view_window_contents, and get_window_id. The following list summarizes the various types of windows present in the GUI subsystem:

- *Permanent Windows* - these are windows that once created, are constantly displayed on the screen.
  - *Text Window* - A text window is used to output lines of text to the screen.
  - *Drawing Window* - Allows text and graphical objects to be drawn on it.
  - *Sign Menu Window* - These menus have a set of button-like icons that the user "clicks" on with a mouse.
- *Popup Windows* - These are windows that are created and show up only when needed. Once the operation is completed they are erased from the screen.
  - *Column Menu Window* - This window differs from the sign menu window in that it is usually used as a submenu once an upper menu choice has been selected. Instead of buttons, its choices are displayed in a column format.
  - *Dialogue Window* - This popup window is used by an application to accept a string of text from the screen.
  - *Acknowledge Window* - This popup window is used to convey messages to the screen. These messages can range from error messages to simply acknowledging the completion of an operation.
  - *Confirm Window* - This is a type of dialogue window that accepts one of three answers from a user: yes, no, cancel.

The following sections describe how these window types are used in the GUI subsystem. There is one section for each GUI package described in Figure 15. It can be observed that the MAGSE interface package is accessed by each package in

the GUI. The MAGSE_Interface package is a central player in every package of the GUI subsystem since it is the gateway the system uses to communicate with the X Window environment to output information to the screen about the window objects it contains.

5.4 Dialogue Window

The dialogue window package actually encapsulates three different types of window: dialogue, acknowledge, and confirm. Their operations are limited to create_window and get_user_response. Its package dependencies are listed in Figure 16

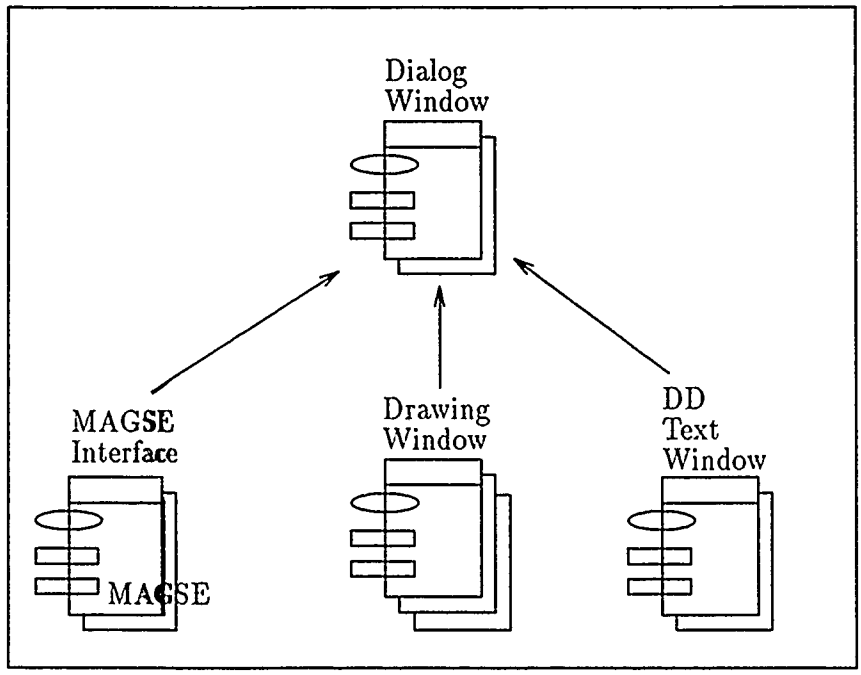


Figure 16. Dialogue Window Package Dependencies

5.5 Title Window

The purpose of the title window is to Identify the SATool II System. It is a text window with no operations associated with it other than the create and view

operations. The only package dependency it has is the MAGSE_Interface package (Figure 17).

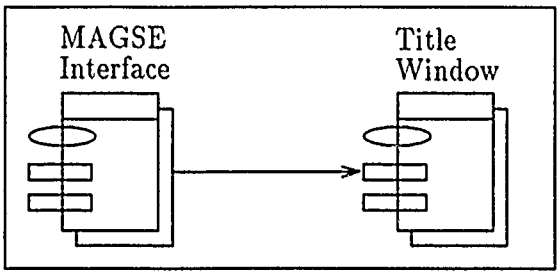


Figure 17. Title Window Package Dependencies

5.6 Drawing Window

The drawing window is used to view the $IDEF_0$ diagrams. The operations associated with it are create_window, clear_window, place_object_on_window, delete_object_from_window. Existing diagrams are output to it via the Diagram button in the main menu and are edited via the diagram editing menus found to its left. Its package dependencies are shown in Figure 18.

5.7 Main Menu Window

The main menu window provides access to the project manager, diagram manager, data dictionary manager and environment manager identified in Figure 11 of Chapter 4. These managers are divided into six sections that can be accessed via the six buttons seen in the main menu window. The package dependencies at this level are seen in Figure 19. The following subsections will provide a description of the operations provided by each button in the main menu.

5.7.1 Project Button The project button encapsulates the project manager functions. Its package dependencies are shown in Figure 20. This button provides ten operations:

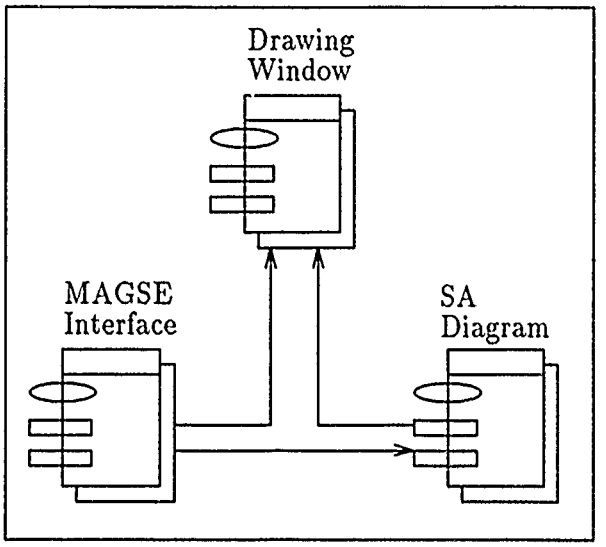


Figure 18. Drawing Window Package Dependencies

1. *Create Project* - Allows a user to create a project from scratch. The SATool II system requires unique project names. If a user tries to create a project using the name of an exiting project, the system will output an error message to the screen and allow the user to try again.

2. *Load Project* - Allows the user to access an existing project.

3. *Save Project As* - Multiple versions of a project can be saved by accessing this operation. A given project is loaded or created. If the user wants to make modifications to the project and save the original project as well as the modified one, he can do so by saving the project under a new name. As with the create operations names here must also be unique.

4. *Save Project* - This operation is used when the project that was loaded or created is to be stored with the name given to it when it was created or loaded. Once the project has been saved, the project is deleted from the SATool II program environment.

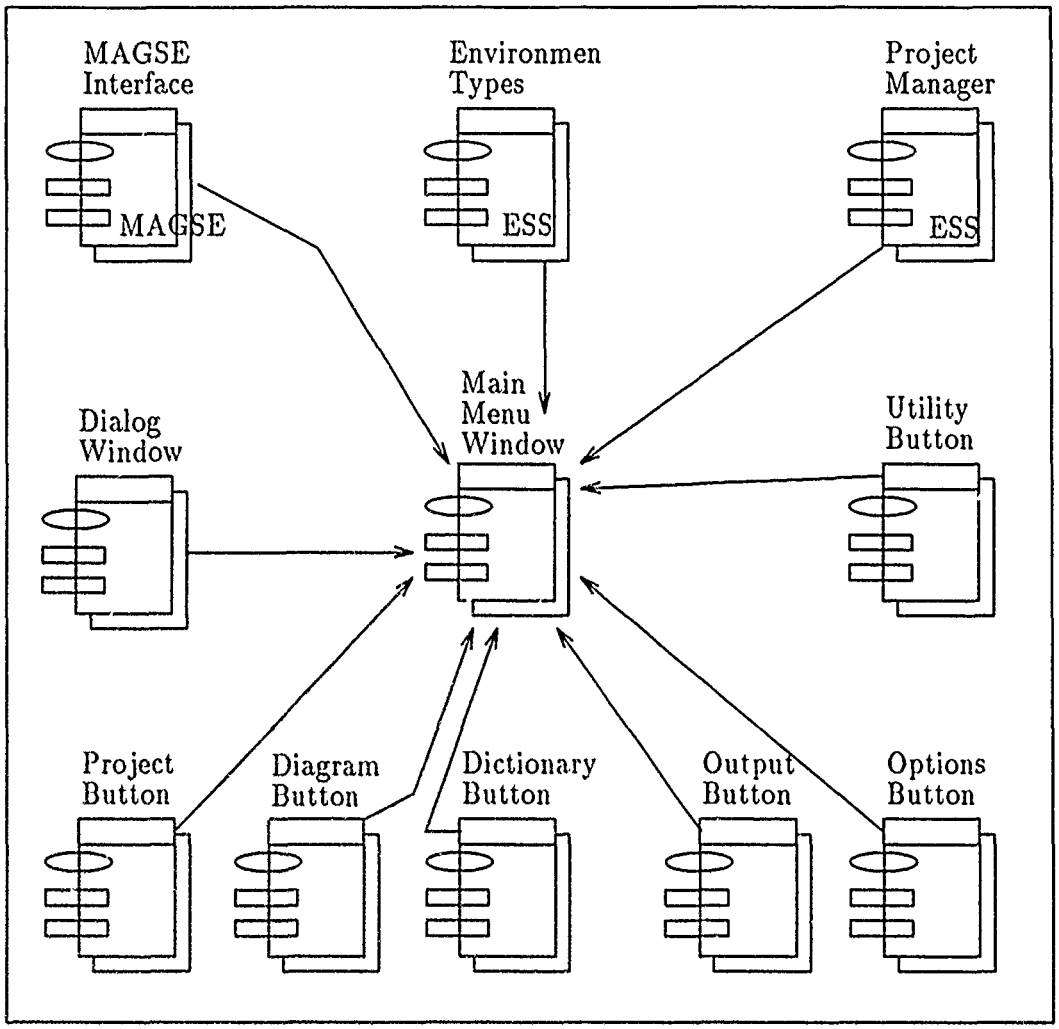


Figure 19. Main Menu Window Package Dependencies

5. *Lay Out Project* - This operation is used to automatically create diagrams, given the essential model objects file exists.

6. *Derive Project* - This operation is used to automatically create the essential model objects, given the drawing model objects file exists.

7. *Show Directory* - Allows the user to see the projects available for editing in the present directory.

8. *Change Directory* - Allows the user to access a different directory so he can view the projects contained in it.

9. *Clear Project Environment* - Deletes the present project from the SATool II environment without saving the changes, if any.

10. *Exit Program* - Allows the user to exit the SATool II program after first checking if the present project has been saved. If it has not been saved, it gives the user the option of saving it under the given name or a new name.

5.7.2 Diagram Button The Diagram manager is divided into three parts. This was done to create a more efficient system for the user. The edit operation was given its own windows and menus. This diagram editor is located left of the drawing window (see Figure 14). The print operation is handled by the Output Button located in the main menu. The Diagram Button was given the view operation. Within it there are three direct viewing options and two diagram traversal options:

1. *Show A-0 Diagram* - Outputs to the diagram window the first diagram in the hierarchy, the A-0 diagram.

2. *Select by Diagram Name* - Outputs to the diagram window the diagram identified by the user. The id used by this operation is the activity name.

3. *Select by Hierarchy* - First, the operation outputs a hierarchical view of the diagrams via a pyramid scheme of lines and boxes. The view can go as far

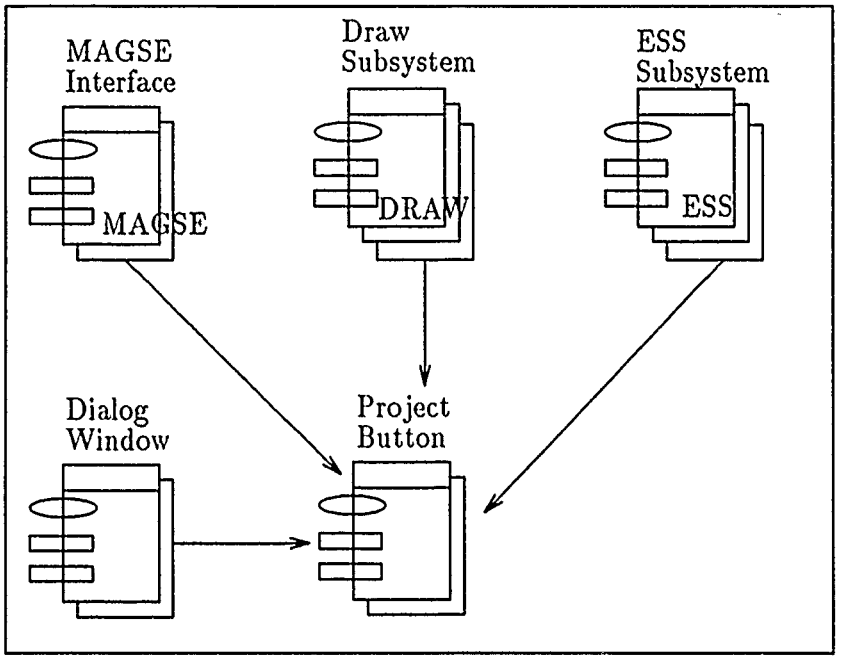


Figure 20. Project Button Package Dependencies

as twenty levels deep. The user is allowed to click on one of the boxes shown in the hierarchy diagram. The chosen diagram is then output to the diagram window.

4. *Go to Child Diagram* - Allows downward traversal of the diagram hierarchy from parent to child, one layer at a time.
5. *Return to Parent* - Allows upward traversal of the diagram hierarchy from child to parent, one layer at a time.
6. *Refresh Diagram* - Refreshes the diagram image in the drawing window.

Once the desired diagram is output to the drawing window, it can be edited using the diagram edit menu windows located at the left hand side of the screen. The diagram button package dependencies are shown in Figure 21.

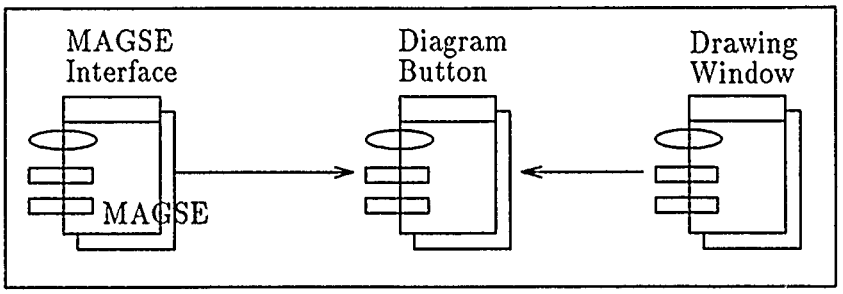


Figure 21. Diagram Button Package Dependencies

5.7.3 Dictionary Button When a user clicks on the dictionary button, a whole new layer of windows is output to the screen. The data dictionary editor operations and package dependencies will be described in a separate section.

5.7.4 Output Button The Output button is used to control the hard copy printouts the user might want to get from his project diagrams. The operations provided by this button allow the user to print out the $IDEF_0$ diagrams as well as the data dictionary entries. The package dependencies are shown in Figure 22. This button has two operations:

1. *Output Data Dictionary* - Outputs all the data dictionary entries into printable text file. The format of the entries is the same as that seen in the Data Dictionary Editor.
2. *Output Diagram* - Provides information to the user on how to output any given diagram using the X Window environment.

5.7.5 Options Button The environment manager is encapsulated by the Options Button. It provides operations to change the present SATool II editing environment. Its package dependencies are shown in Figure 23. It has four operations:

1. *Grid* - Allows user to turn on or off a grid in the drawing window. This grid can be used to aid in the construction of a diagram.

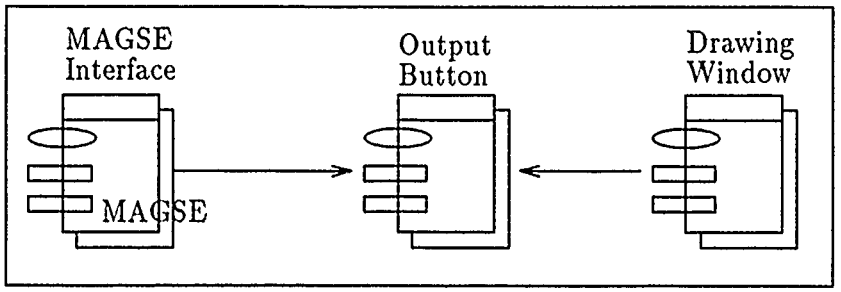


Figure 22. Output Button Package Dependencies

2. *Drawing Font* - Changes the text font for labels and notes appearing in the drawing window.
3. *Line Thickness* - Changes the line thickness of activity boxes and line segments of a diagram.
4. *Dimensions* - Changes the size of objects in the diagram.

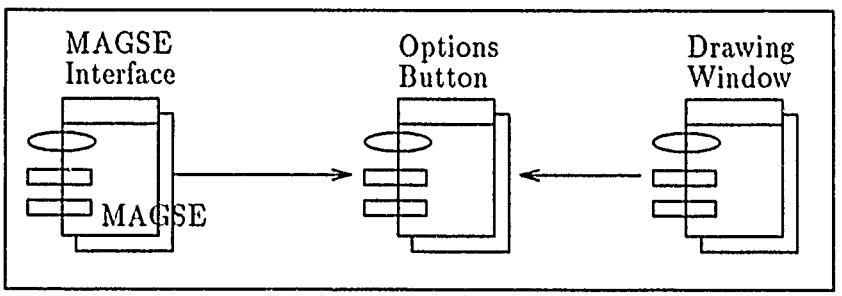


Figure 23. Options Button Package Dependencies

5.7.6 Utility Button The utility button has two operations. The first is *Check Syntax*. This operation allows the user to do a CLIPS syntax check on the essential model objects in the project. Presently, there is no equivalent check for drawing model objects. The second operation provided by the utility button is *Refresh Screen*. It can be used to redraw all the windows and their text. The package dependencies for this button are shown in Figure 24.

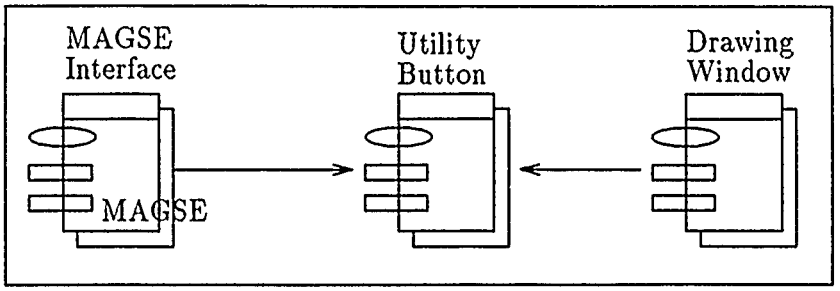


Figure 24. Utility Button Package Dependencies

5.8 Tools Title Window

This window is used to identify the diagram editing windows that sit below it. It has no operations associated with it. Its package dependencies are shown in Figure 25.

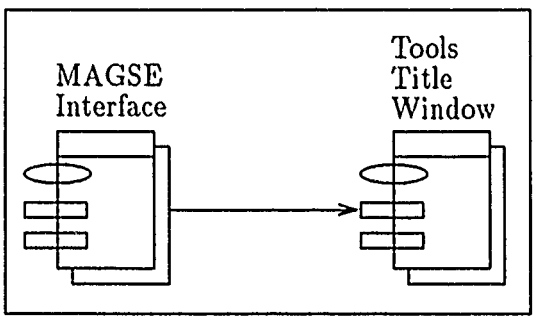


Figure 25. Tools Title Window Package Dependencies

5.9 Tools and Objects Windows

The Tools and Objects windows are directly related to each other. The user can access these menus directly so he can create $IDEF_0$ diagrams or edit existing ones. These menus were placed in permanent sign menu windows instead of the popup column menu windows to provide easier access to the user. The way these two windows work together is by allowing the user to click on one of the tools window buttons. If the create, update, move, or delete buttons are selected, the system

then waits for the user to click on one of the object buttons. The tools window and objects window package dependencies are shown in Figure 26 and Figure 27 respectively. Following is a discussion of the operations provided by each button in the tools menu.

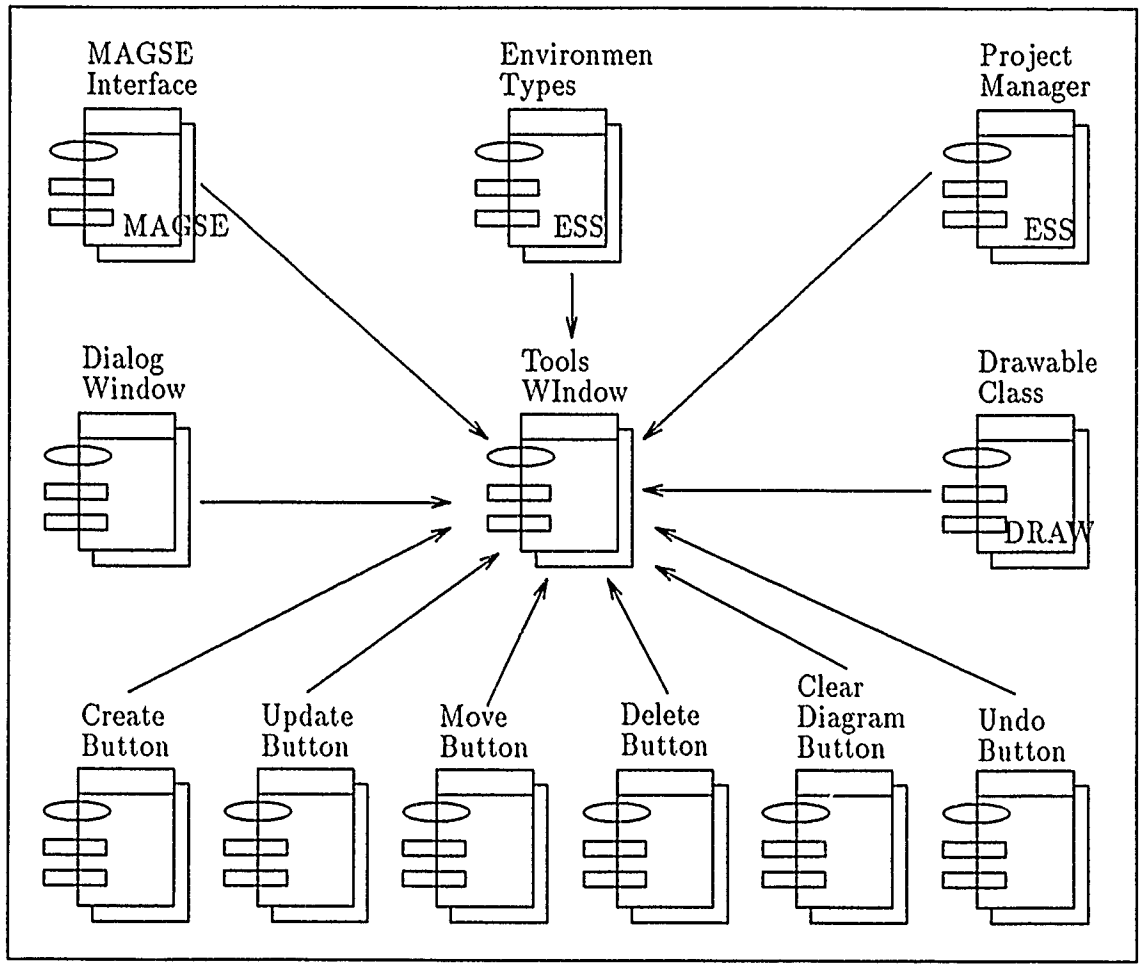


Figure 26. Tools Window Package Dependencies

5.9.1 Create Button The create button allows the user to create an A-0 diagram when there is a new project in the system. If the project already has one or more diagrams, the user must first bring up one of those diagrams before any creation or editing can be done. This ensures that the diagram system is hierarchically correct. Once a diagram has been created, it is displayed in the drawing window and

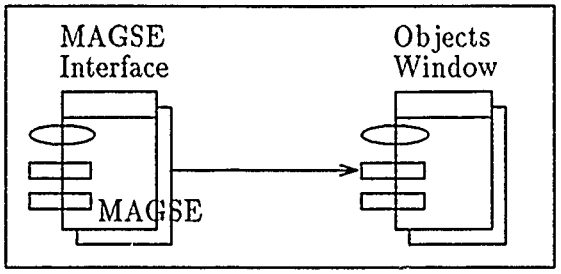


Figure 27. Objects Window Package Dependencies

the user can begin to place activities and data elements in it. Subsequent diagrams can be created by first clicking on the *Create* button, then the *Diagram* button, and then clicking on an activity box in the diagram. This creates the diagram associated with the given activity box. If that box already has a diagram associated with it, an error message is issued. The package dependencies are shown in Figure 28.

5.9.2 Update Button The update button allows updates of data element names, activity names and numbers, label text, and note text (includes footnotes and metanotes). Package dependencies are outlined in Figure 29.

5.9.3 Move Button This button allows an object to be moved from one place to another in the diagram. It ensures that illogical placements do not occur. For example, placing an arrow-head inside an activity box. The package dependencies for this button are shown in Figure 30.

5.9.4 Delete Button Objects are deleted by first clicking on the *Delete* button, then an object button, and then clicking on the target object in the diagram. All traces of that object then erased from the essential subsystem and the drawing subsystem. Package dependencies for this button are shown in Figure 31.

5.9.5 Clear Diagram Button Clicking on this button will cause the present diagram to be erased from the drawing window. There is no need to call a save

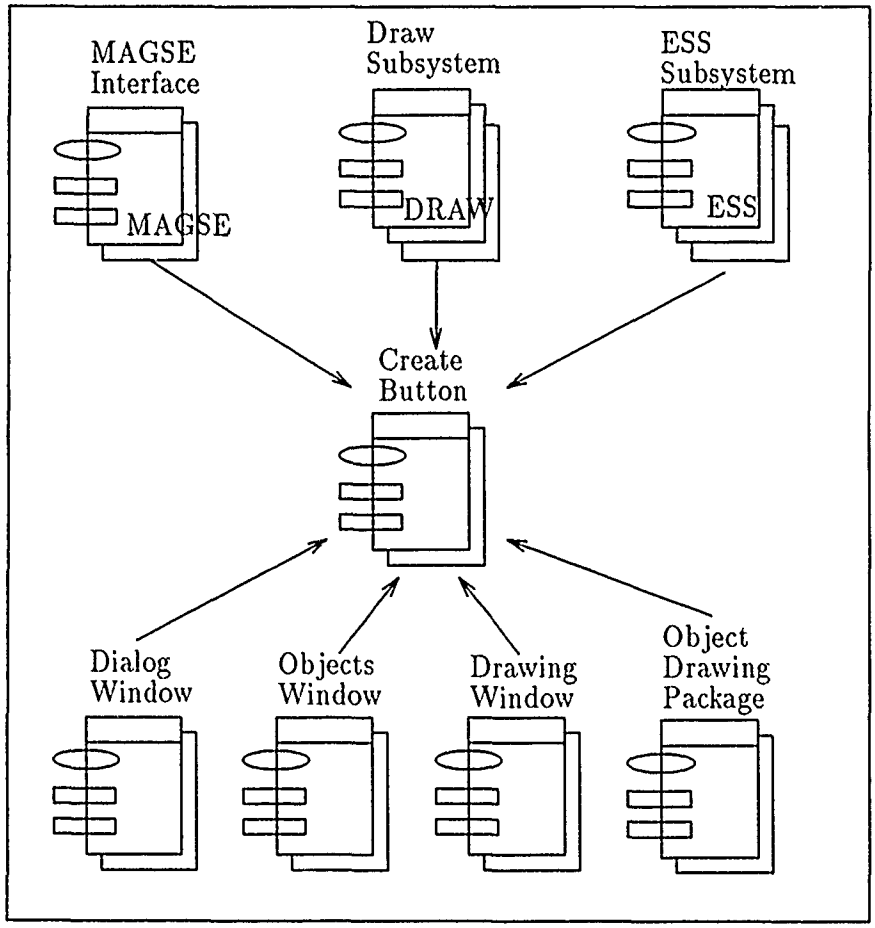


Figure 28. Create Button Package Dependencies

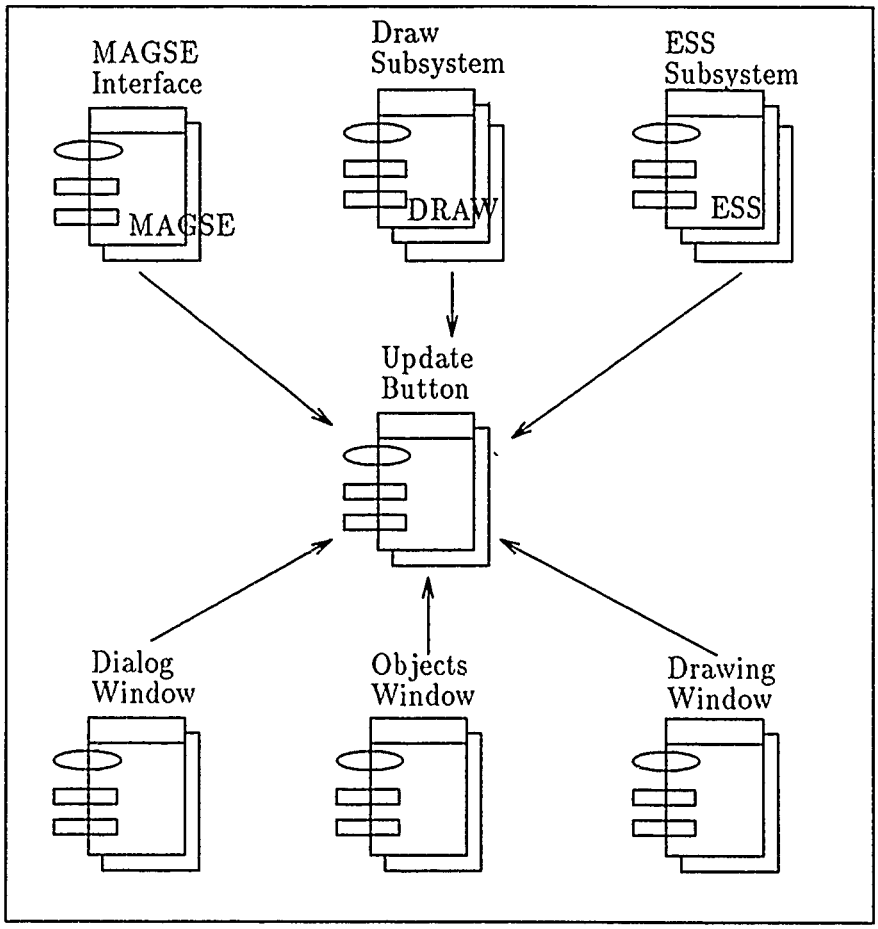


Figure 29. Update Button Package Dependencies

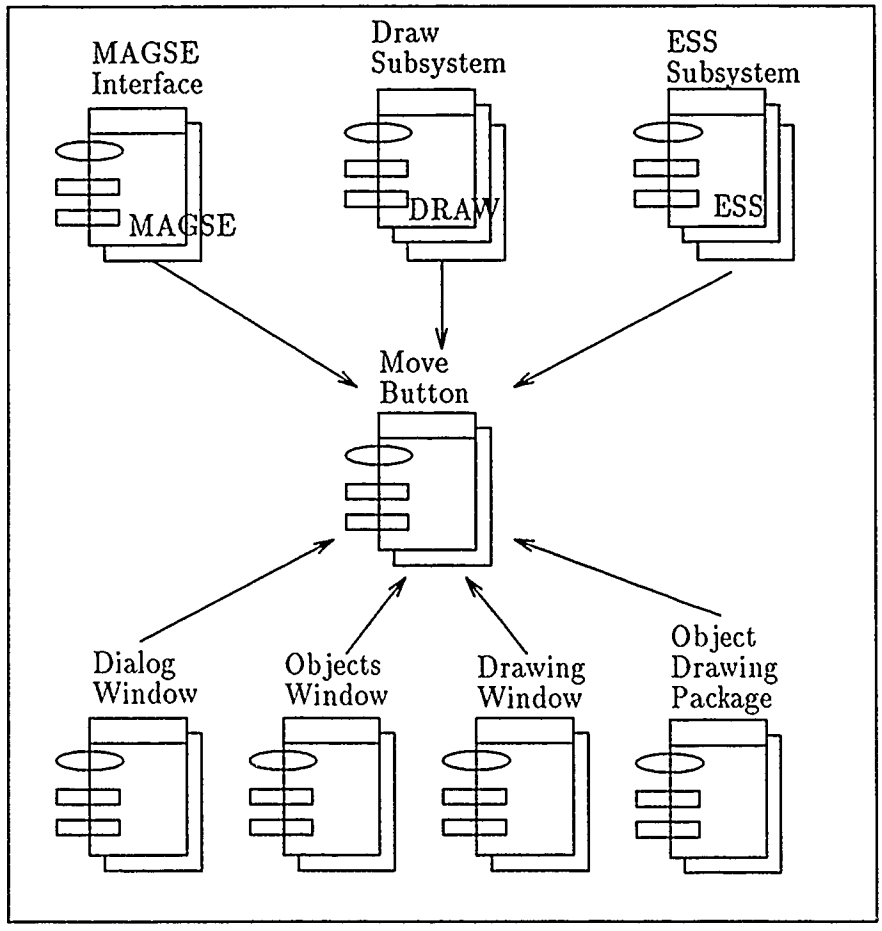


Figure 30. Move Button Package Dependencies

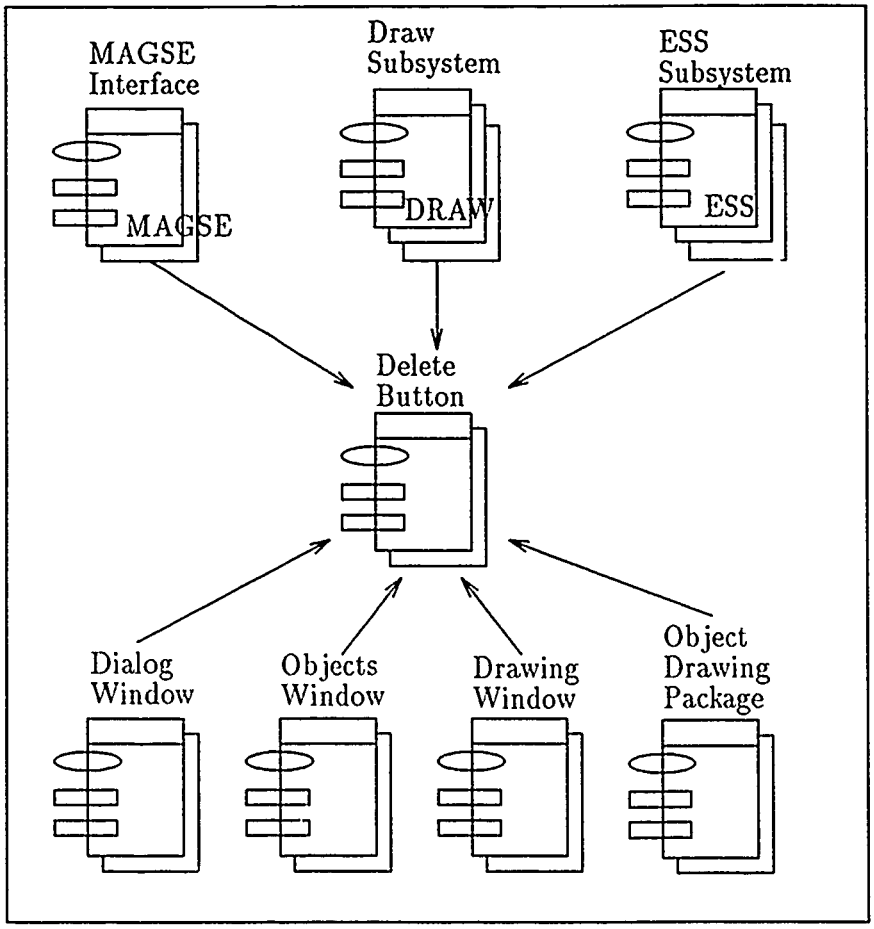


Figure 31. Delete Button Package Dependencies

operation because all creations and modifications are saved into the respective subsystems when they are made. Deleting the diagram from the screen will cause the system to wait for a diagram to be called up via the *Diagram button* in the main menu window before any more operations can be performed with the diagram edit menu. Figure 32 shows the package dependencies for this button.

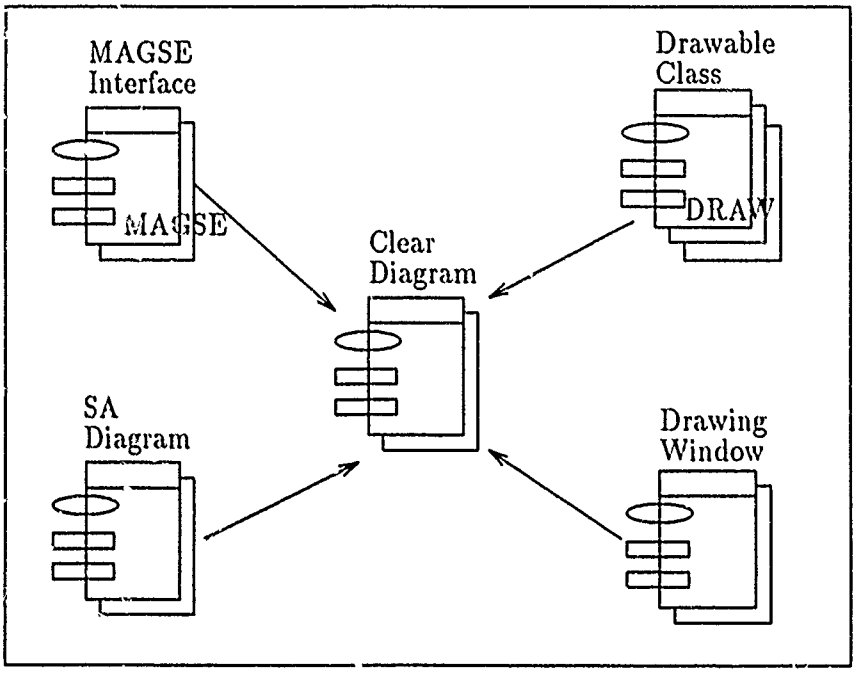


Figure 32. Clear Diagram Package Dependencies

5.9 6 Undo Button This button lets the user undo the last command. If a creation was made, then the created object is deleted. If an object was moved, it is placed back into its original location, and so on. The package dependencies are shown in Figure 33

5.10 Data Dictionary Editor

When the *Dictionary* button is pressed in the main menu window, the user is given four choices He can view the data dictionary entries for all activities, all data

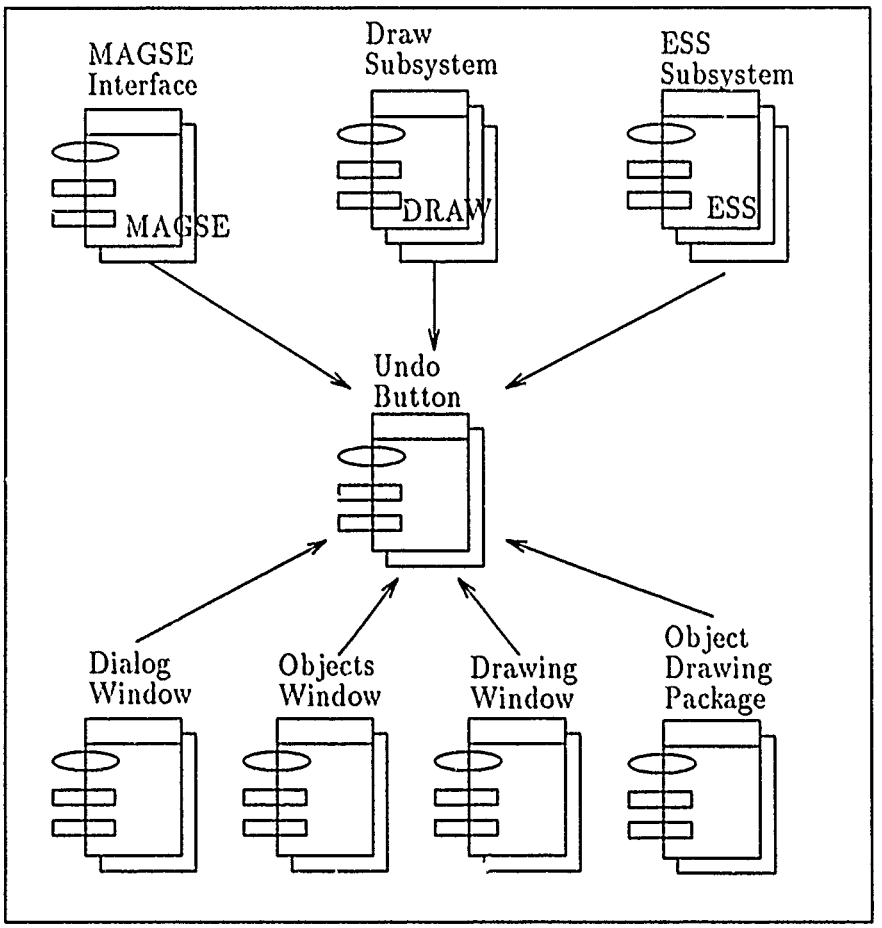


Figure 33. Undo Button Package Dependencies

elements, a single activity, or a single data element. Depending on which choice is made, the system will then bring up a new set of windows.

There are separate edit screens for the activity and data element entries (see Figure 34 and Figure 35). In general, each screen has four windows: a title window, an edit window, an options window, and a text window. The title window is used to identify the Data Dictionary editor. The text window shows the data dictionary entry information for each activity or data element, one at a time. The edit window allows changes to be made to the text via its edit buttons. The entry is then automatically updated in the screen. The options menu allows the user to traverse through all the data dictionary entries via the *Next Activity* or *Next Data Element* button. Each entry may be several pages long and can be viewed page by page with the em Next Page button. This window system can be exited via the *Exit* button. The data dictionary package dependencies are outlined in Figure 36.

5.11 Summary

This chapter presented the implementation of the Graphical User Interface subsystem. In the previous chapter this subsystem was identified as having four general object managers. In this chapter it was shown how these four managers were implemented within the SATool II system via the different menus and windows provided by the application. The GUI is divided into a set of window objects and the SATool II subprogram is used as a means of control for these windows and the operations that are defined for each.

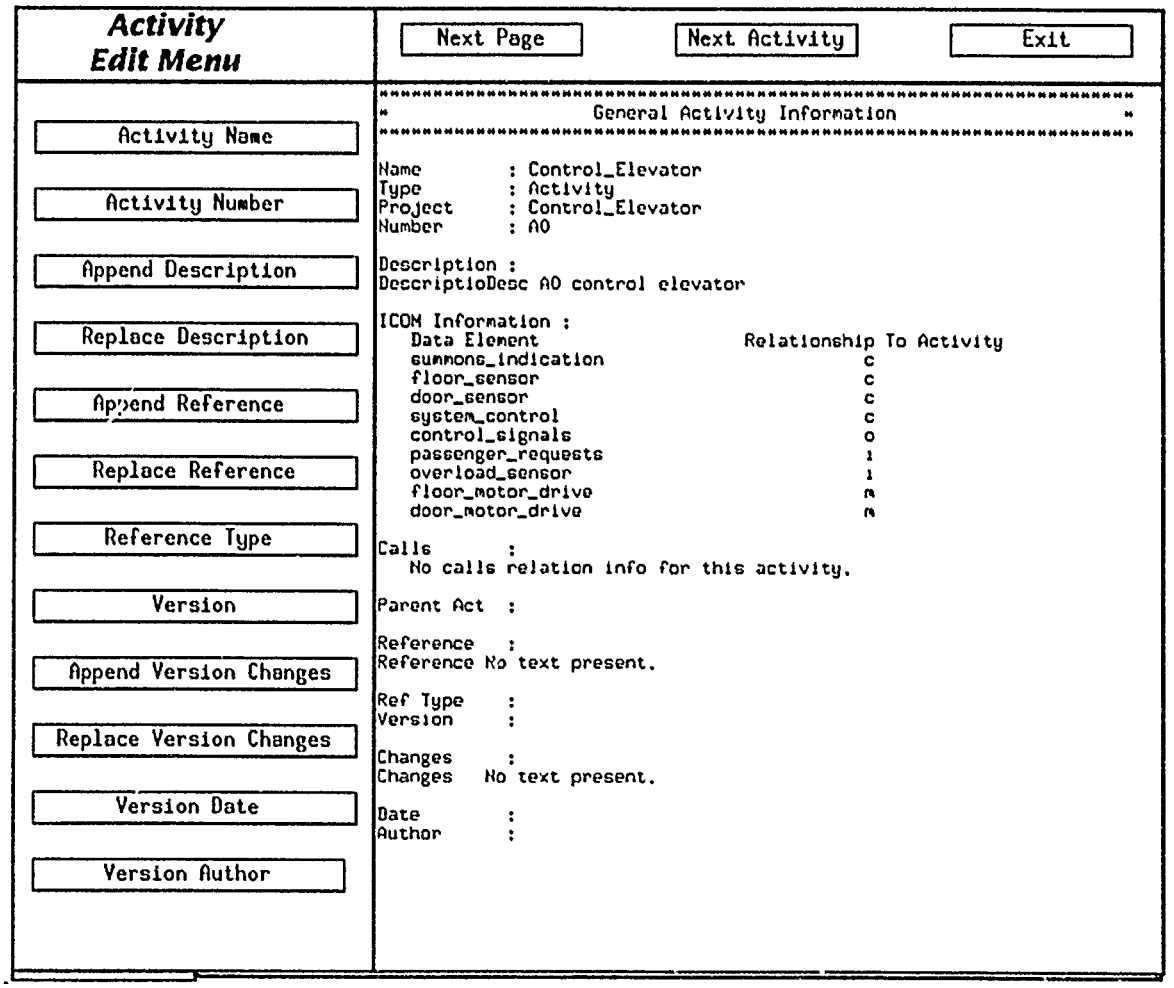


Figure 34. Activity Data Dictionary Editor Screen

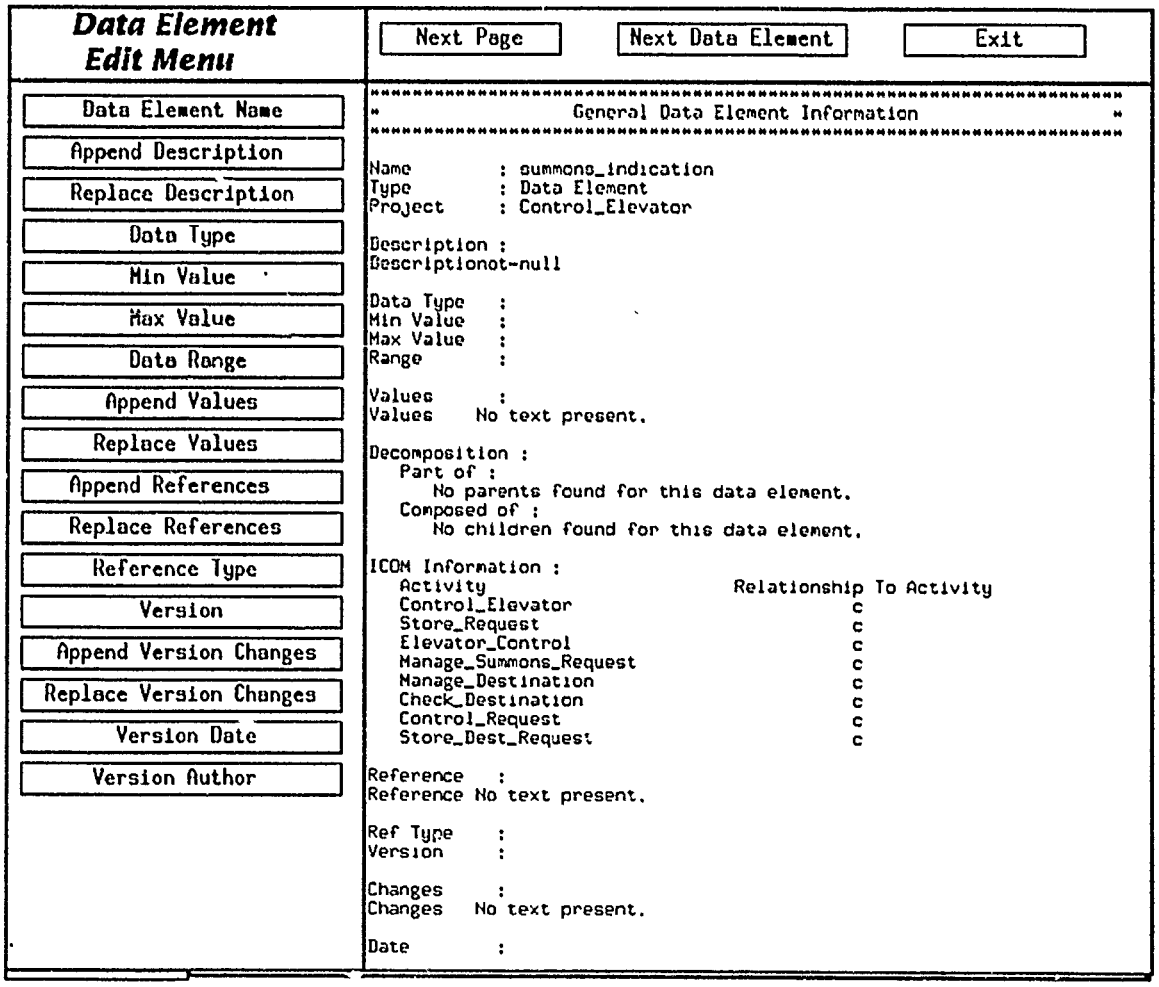


Figure 35. Data Element Data Dictionary Editor Screen

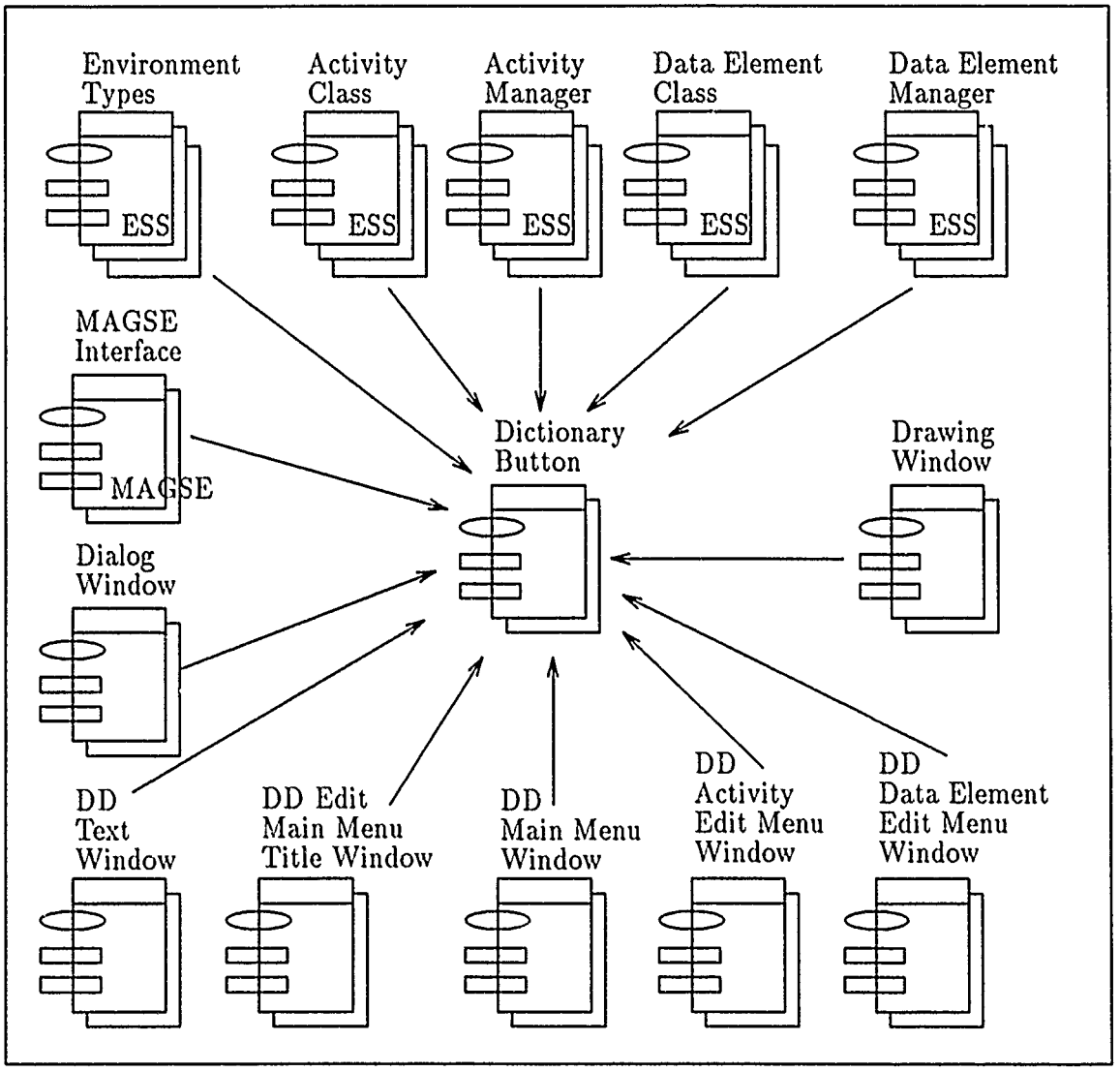


Figure 36. Data Dictionary Button Package Dependencies

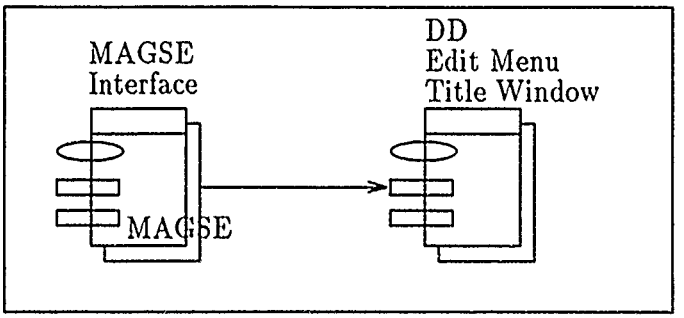


Figure 37. Data Dictionary Main Menu Title Window Package Dependencies

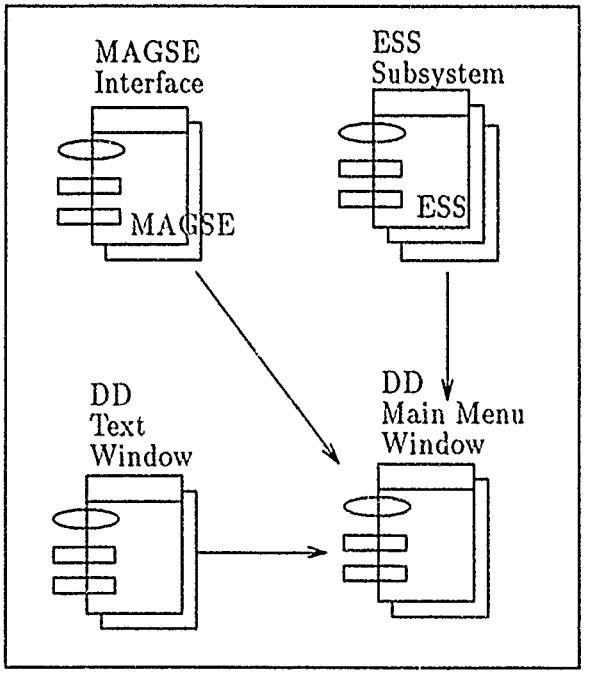


Figure 38. Data Dictionary Main Menu Window Package Dependencies

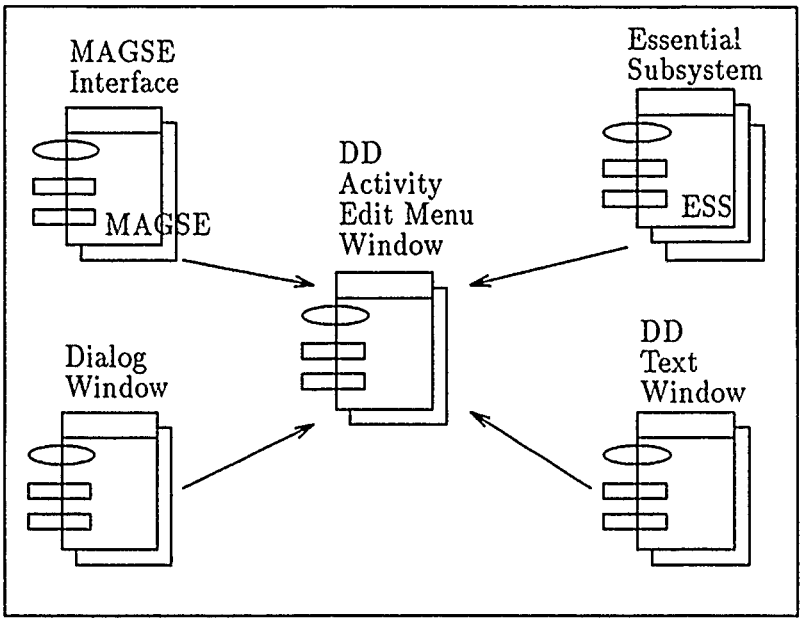


Figure 39. Data Dictionary Activity Edit Menu Window Package Dependencies

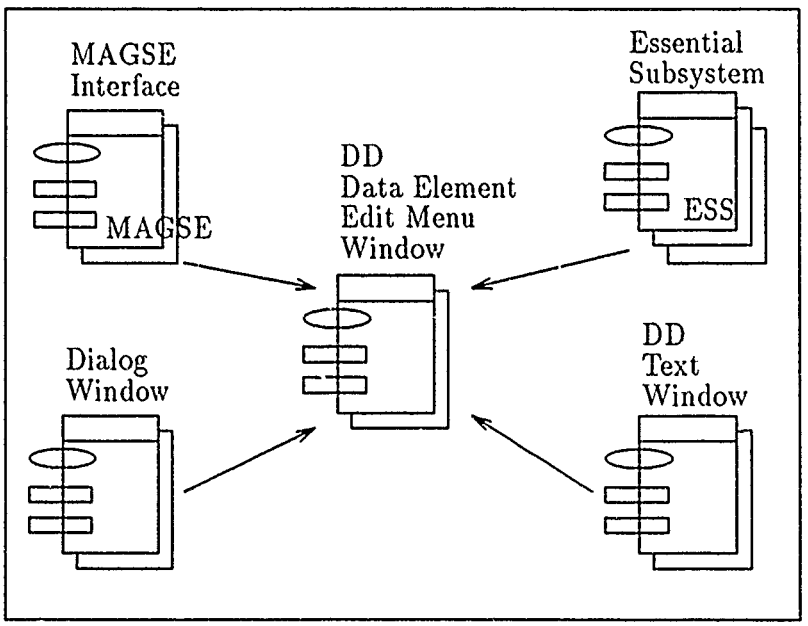


Figure 40. Data Dictionary Data Element Edit Menu Window Package Dependencies

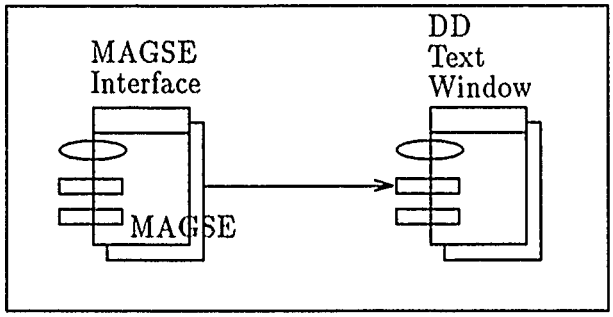


Figure 41. Data Dictionary Text Window Package Dependencies

VI. Testing and Evaluation

6.1 Introduction

This chapter presents the testing procedures, test results, and evaluation of the SATool II system. "Software testing is defined as the execution of a program to find its faults" (13:191). Myers states that "a good test case is one that has a high probability of detecting a previously undiscovered defect, not one that shows that the program works correctly" (20). The purpose of the tests performed on the SATool II system was to determine how many faults could be found. A test was successful if it found an anomaly in the system. It failed if it did not detect an error. The following section describes the tests performed on SATool II. The results of these tests are then discussed. Finally, an evaluation is made of the system as a whole based on the test results.

6.2 Testing

There are seven types of tests that can be performed on a software system (13:192-204). The following subsections discuss the tests performed on the SATool II system based on these seven types of tests.

6.2.1 Unit Tests Also known as white box testing because the test is based on knowledge of the internal design of the module. It is used to validate single programs or modules. These are essentially path tests and are typically conducted in isolated or special test environments. Each module in the SATool II system was tested using this white box approach. This included all the operations performed by the main menu buttons and the operations performed by the diagram editing buttons. General test cases were defined for each module based on the possible paths the module could follow (see Appendix F).

6.2.2 External Function Tests Validate the external system functions, as stated in the external specifications. This is also known as black box testing because the test performed has no knowledge of the internal design of the modules being tested. This type of test was used in conjunction with the integration test described below.

6.2.3 Integration Tests Validate the interfaces between system parts (modules, components, subsystems). It can be performed in one of three ways:

- *Bottom-Up* - Each module is tested separately using special development drivers that provide the needed system functions. As more modules are added to the system, the driver is replaced by the modules that perform the simulated functions.
- *Top-Down* - Uses a prototyping approach. A basic system skeleton is constructed and new modules are added and tested as they are developed. The function of lower level modules are simulated by program stubs.
- *Big Bang* - Each module is developed first. Then, the are all assembled and run together. This is the least effective of the three methods. The need for special drivers or module stubs is eliminated; but, each module is only given a cursory test with this method and the likelihood of a total system integration failure is great.

Integration testing for SATool II was performed using the top-down approach. The main subprogram was developed and implemented with stubs for each major function. As each module was developed, it was attached to the main system and tested to ensure proper integration.

6.2.4 System Tests Validate the system to its initial objectives. The robustness of the system as a whole is considered during these tests. They take into

consideration factors like peak loads and volume the system can accept, security, performance under peak and normal conditions, system reliability, and recovery mechanisms. The SATool II code was designed to be internally robust. It tries to take into consideration all possible inputs to the system. When a failure is detected anyway, a secondary mechanism takes control where the program terminates as gracefully as possible. So, the system testing for SATool II was actually embedded into the unit and integration testing cases.

6.2.5 Acceptance Tests Validate the system or program to the user's requirements. This test was done in conjunction with the installation test described below.

6.2.6 Installation Tests Validate the instability and operability of the user's system. In other words, test the system in a real user's environment. This test was performed by getting several volunteers to use the system and fill out questionnaires pertaining their evaluation of the system based on a set of criteria. Appendix G contains a sample of the standard form used by the Department of Electrical and Computer Engineering at AFIT to evaluate software systems.

6.2.7 Regression Tests Run a subset of previously executed integration and function tests to ensure that program changes have not degraded the system. This type of test is usually performed once a system has been developed and is operational. Therefore, this type test was not used at this point for SATool II. However, if future regression testing should be required, Appendix F contains the test cases developed for this research effort.

6.3 Test Results and Evaluation

6.3.1 MAGSE Interface Throughout the development of the SATool II application several errors were detected in the MAGSE Interface procedures. These errors have been corrected and documented in the code of the following procedures:

- MAGSE_Interface.Input_Device.Get_Confirm_Choice
- MAGSE_Interface.Input_Device.Get_Dialogue_Response
- MAGSE_Interface.Input_Device.Get_Menu_Entry Choice
- MAGSE_Interface.Input_Device.Wait_For_Acknowledgement

The only observed discrepancy that has not been corrected in the MAGSE subsystem is related to the exposure of window text. If a window is hidden or covered by another window in the SATool II application and then shown again, the contents of the window are not re-exposed as well. Since this is not a problem that occurs often, a temporary solution has been created by the addition of a *Refresh Screen* operation to the Utility Button menu.

6.3.2 Graphical User Interface The unit test cases presented in Appendix F were used to validate the proper function of each operation the SATool II performs. These test cases were useful in uncovering several logical errors. The errors encountered have been identified and are currently being corrected.

The test cases for the integration test consist of a set of $IDEF_0$ project diagrams for a project named ARTMOS (see Appendix F). The purpose of re-creating this set of diagrams within the SATool II system is to ensures that the system can accurately create the essential and drawing model information for a project that containes a comprehensive set of diagrams. This test should be performed on the system once the unit testing errors have all been corrected.

The installation test should be performed by obtaining user evaluations with the CAD-Tool Human-Computer Interface Evaluation form (see Appendix G). This form can be used to measure the user satisfaction with the tool's graphical user interface. This test should be performed once the system has been validated at the integration level.

6.4 Summary

This chapter presented the test procedures used to evaluated the SATool II system. Comprehensive unit tests were performed during the system development and were helpful in uncovering several logic errors. Corrections are currently being made to the SATool II code based on the unit test results. The integration test and installation test should be performed once the code has been validated by the unit test cases.

VII. Conclusions and Recommendations

7.1 Summary

This thesis was divided into seven chapters. The first chapter presented the background information that triggered this research effort. Chapter 2 presented the findings of the literature review performed in order to explore system enhancement alternatives for the SATool II system. Chapters 3, 4, and 5 presented the Requirements Analysis, Design and Implementation of the SATool II system. The SATool II tests, results and evaluation were presented in Chapter 6. This chapter summarizes the thesis findings and presents several recommendations for further work to be done with the SATool II system project.

7.2 Conclusions

7.2.1 Research Accomplishments This investigation resulted in several accomplishments in relation to the design and implementation of the Graphical User Interface for the SATool II system:

- The revision of the drawing model for the SATool II system. The revision redefined the relationships between the objects in the model and the attributes for each object. This revision made a simpler implementation of the GUI possible.
- The design and implementation of the revised drawing model.
- Development of a layout algorithm for $IDEF_0$ diagrams.
- An object oriented design and implementation of the GUI was completed. The system was implemented so that future enhancements to the system can be easily added.

- Demonstration that the essential model implementation and the revised drawing model implementation have been fully integrated into the SATool II system. This ensures that future revisions of the models are not necessary to get the system to operate correctly.

7.2.2 SATool II The integration of the essential, drawing, and MAGSE subsystems into the GUI was a much larger task than seemed on the surface. Working with faulty code found in the MAGSE subsystem only made the job more difficult. With all the revisions and stumbling blocks encountered, the project was still accomplished. All the major operations required for proper system operation were implemented. The system is now at a point where all component subsystems have been fully integrated. Major revisions of the models are not necessary or desired at this point since the system has been implemented using the models as they currently stand. However there are a few minor modifications recommended for the drawing and essential models. These are discussed in the next section.

7.3 Recommendations

7.3.1 SATool II Menu Selections There are still some menu options that have not been implemented. Future revisions of the system should include the addition of the operations defined for these menu options. These operations include:

- Main Menu Window
  - Project Button
    - Lay Out Project
    - Derive Project
    - Show Directory
    - Change Directory
  - Diagram Button

        * Show by Hierarchy

    - Options Button

        * Grid

        * Drawing Font

        * Line Thickness

        * Dimensions

- Tools Menu Window

    - Move Button

        * Box

        * Line_Segment

        * Simple_Turn

        * Junctor

        * Arrow

        * Squiggle

    - Undo Button

7.3.2 System Enhancements Chapter 2 outlined several enhancement options that could be added to the SATool II system in the future. These options included highlighting via color, an online help function, an automatic diagram layout function that creates drawings from an essential model description, and a configuration control function for the projects managed in the SATool II system.

7.3.3 The MAGSE Subsystem The MAGSE subsystem is connected to the X Window System via SAIC's version of Ada bindings to X Lib (14). A good enhancement option for this subsystem would be to replace the X Lib/Ada binding code with X Window's Motif which performs a function similar to that of the MAGSE

subsystem. The MAGSE operations should remain tied to the SATool II system. Only the internal structure of the MAGSE should be changed. This will avoid a major code rewrite to the system.

7.3.4 The Essential and Drawing Models The Tools Menu of the SATool II system provides a button for updating textual information maintained by the diagram header like C-Number, Readers, and Reader Dates. This textual information presents an incongruency for the SATool II system in the sense that all the other textual information found in the diagram header is updated via the data dictionary editor. If these particular diagram attributes are not necessary for the essential model, why have them at all? If they are an essential part of the diagram, then they should be added as attributes of the activity objects that represent each diagram.

Another incongruency found in the implementation of the SATool II system is the fact that the label object can be associated with either a squiggle or a data element. This dualism causes the label object to maintain the same information (data element name) in two separate fields when it is associated with a data element and have an empty field (Related Data Element) when it is associated with a squiggle. A better arrangement would be to have two separate types of labels, one associated only with data elements and the other with squiggles.

Appendix A. *Essential Subsystem Implementation Packages*

The essential subsystem packages and the CLIPS system packages are presented in Figure 42, Figure 43, and Figure 44. These figures outline the relationship between the object managers of the essential subsystem and SATool II. A full description of these packages can be found in (17) and (24). The Environment_Types package shown in Figure 42 contains the Generic Multiple Object Manager all the object managers in the essential subsystems are based on. Each object manager is in turn accessed as a separate entity by the SATool II system. This package also contains global variables accessed by all managers.

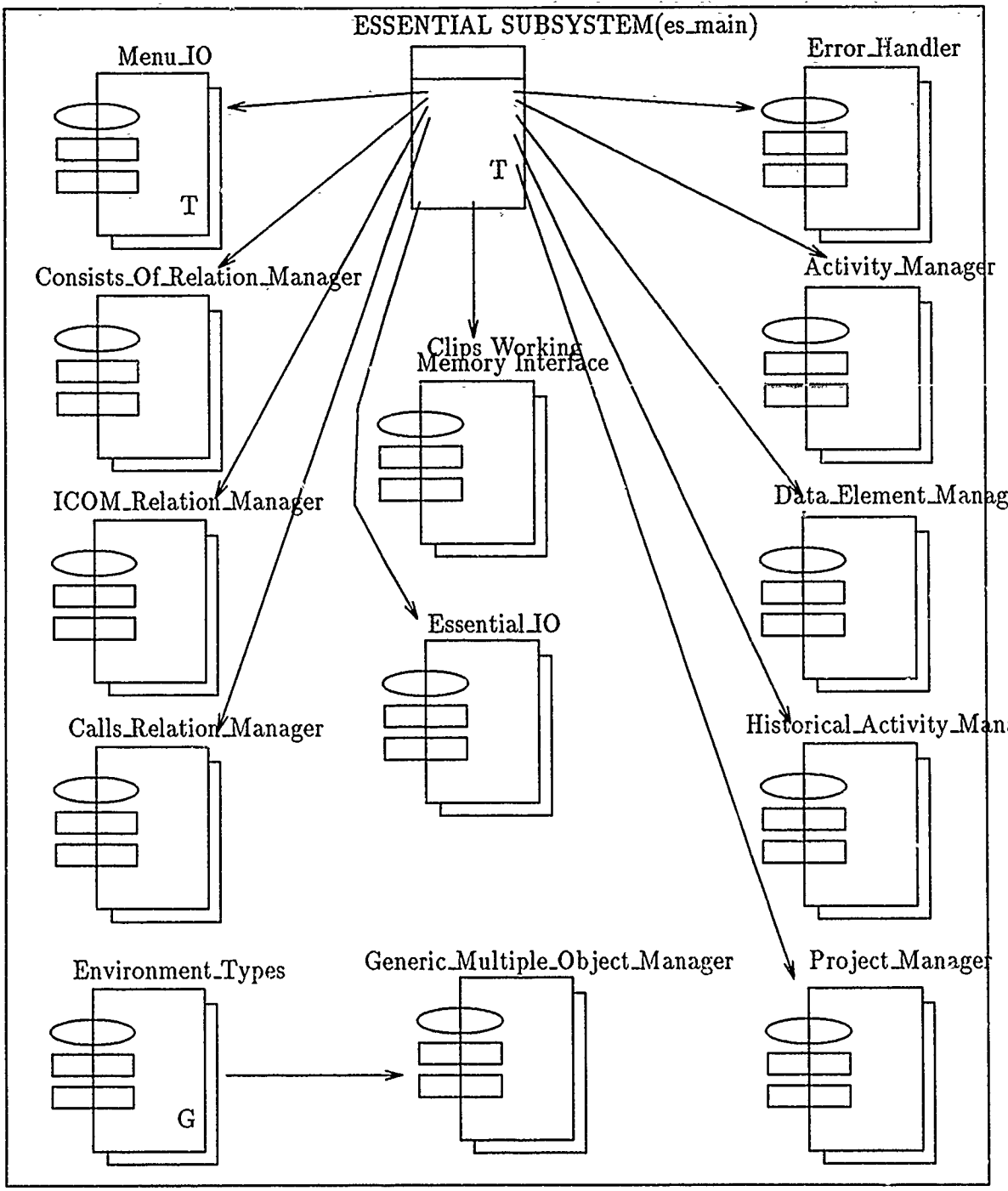


Figure 42. Essential Subsystem Package Dependencies

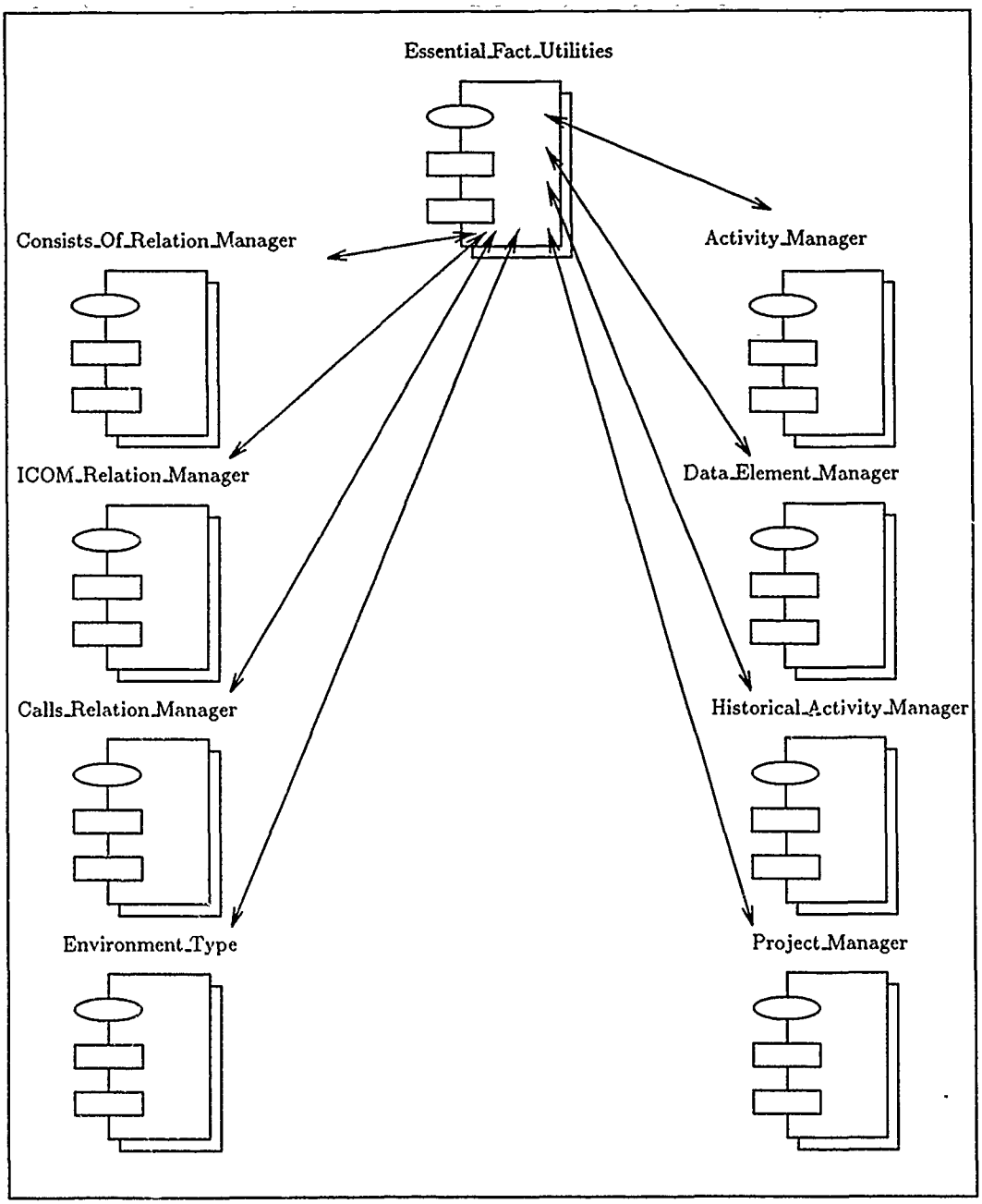


Figure 43. Essential Fact Utilities Package Dependencies

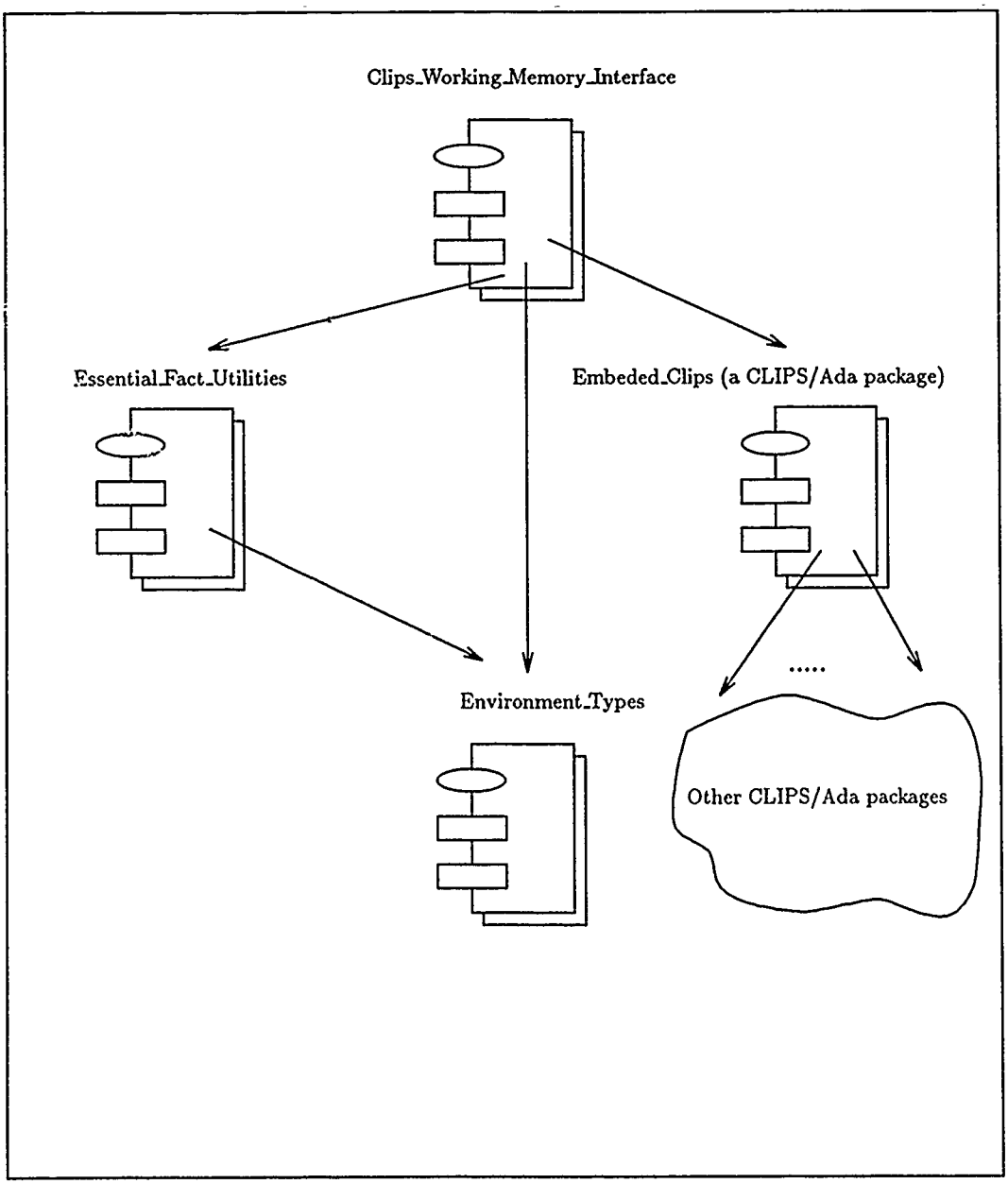


Figure 44. CLIPS System Package Dependencies

Appendix B. *Drawing Subsystem Implementation Packages*

The drawing subsystem packages are presented in Figure 45. This figure shows the relationship between the object managers of the drawing subsystem and SATool II. A full description of these packages can be found in (27). The object managers for the drawing subsystem are all based on the Generic Multiple Object Manager which is contained in the Environment_Types package shown in Figure 42 of Appendix A. Each object manager in the drawing system is in turn accessed as a separate entity by the SATool II system. The Drawable_Class package contains global variables common to all drawable objects.

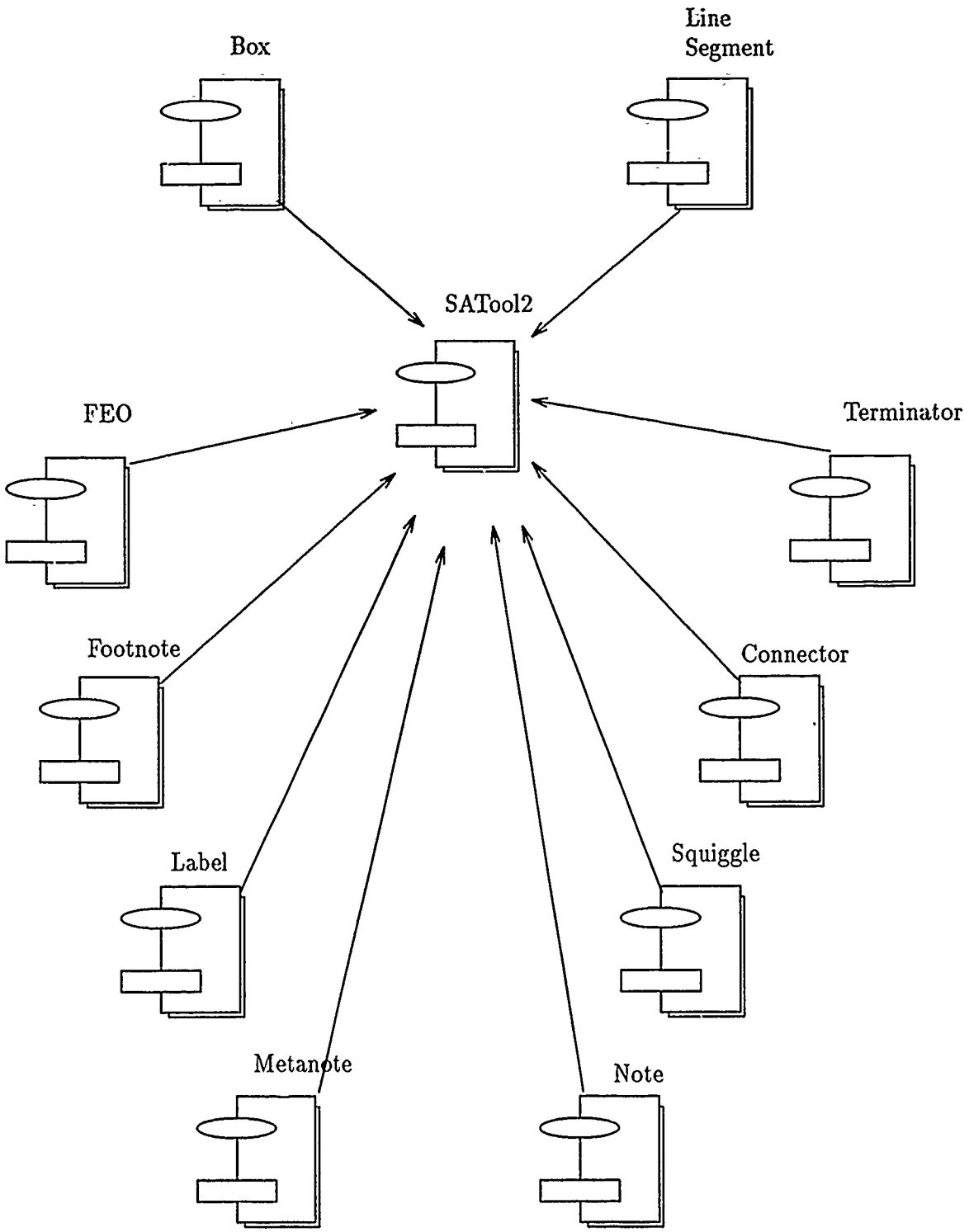


Figure 45. Drawing Subsystem Package Dependencies

Appendix C. *MAGSE Subsystem Implementation*

The MAGSE subsystem packages are presented in Figure 46. This figure shows the relationship between the MAGSE subsystem and SATool II. A full description of these packages can be found in (27). MAGSE_Interface is the global package that allows the SATool II system access to all other packages in the subsystem.

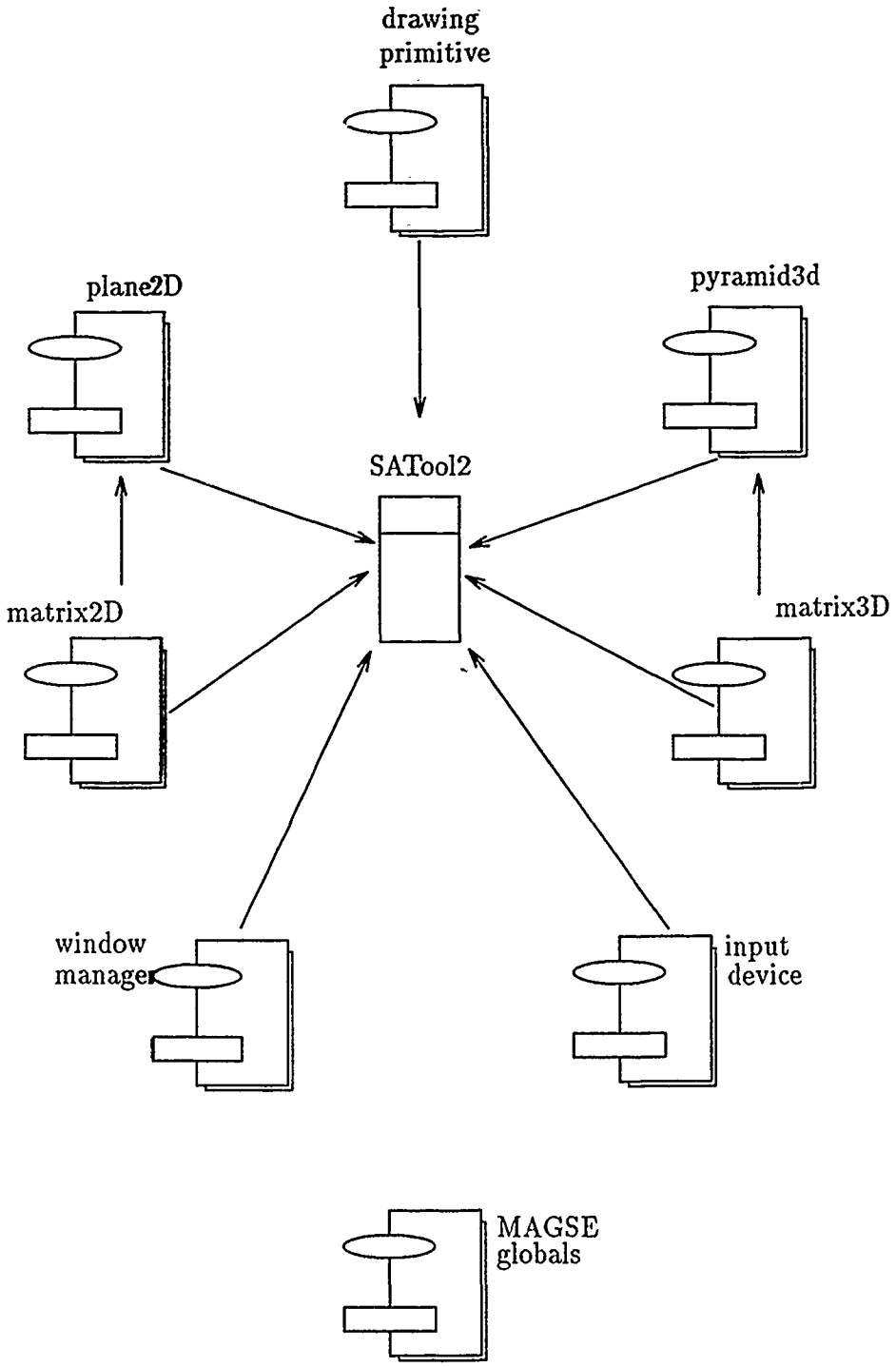


Figure 46. MAGSE Subsystem Package Dependencies

Appendix D. *SATool II User's Manual*

D.1 Introduction

D.1.1 Background and Purpose The SATool II software is an $IDEF_0$ project editor program. $IDEF_0$ stands for ICAM (Integrated Computer Aided Manufacturing) Definition Method Zero and is a rule-based graphical symbol notation language. $IDEF_0$ was originally designed to describe the function model of a manufacturing system or environment. This function model then acted as a structured representation of the system functions and of the information and objects relating those functions. AFIT has expanded the purpose of $IDEF_0$ by using it in the requirements analysis phase of the software life cycle. SATool II adds the power of the computer to this requirements analysis process.

D.1.2 Features SATool II lets you completely create, modify, save, load, check the syntax of, and print the diagrams and data dictionary information of an $IDEF_0$ project faster and easier than using pencil and paper or even a general-purpose computerized paint program. It supplies you with specific tools and services to build diagrams using the $IDEF_0$ language notation; however, it is not designed to teach you how to put a project together. Nevertheless, SATool II's built-in help facility will guide you through its many features and work with you in producing $IDEF_0$ diagrams. The user interface of SATool II should look very familiar to you. It is based on the design of popular case tools, word processing programs, paint programs, desktop publishing programs, and circuit design software.

D.1.3 System Requirements SATool II is an X Window System client program. This means that the computer having the graphical display monitor that SATool II appears on must have an X server and an X window manager program running on it, a keyboard, and a 3-button mouse. However, the SATool II program itself can run remotely on a computer networked with the computer having the graphical display, keyboard, mouse and X programs. The SATool II program currently can run on a computer (host or remote) with the UNIX operating system and a 68000 processor. Refer to the Getting Started section for more information.

D.1.4 Overview This manual appeals to a variety of user personalities. The anxious user can read the Getting Started section and have SATool II running almost immediately. The more methodical user can refer to the Getting Started section and then the Guided Tour section for a walk through the basics of SATool II. The curious user can study the in-depth descriptions in the Objects and Tools section and the Main Screen Menus section. The *I need it right now* user can skim the Getting Started section and the Printing a Window section to get enough information on how to produce an $IDEF_0$ diagram on paper as quickly as possible.

D.2 Getting Started

D.2.1 Quick Start If you don't have time to read the manual, the steps below can get you up and running right now. The workstation you are using should have a graphical display, keyboard, 3-button mouse, and an X server and X window manager running on it. If you are missing any of these items, you will need to refer to the other sections in this manual before starting. Here are the Quick Start steps:

1. Create an SATool II directory and change to that directory.
2. Type 'satool2 RETURN'. A window outline will soon appear on the screen.
3. Use the mouse to position the window outline to a desired screen location; press the left button to finally place the window.
4. When the SATool II screen appears, use the left mouse button to click on PROJECT on the left side of the main menu.
5. In the PROJECT menu, click on the *Create Project* selection.
6. When the dialogue box appears, type in the name of the project you wish to create and press RETURN.

You are now ready to create and modify an $IDEF_0$ diagram or multiple $IDEF_0$ diagrams. You begin by creating a diagram via the Tools Window. First you click on CREATE and then on DIAGRAM. You have the option of filling in the diagram header information at this time. If you prefer not to do so, the information can be updated later on via the data dictionary editor (DICTIONARY button). At this point you can create objects in the diagram by clicking on CREATE and any of the twelve objects on the Objects Window. Subsequent diagrams are created hierarchically by clicking on CREATE then on DIAGRAM and then on a target activity box in the drawing window. To save your work, click on PROJECT. Then click on the *Save Project* selection. To exit SATool II, click on PROJECT. Then click on the *Exit Program* selection.

D.2.2 Operating Environment SATool II uses the window and graphics features of the X Window System produced by MIT. To run SATool II, you must have the following:

- A computer workstation or personal computer with an X server program and an X window manager running on it, a graphical display, a keyboard, and a 3-button mouse
- A host computer which runs the UNIX operating system and has a 68000 processor (The host computer can be the same as the computer workstation or personal computer)

- Directory path access to the executable version of the SATool II program

You can execute SATool II on a remote computer by using the UNIX remote shell command (rsh). Refer to the *-display* command line option for more information.

D.2.2.1 Display One of the outstanding aspects of the X Window System is its ability to send graphical display information over a network. The display option tells SATool II and the X Window System the name of a remote computer and display to exchange graphical information with. An application program that uses the X Window System is referred to as an X client. An X client can execute on one computer and have all of its graphical input and output handled on the display, keyboard, and mouse of a remote computer connected through a network. When an X client begins execution, the X Window System automatically chooses the most efficient communication route between the client and the display. If the client executes on the same computer where the display is located, this route is rather simple. However, the UNIX remote shell command, the X network protocol, and an X server program make it possible to remotely execute an X client. Below is an example scenario on how to execute a remote X client and establish the remote connection back to the host computer:

> Two UNIX workstations, each with a graphical display, are on the same network and have the same file server. The workstations are named alpha and beta. Beta runs programs three times faster than alpha. A user who logs into alpha can remotely use beta (rlogin or rsh) without a password check. The same is true for a user on beta. The user on alpha has already entered the *xinit* command and has an X window manager up on the screen. The user on beta is running another type of window system. The user of alpha wants to run SATool II on beta because the computer is faster, but he wants alpha to handle all the display input and output because that is the computer he is at. To do this, he enters the following commands:
>
> ```
> %alpha: xhost beta
> %alpha: rsh beta "SATool II -display alpha:0" &
> ```
>
> Here is a list of important points to remember when following this example:

- An X server with an X window manager and Xterm client must be running on the workstation of the console that you are at. In the example above, that workstation is alpha.

- You enter the commands in the example above in the xterm window on your computer.
- The *xhost* tells the X server on your computer which remote computers are allowed to make connections to the X server on alpha.
- The *rsh beta SATool II* tells UNIX to run SATool II on beta's processor. Beta must have direct access to the SATool II executable code (symbolic links cause problems). The X server on alpha does not need to know anything about the location or purpose of the SATool II program. It only needs to know that SATool II is an X client.
- The *-display alpha:0* tells SATool II to receive all of its graphical input from and send all of its graphical output to display 0 on alpha. The displays attached to a single processor in UNIX are numbered starting with 0. Because alpha has only one display, the number for that display is 0.
- The ampersand tells UNIX to begin the program on beta as a separate process and return input and output functionality to the xterm window on alpha.

D.3 A Guided Tour

D.3.1 Introduction This section walks you through the basics of SATool II. The tour is designed to give you hands on experience immediately. You learn how to use some of the tools and, at the same time, how to give some simple commands and responses to the acknowledge windows, menus, confirm windows, and dialogue boxes that appear on the screen. When you finish this section you will be able to create, modify, save, and load an $IDEF_0$ project using SATool II. Note that the word *click* is used frequently throughout this manual. This refers to pressing and releasing a mouse button.

D.3.2 The Main Screen In the GETTING STARTED section you learned how to execute SATool II either on your computer or remotely. Follow the steps in that section at this time to get SATool II started. Figure 17 shows the main screen face of SATool II. This same screen face should be in a window on your display. The main screen has the following parts:

- Title Window. The title window is located at the top of the main screen and identifies the program as SATool II.
- Main Menu Window. The main menu window is located just below the help window. It contains the names of pull-down menus that list commands that you can give SATool II.
- Tools Title Window. The tools title window is located at the top of the Tools Window and identifies the Tools Window used to edit diagrams.

- Tools Window. The tools window is located on the left side of the main screen below the Tools Title Window. The tools window contains a button for each tool that you can use to place or modify an $IDEF_0$ object in a diagram.
- Objects Window. The objects window is located on the left side of the main screen just below the Tools Window. It contains a button for each of the $IDEF_0$ objects that you can place in a diagram.
- Drawing Window. The drawing window is the large central work area for SATool II. This is where you will create and modify a diagram in an $IDEF_0$ project. In the drawing window you will see the top and bottom headers of an $IDEF_0$ diagram. The drawing window is also where SATool II displays pop-up windows and various views of the diagrams making up a project.
- Cursor. The cursor changes form when you move from one window to the next to remind you of the purpose of the window. In the case of the drawing window, the cursor is an arrow.

D.3.3 The Keyboard and Mouse SATool II accepts input from you through the keyboard and through the mouse. On the keyboard, SATool II recognizes only the keys that normally appear on a typewriter. SATool II only uses the keyboard with a dialogue box. When a dialogue box is displayed you can use the keyboard to enter characters and press RETURN for SATool II to accept the characters. SATool II uses the mouse for all other user input. Moving the mouse also moves the cursor on the screen. SATool II detects the cursor position and its movement. It also detects when you press a button on the mouse.

D.3.4 Using Tools and Objects When you begin SATool II, you have a project with no diagrams. The drawing window displays a template of the $IDEF_0$ context diagram A-0. If you click on PROJECT in the main menu and then click on *Load Project*, SATool II will prompt you for a project name and load the project into the SATool II environment. To add an $IDEF_0$ object to the diagram, click on the CREATE button in the Tools Window and then click on one of the objects in the objects window. Once you have created an object, you can delete it from the diagram by selecting the DELETE button, clicking on the target object type in the Object Window, and then clicking on the desired object in the Drawing Window.

D.3.5 Creating and Viewing a Project SATool II is not a general-purpose paint program, therefore it doesn't just let you pick an object and do whatever you want with it in the drawing window. It knows enough about $IDEF_0$ to guide you in using the objects and tools properly to create $IDEF_0$ diagrams. One example of this guidance is the operation for creating diagrams. Once you have created the A-0 diagram, you can use this operation only in conjunction with a box. When you select the CREATE tool button followed by the DIAGRAM object button, SATool

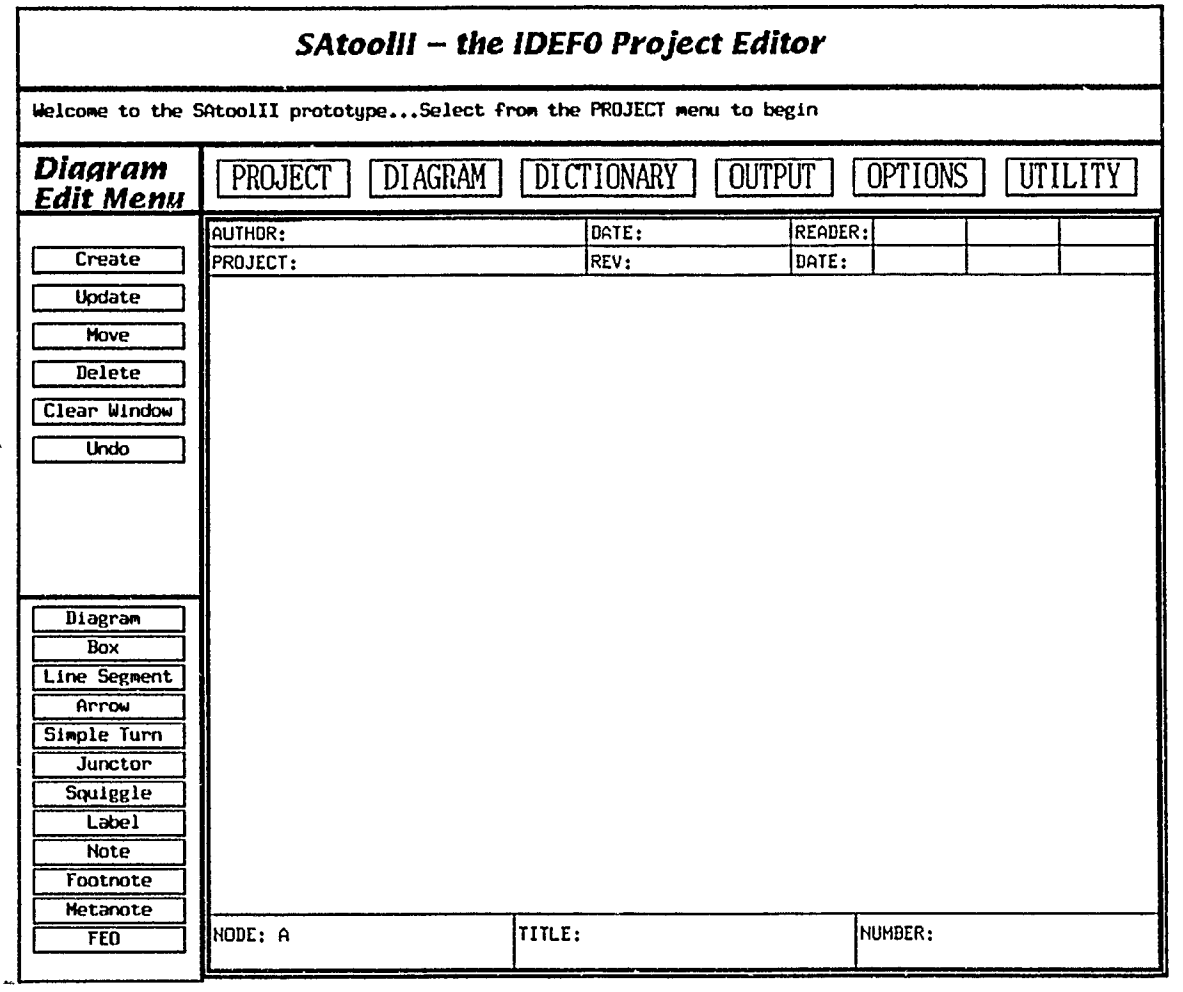


Figure 47. SATool II Main Screen Face

II waits for you to select a box in the current diagram. When you do, it replaces the diagram in the drawing window with the child diagram of that box. This diagram will be empty if you just created it. To return to the diagram that the box belongs to, click on the DIAGRAM button in the Main Menu Window and select the *Return to Parent* selection. SATool II will return the parent diagram to the drawing window. You can be assured that anytime SATool II changes the diagrams in the drawing window that it has saved the contents of the current diagram in memory.

D.3.6 Saving and Loading a Project To save the project you have been working on, click on PROJECT in the Main Menu Window. Select *Save Project* if you wish to save the project under its current name. Otherwise, select *Save Project As* if you wish to save the project under a different name.

To load a project, click on PROJECT in the main menu and select *Load Project.* SATool II will then ask you for the name of the project. Enter the project name without any extensions attached. An error message will appear in an Acknowledge Window if the project was not found in the current directory. You will be asked to confirm a load operation if you attempt to load a project when there is a project already in the SATool II environment that hasn't been saved already or cleared from the system.

D.3.7 Error Handling Whenever SATool II detects an error, it will display an acknowledge window containing an error message. If the error is not serious, SATool II will just wait for you to press a key or click a button to continue. If the error is serious, Satool II will display a confirm window after you press a key or click a button. In the confirm window SATool II will ask if you want to continue the session or let the error go to the operating system and abort the program. When a confirm window appears for a serious error, your best move is to not abort the program. Select *NO* and then immediately try to save your project and exit SATool II. SATool II keeps track in an error file of any errors that occurred in the current session. Check this file after you exit SATool II if a serious error occurs. If will give you a better idea of what happened and what you can do about it.

D.3.8 Exiting SATool II To exit SATool II, first save your current project, then click on PROJECT in the main menu and select *Exit Program.* If you haven't saved the project, SATool II will ask you to confirm the exit operation without saving the project first.

D.3.9 Summary This section walked you through the basics of SATool II. In this section you learned how to:

- Start SATool II
- Identify different parts of the main screen

- Use the keyboard and mouse
- Use the objects and tools
- Create and view a project
- Save and load a project
- Exit SATool II

To print the contents of a diagram or project refer to the section on PRINTING A WINDOW.

D.4 Objects and Tools

D.4.1 Introduction This section describes the objects in the objects window and the tools in the tools window that you use together to create and modify diagrams. To use a tool with a specific object you must first click on the tool button and then on the object button.

D.4.2 The Objects SATool II is an $IDEF_0$ project editor. It was designed to supply you with a familiar $IDEF_0$ environment to work in; hence, the purpose of the $IDEF_0$ objects in the objects window. Below is a description of each of the objects.

D.4.2.1 Box A box represents an activity on an $IDEF_0$ diagram. Along with its location in a diagram and its name, it also has many activity attributes connected with it. The activity attributes can be inspected via the DICTIONARY button in the Main Menu Window.

D.4.2.2 Line Segment A line segment, in connection with other line segments and terminators, shows the flow of data from one box to the next. This data is identified by a label related to the line segment. SATool II restricts a line segment to lie either vertically or horizontally on a diagram. This means that the angle between two line segments is 90 degrees or 180 degrees. One or more line segments connected by terminators represent a data element. Along with the locations of these line segments in a diagram, the data element they represent has many attributes connected with it. The data element attributes can be inspected via the DICTIONARY button in the Main Menu Window.

D.4.2.3 Simple Turn A simple turn is a symbol that you attach at the end of a line segment when you want the data element to turn in one of four directions. It has two connections, one at each end of the curve that forms the turn. The system assumes that the line segments attached at each end of the simple turn belongs to the same data element. This assumption is based on the current $IDEF_0$ syntax rules.

D.4.2.4 Junctor A junctor is similar to a simple turn in the sense that it connects lines to lines. Its shape allows up to three lines to be attached to each end of it. However, SATool II does not assume that each line segment attached to the junctor is part of the same data element. Via dialogue windows you let the system know what line belongs to what data element. Junctors are normally used with data elements that must be broken down into separate sub-elements or when the same data element must go to more than one destination.

D.4.2.5 Arrow There are six types of objects used to create the $IDEF_0$ arrows: simple arrow head, incoming tunnel parenthesis, outgoing tunnel parenthesis, to-all circle, from-all circle, and dot arrow head. When creating an arrow a menu will pop up so you can select one of the above objects.

D.4.2.6 Squiggle A squiggle is a device used when crowding on part of a diagram causes poor readability. You use it to relate a label to a line segment or a footnote marker to a line segment when you cannot place the label close enough to the object. A label has two endpoints, one near the label or footnote marker and one near the line segment.

D.4.2.7 Label A label is used to identify the specific data element a line segment represents. It is automatically created and positioned when the first line segment of a data element is created. If it is not positioned in a good location, you can move the label via the MOVE button to a better position. If this is not possible because of crowding in the diagram, use a squiggle to show the relationship between a label and a line segment.

D.4.2.8 Note A note lets you put nongraphical information of an analysis into a diagram. If the object that the note refers to is not obvious, use a squiggle to clarify the situation.

D.4.2.9 Footnote A footnote is the same as a note except that you can use it instead of a note if crowding in part of the diagram forces you to move the text to another location.

D.4.2.10 Metanote A metanote is not part of the $IDEF_0$ description in a diagram. Instead it is observations about the diagram, such as the way it is laid out or the choice of label or box names.

D.4.2.11 FEO An FEO, or for exposition only, is a special effect feature you can place in a diagram. It is not part of a diagram, but is used to illustrate the purpose of a particular action taken on the diagram. SATool II implements this as a character phrase that you can add to a diagram.

D.4.3 The Tools You can use a tool from the tools window to perform a drawing operation with an object from the objects window. Each operation in the Tools Window expects you to click on the object type you wish to have the operation performed on before the operation can be performed. Below is a description of what you can do with the tools.

D.4.3.1 Create This tool adds an object to a diagram. After selecting the CREATE button, click on the desired object button. SATool II will then output one or more pop up menus to gain more information about the object you want to create. Then, it waits for you to mark the position of the upper left corner of the object. If you decide later that you want to place the object in a different position, use the MOVE tool.

D.4.3.2 Update This tool lets you change the text in the diagram header, or in a note, footnote, metanote, or an FEO. After selecting the UPDATE button, click on the object type you want to update in Object Window. SATool II will then wait for you to click on the object you want to update in the diagram.

D.4.3.3 Move This tool moves an object from one location in a diagram to another. After clicking on the MOVE button, click on the object type you want to move in the Object Window. SATool II will then wait for you to indicate which object in the diagram you want to move. It then waits for you to mark the new position of the upper left corner of the object.

D.4.3.4 Delete This tool removes an object from a diagram. After selecting the delete tool, SATool II waits for you to indicate which object in the diagram you want to delete. This is a very powerful tool, but it is also very particular. SATool II will not let you use this tool to leave behind an $IDEF_0$ disaster. Consequently, deleting one object may mean deleting several objects with it to keep the diagram in order. Before SATool II takes such action if will ask for a confirmation from you on what it plans to do. If the drawing window has parts of objects left in it after a delete operation, click on the *Redisplay DIAGRAM* selection in the DIAGRAM menu to clean up the drawing window.

D.4.3.5 Clear Window This tool is used to clear the Drawing Window of its current diagram. Clearing the window does not delete the diagram. That must be done with the DELETE button. All the CLEAR WINDOW button does is remove the diagram from sight. The diagram can be called back up again by via a selection from DIAGRAM button in the Main Menu Window.

D.4.3.6 Undo The undo tool reverses the most recent drawing operation that you just had SATool II perform with an object. If you make a drawing

mistake and you want SATool II to undo it, this is the tool to use. As soon as you click on undo, the previous operation is reversed.

D.4.4 Summary This section described the objects in the objects window and the tools in the tools window. In this section you learned how to use an object and tool from these windows to create and modify diagrams.

D.5 Main Screen Menus

D.5.1 Introduction SATool II has six pull-down menus on the main screen: PROJECT, DIAGRAM, DICTIONARY, OUTPUT, OPTIONS, and UTILITY. To pull down a menu and display its contents, click on the menu button. As you move the cursor within the menu, SATool II highlights the command under the cursor. To select the command, position the cursor over it until it is highlighted, then click on it. If you click outside the menu, the menu disappears and no command is selected. Many of the commands you select in the pull-down menus will bring up dialogue boxes, acknowledge windows, confirm windows, or other menus. To cancel any of these commands, do the following based on the type of window in use:

- Menu - click outside the menu
- Confirm window - click on the cancel button
- Acknowledge window - press any button or key because no action is ever performed

Anytime an error occurs, SATool II will display the error information in an acknowledge window. This type of window has no effect on the state of the project, but it will stay on the screen until you either press a key or click a button. The rest of this section provides a detailed description of the commands listed in each pull-down menu.

D.5.2 PROJECT Menu The project menu supplies you with a list of commands you can use on an $IDEF_0$ project.

D.5.2.1 Create Project Allows you to create a project from scratch. The SATool II system requires unique project names. If you try to create a project using the name of an exiting project, the system will output an error message to the screen and allow you to try again.

D.5.2.2 Load Project The load project command uses a dialogue box to prompt you for the name of the project to load. If a project is already in memory, it will also use a confirm window to ask if you want to destroy that project in memory only.

D.5.2.3 Save Project As The save project as command uses a dialogue box to prompt you for the name that you want to save a project under.

D.5.2.4 Save Project The save project command saves the information in memory into the project files with the current project name.

D.5.2.5 Lay Out Project The lay out project command creates all the diagrams of a project completely from the essential model information.

D.5.2.6 Derive Project The derive project command derives the contents of the essential model and data dictionary completely from the drawing model (diagram) information.

D.5.2.7 Show Directory The show directory command uses an acknowledge window to display the names of the projects in the current default directory.

D.5.2.8 Change Directory The change directory command uses a dialogue window to prompt you for the name of a new default directory.

D.5.2.9 Clear Project Environment Erases the current project from the SATool II project environment.

D.5.2.10 Exit Program The exit program command ends the SATool II program. If you haven't saved the current project before ending, SATool II uses a confirm window to ask if you want to continue the exit operation.

D.5.3 DIAGRAM Menu The diagram menu supplies you with a list of commands you can use on an $IDEF_0$ diagram.

D.5.3.1 Show A-0 Diagram Outputs to the Drawing Window the first diagram in the hierarchy, the A-0 diagram.

D.5.3.2 Select by Diagram Name This operation outputs the diagram that you specify by name to the Drawing Window.

D.5.3.3 Select by Hierarchy First, the operation outputs a hierarchical view of the diagrams via a pyramid scheme of lines and boxes. The view can go as far as twenty levels deep. You are then allowed to click on one of the boxes shown in the hierarchy diagram. The chosen diagram is then output to the diagram window.

D.5.3.4 Go to Child Diagram Allows downward traversal of the diagram hierarchy from parent to child, one layer at a time.

D.5.3.5 Return to Parent Allows upward traversal of the diagram hierarchy from child to parent, one layer at a time.

D.5.3.6 Refresh Diagram The redisplay diagram command clears the drawing window and redraws all the diagram contents.

D.5.4 DICTIONARY Menu When the DICTIONARY button is pressed, you are given four choices. You can view the data dictionary entries for all activities, all data elements, a single activity, or a single data element. Depending on which choice is made, the system will then bring up a new set of windows.

There are separate edit screens for the activity and data element entries (see Figure 48 and Figure 49). In general, each screen has four windows: a title window, an edit window, an options window, and a text window. The title window is used to identify the Data Dictionary editor. The text window shows the data dictionary entry information for each activity or data element, one at a time. The edit window allows changes to be made to the text via its edit buttons. The entry is then automatically updated in the screen. The options menu allows the user to traverse through all the data dictionary entries via the *Next Activity* or *Next Data Element* button. Each entry may be several pages long and can be viewed page by page with the em Next Page button. This window system can be exited via the *Exit* button.

D.5.5 OPTIONS Menu The options menu supplies you with a list of commands you can use to change user-definable options referenced by SATool II to determine the user interface appearance and degree of service.

D.5.5.1 Grid The grid command lets you bring up a X/Y grid in the drawing window to help you in positioning objects. The command uses a menu so you can select on or off. The default value is off.

D.5.5.2 Drawing Font The drawing font command uses a submenu to display the names of X Window System fonts that you can uses for the text in the diagrams in the drawing window. The font name you choose stays in effect for the current session until you change it to something else. The default value is "9x15".

D.5.5.3 Line Thickness The line thickness command uses a submenu to display a choice of line thicknesses for the objects drawn in a diagram in the drawing window. This line thickness stays in effect for the current sessions until you change it to something else. The default value is 1.

D.5.5.4 Line Rerouting The line rerouting command lets you tell SATool II if you want it to automatically reroute any remaining line segments in the current diagram following an add, move, or delete tool operation. The purpose of the line

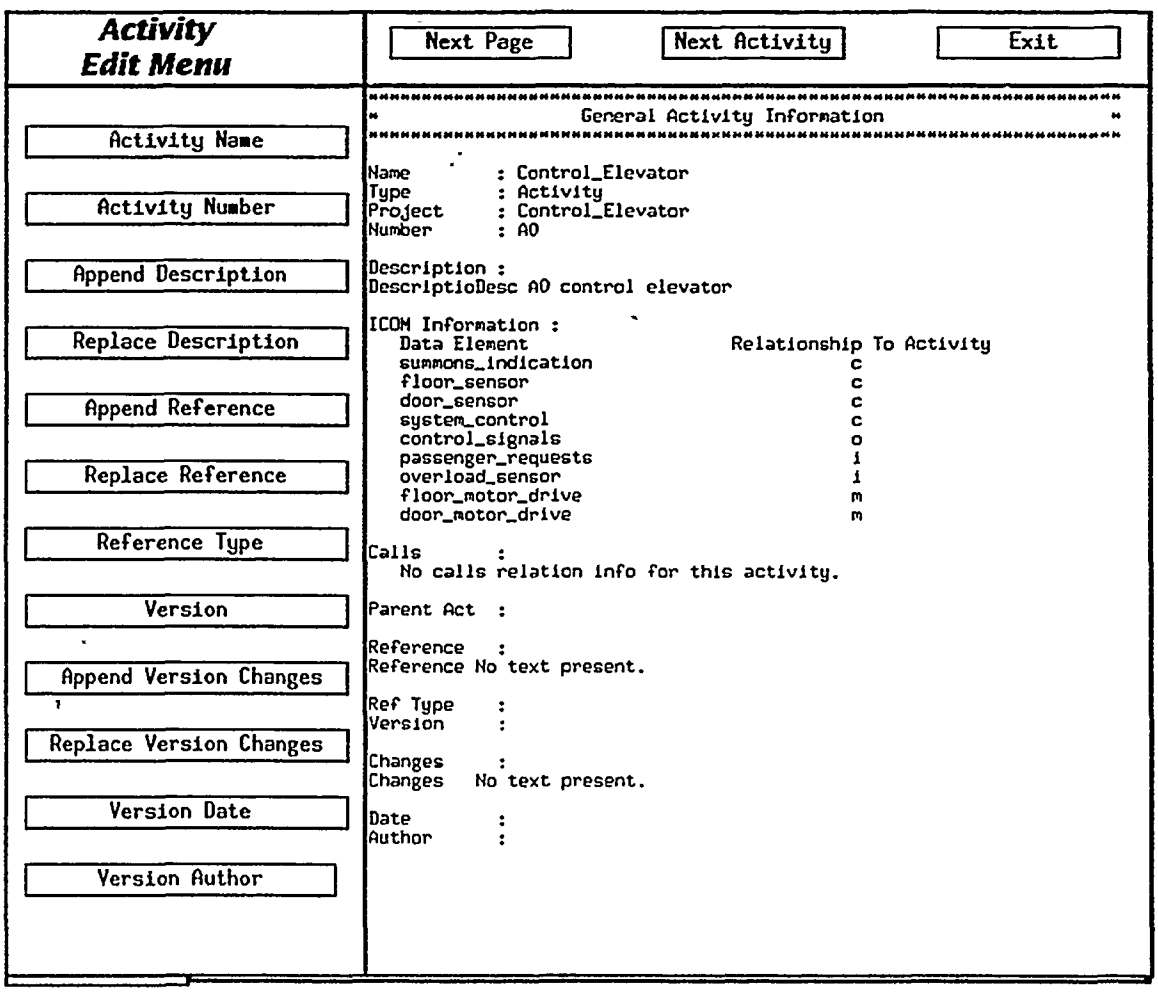


Figure 48. Activity Data Dictionary Editor Screen

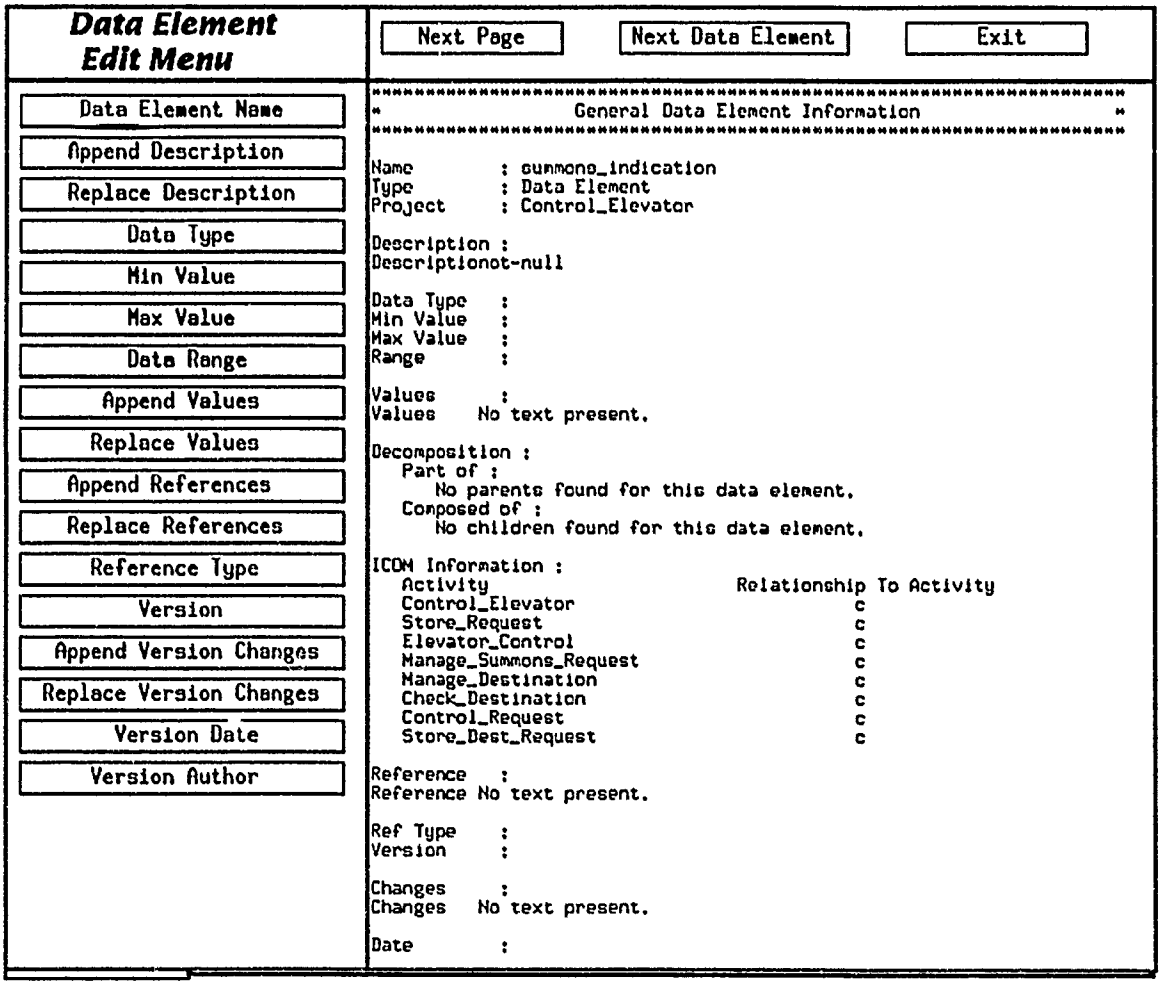


Figure 49. Data Element Data Dictionary Editor Screen

rerouting is to ensure lines do not pass through boxes and that they take the most direct route possible within the restriction of the 90 degree and 180 degree rule. The command uses a submenu for you to select on or off from. The default value if off.

D.5.5.5 Dimensions The dimensions command lets you tell SATool II how you want the diagrams in the drawing window displayed, either in two-dimensions or three-dimensions. The command uses a submenu for you to select 2-D or 3-D from. The default value is 2-D.

D.5.5.6 Syntax Observance The syntax observance command lets you tell SATool II if you want it to check the $IDEF_0$ syntax of the current diagram in the drawing window every time you perform a drawing operation. The command uses a submenu for you to select yes or no from. The default value is no.

D.5.5.7 Help Level The help level command lets you tell SATool II the level of help you want it to give you. This affects the amount of help information that appears on the screen each time you click on an object, tool, or menu button. The default value is 0, which means that only the help window is used to display help information. If you use any level greater than 0, SATool II uses an acknowledge window to display the help information. The command uses a submenu for you to select a level from.

D.5.5.8 Warning Beep The warning beep command lets you tell SATool II if you want a warning beep to sound each time SATool II displays an acknowledge window. The command uses a submenu for you to select yes or no from. The default value is no.

D.5.6 UTILITY Menu The utility menu supplies you with a list of commands you can use to select other miscellaneous functions offered by SATool II.

D.5.6.1 Check Syntax The check syntax command tells SATool II to check the $IDEF_0$ syntax of the current project and report the errors on the screen. This syntax checking process involves asserting facts about the project, applying a rule base to the facts, and then listing any errors. The command uses a confirm window to ask if you are sure you want to check the syntax.

D.5.6.2 Refresh Screen This operation is used to refresh the text of all the windows currently shown on the screen.

D.6 Printing A Window

D.6.1 Introduction SATool II currently does not have a built-in capability to send a diagram or project to a printer or even create a file that can later be sent to

a printer. But you can still easily obtain a printed copy of the $IDEF_0$ diagrams that you created using SATool II. This section briefly describes the X client programs used to capture a window and print it. It also tells you how to use these programs to obtain a laser printer copy of an $IDEF_0$ diagram created using SATool II. All the SATool II screen and menu illustrations in this manual were obtained using this method.

D.6.2 X Clients for Window Capturing and Printing Below is a short description of the four X client programs that are used in capturing (dumping) and printing X windows.

D.6.2.1 xwd : X Window Dump Program This program stores a window image in a specially formatted X Window dump file.

Program Command Line Options:

```
-help          Shows 'Usage:' command syntax
-nobdrs        Pixel border is not included in window dump
-out <file>    Output file name; default is standard out
-root          Makes a dump of the entire root window
```

D.6.2.2 xpr : X Window Dump Translator Program This program translates an X Window dump file into a printable output file.

Program Command Line Options:

```
-scale  <scale>    Scales bits; 3 changes 1X1 to 3X3
-height <inches>   Maximum height of window on page
-width  <inches>   Maximum width of window on page
-left   <inches>   Left margin otherwise image is centered
-top    <inches>   Top margin otherwise image is centered
-landscape         Prints image in landscape mode; default matches
                   window longest side to paper longest side
-portrait          Prints image in portrait mode; see above
-rv                Reverses foreground and background colors
-compact           Compresses white pixels on PostScript only
-output <file>     Output file name; default is standard out
-append <file>     Appends image to previously produced xpr file
-noff              Appended window appears on same page as first
-split <n>         Splits window into several pages
-device <device>   Specifies the device format to use for output.
                   For:  LN03              -device ln03
                         LA100             -device la100
```

```
PostScript          -device ps
IBM PP3812          -device pp
Apple LaserWriter   -device lw or ps
```

Special Notes:

- The LN03 can handle windows up to 2/3 of the screen size
- LA100 pictures are always in portrait mode with no scaling
- Postscript cannot handle -append, -noff, or -split options

D.6.2.3 xdpr : X Window Dump, Translate, and Print Program This program runs the commands xwd, xpr, and lpr(1) to dump an X Window to a file, translate the file contents to a printable form, and send the translated file to a laser printer.

Program Command Line Options:

```
-filename          Specifies existing file containing xwd dump
-P<printer>        Specifies name of printer to be used
-device <device>   Specifies type of printer; see xpr options
-help              Displays list of options for xdpr

( All other options are passed to xwd, xpr, and lpr(1) )
```

D.6.2.4 xwud : X Window Undump Program This program undumps an X Window dump file into the coordinates of the original window

Command Line Options:

```
-help          Displays list of options
-in <file>     Specifies input file; default is standard input
-inverse       Undumps file in reverse video; monochrome dumps only
```

D.6.3 Printing an IDEF$_0$ Diagram and Project Follow these steps to create a printed copy of an IDEF$_0$ diagram that was created using SATool II.

Note: If you already have SATool II running, skip to step 6.

1. Have an X Window Manager running on your workstation and have two xterm windows on the screen.
2. In the first xterm window enter

```
satool2
```

3. When the window outline appears on the screen, position the SATool II window so that you will have access to the second xterm window.
4. Click the left mouse button to mark the position of the SATool II window outline and bring up the SATool II program.
5. Use SATool II to create an $IDEF_0$ diagram or load a project into SATool II using the Load Project selection in the PROJECT menu.
6. Bring up in the drawing window of SATool II the $IDEF_0$ diagram that you want a printed copy of.
7. Move the cursor into the second xterm window and enter

```
xwd > diagram_name.xdmp
```

   where diagram_name is some meaningful name for the $IDEF_0$ diagram. The xwd program copies the $IDEF_0$ diagram into an X Window dump file. When the xwd program starts up, the cursor will turn to a cross-hair.
8. Move the cross-hair inside the SATool II drawing window and press the left mouse button. The xwd program will beep once to say it started storing the window in a file and beep twice when it is done.
9. Move the cursor back into the second xterm window and enter

```
xpr -device ps < diagram_name.xdmp -output diagram_name.ps
```

   The xpr program translates the $IDEF_0$ diagram file from the X Window format to a PostScript format and puts the translated information into a new file.
10. With the cursor still in the second xterm window, enter

```
lpr -Pprinter_name  diagram_name.ps
```

    The lpr program sends the $IDEF_0$ diagram file to the laser printer whose name is printer_name. The printing process takes about 10 minutes.

To get printed copies of all the diagrams in a project, follow steps 6 through 10 for each diagram.

Appendix E. *SATool II Configuration Guide*

E.1 Introduction

The SATool II system consists of four subsystems: The Machine-Independent Ada Graphical Support Environment (MAGSE), Essential Model Manager, Drawing Model Manager, And Graphical User Interface(GUI). The MAGSE subsystem is dependent on the Ada source code interface (Ada bindings) to the X Window system provided by the Science Applications International Corporation (SAIC) to perform its graphical user interface and drawing chores (27:A-1). The Essential Model Manager is dependent on the CLIPS/Ada source code to perform its syntax checking functions. The Drawing Model Manager is dependent on the Generic Multiple Object Manager provided by the Essential Model Manager. Finally the GUI is dependent on all the above subsystems to perform all of its operations.

This configuration guide details the compiling order of the MAGSE, Essential Model Manager, Drawing Model Manager, and GUI files required to create the SATool II executable. The configuration guide for the Ada bindings to the X Window system is found in (27). The configuration guide for the CLIPS/Ada source code is found in (17). The next section presents the SATool II configuration file using a UNIX script file format to better illustrate the configuration of the system.

E.2 SATool II Configuration File

```
#####################################################################
#   SATool II Package Compiling and Program Creation Order          #
#                                                                   #
# Instructions : Execute Parts A through D, with their              #
#                respective levels, to produce the SATool II        #
#                system executable program.                         #
#                                                                   #
#         Note : The compilation order for the Essential Model      #
#                Manager subsystem and the MAGSE subsystem are      #
#                interchangeable.  Part B could be compiled         #
#                before Part A since there are                      #
#                no dependencies between them.                      #
#####################################################################

#####################################################################
# Part A : Machine-independent Ada Graphics Support Environment     #
#          Packages (MAGSE)                                         #
#####################################################################
```

```
# Level A-1 (specs)
ada magse_interface_spec.a
ada magse_globals_spec.a

# Level A-2 (bodies)
ada magse_interface_body.a
ada magse_globals_body.a
ada magse_window_manager_body.a
ada magse_drawing_primitive_body.a
ada magse_input_device_body.a
ada magse_matrix2d_stack_body.a
ada magse_plane2d_body.a

# Level A-3 (optional)
# Only compile these files if 3-D rendering will be used
ada magse_matrix3d_stack_body.a
ada magse_pyramid3d_body.a


#######################################################################
# Part B : Essential Model Packages                                   #
#######################################################################

# Level B-1 (specs)
ada es_genev_spec.a
ada es_proj_spec.a
ada es_activ_spec.a
ada es_datel_spec.a
ada es_hista_spec.a
ada es_ICOM_spec.a
ada es_conof_spec.a
ada es_mnuio_spec.a
ada es_calls_spec.a
ada es_factu_spec.a
ada es_esmio_spec.a
ada es_clpwm_spec.a
ada es_load_save_spec.a


# Level B-2 (bodies)
ada es_genev_body.a
ada es_proj_body.a
```

```
ada es_activ_body.a
ada es_datel_body.a
ada es_hista_body.a
ada es_ICOM_body.a
ada es_conof_body.a
ada es_mnuio_body.a
ada es_calls_body.a
ada es_factu_body.a
ada es_esmio_body.a
ada es_clpwm_body.a
ada es_load_save_body.a

#######################################################################
# Part C : Drawing Model Packages                                     #
#######################################################################

# Level C-1 (specs)
ada dr_dr_sp.a
ada dr_ar_sp.a
ada dr_bx_sp.a
ada dr_cn_sp.a
ada dr_fe_sp.a
ada dr_ft_sp.a
ada dr_jc_sp.a
ada dr_lb_sp.a
ada dr_ln_sp.a
ada dr_mt_sp.a
ada dr_nt_sp.a
ada dr_sq_sp.a
ada dr_st_sp.a
ada dr_dg_sp.a
ada dr_drawable_io_spec.a

# Level C-2 (bodies)
ada dr_dr_bo.a
ada dr_ar_bo.a
ada dr_bx_bo.a
ada dr_cn_bo.a
ada dr_fe_bo.a
ada dr_ft_bo.a
ada dr_jc_bo.a
ada dr_lb_bo.a
```

```
ada dr_ln_bo.a
ada dr_mt_bo.a
ada dr_nt_bo.a
ada dr_sq_bo.a
ada dr_st_bo.a
ada dr_dg_bo.a
ada dr_drawable_io_body.a

#########################################################################
# Part D: Graphical User Interface Packages                             #
#########################################################################

# Level D-1 (specs)
ada sa_title_window_spec.a
ada sa_drawing_window_spec.a
ada sa_tools_title_window_spec.a
ada sa_objects_window_spec.a
ada sa_dd_text_window_spec.a
ada sa_dialog_window_spec.a
ada sa_options_button_spec.a
ada sa_output_button_spec.a
ada sa_utility_button_spec.a
ada sa_project_button_spec.a
ada sa_dd_activity_edit_window_spec.a
ada sa_dd_data_element_edit_window_spec.a
ada sa_dd_main_menu_window_spec.a
ada sa_dd_edit_menu_title_window_spec.a
ada sa_dictionary_button_spec.a
ada sa_drawable_objects_spec.a
ada sa_clear_diagram_button_spec.a
ada sa_create_button_spec.a
ada sa_delete_button_spec.a
ada sa_move_button_spec.a
ada sa_update_button_spec.a
ada sa_tools_window_spec.a
ada sa_diagram_button_spec.a
ada sa_main_menu_window_spec.a

# Level D-2 (bodies)
ada sa_title_window_body.a
ada sa_drawing_window_body.a
ada sa_tools_title_window_body.a
```

```
ada sa_objects_window_body.a
ada sa_dd_text_window_body.a
ada sa_dialog_window_body.a
ada sa_options_button_body.a
ada sa_output_button_body.a
ada sa_utility_button_body.a
ada sa_project_button_body.a
ada sa_dd_activity_edit_window_body.a
ada sa_dd_data_element_edit_window_body.a
ada sa_dd_main_menu_window_body.a
ada sa_dd_edit_menu_title_window_body.a
ada sa_dictionary_button_body.a
ada sa_drawable_objects_body.a
ada sa_clear_diagram_button_body.a
ada sa_create_button_body.a
ada sa_delete_button_body.a
ada sa_move_button_body.a
ada sa_update_button_body.a
ada sa_tools_window_body.a
ada sa_diagram_button_body.a
ada sa_main_menu_window_body.a

# Level D-3 (satool2 driver)
ada satool2.a

# Level D-4 (load the driver)
a.ld -o satool2 satool2 /usr/X/lib/libX11.a utils.o

# Execute driver by typing:  satool2 RETURN

## end of file ##
```

Appendix F. *SATool II Test Cases*

This appendix contains the unit test cases and integration test case used to evaluate the SATool II application. The unit test cases were developed based on the types of conditions the procedures safeguarded against (white box testing). If an error is successfully detected, the logic of the code is then reviewed and corrected. The integration test is based on re-creating the $IDEF_0$ diagrams of a "real world" project in the SATool II development environment. This test ensures the coverage of conditions not considered during the white box testing of each system operation.

F.1 Unit Test Cases

Unit test cases were performed for each operation in the application. The following subsections detail the test cases for the operations of each window in the application. A successful test is defined as a failure to obtain the expected result.

F.1.1 General Main Screen Windows Unit tests for the SATool II Title Window, Tools Title Window, and Drawing Window consisted of simply bringing them up and showing text or drawings on them.

F.1.2 Main Menu Buttons The Main Menu Window consists of six buttons, each with its own submenu of operations. Following is a description of the test cases for each of these operations.

F.1.2.1 Project Button

- Create Project
  1. Create project in cleared system environment with new name (simple case).
  2. Create project when there is one already loaded.
  3. Create project with name that has already been given to another project.
- Load Project
  1. Load project in cleared system environment with valid name (simple case).
  2. Load project when there is one already loaded.
  3. Load project with invalid (does not exist) name.
  4. Load project with no drawable object files but has essential objects files.
  5. Load project with no essential objects files but has drawables.

- Save Project As
    1. Save with new name (simple case).
    2. Save with used name.
- Save Project
    1. Save with local project name (simple case).
- Clear Project Environment
    1. Clear when there is a project present (simple case).
    2. Clear when environment clear already.
- Exit Program
    1. Exit when last project has been saved (simple case).
    2. Exit when there is still a project in environment that has not been saved.

F.1.2.2 Diagram Button

- Show A-0 Diagram
    1. Show the diagram when there is one in the project (simple case).
    2. Show the diagram when one doesn't exist.
- Select by Activity Name
    1. Show diagram that matches name given (simple case).
    2. Show diagram when there is no name match.
- Go To Child Diagram
    1. Show diagram when there is a child diagram to show (simple case).
    2. Show diagram when there is no child.
- Return To Parent
    1. Show diagram when there is a parent diagram to show (simple case).
    2. Show diagram when there is no parent (true only for A-0 diagram).
- Refresh Diagram
    1. Refresh when there is a diagram loaded (simple case).
    2. Refresh when there is no diagram loaded on screen.

F.1.2.3 Dictionary Button

- View All Activities

  1. View when there are activities in project (simple case).
  2. View when there are no activities in project.

- View a Specific Activity

  1. View when there are activities in project and there is a name match (simple case).
  2. View when there are activities in project but there is no name match.
  3. View when there are no activities in project.

- View All Data Elements

  1. View when there are data elements in project (simple case).
  2. View when there are no data elements in project.

- View A Specific Data Element

  1. View when there are data elements in project and there is a name match (simple case).
  2. View when there are data elements in project but there is no name match.
  3. View when there are no data elements in project.

Within the Dictionary Editor Environment:

- Next Page

  1. View next page of dictionary entry when there is a next page (simple case).
  2. View next page when there is no other page for the entry.
  3. View next page after an entry update has been made.

- Next Activity/Data Element

  1. View next dictionary entry when there is a next entry (simple case).
  2. View next dictionary entry when there is no other.

- Exit

1. Exit to main screen.

- Activity/Data Elements Edit Menu Buttons - for each menu entry:

1. Input valid string (within limits).
2. Input zero character string.
3. Input string larger than limits.

F.1.2.4 Output Button

- Output Data Dictionary

1. Output when there are activity and data element entries present in the system.
2. Output when there are no activities or data elements present.

- Output Diagrams

1. Output message.

F.1.2.5 Options Button The operations of this button have not been implemented yet.

F.1.2.6 Utility Button

- Check Syntax

1. Check syntax of project that has essential model objects within (simple case).
2. Check syntax of empty project.

- Refresh Screen

1. Refresh when the windows are blank.
2. Refresh when the windows have text.

F.1.3 Tools Window Buttons

F.1.3.1 Clear Diagram Button

1. Clear when there is a diagram present (simple).
2. Clear when there is no diagram present.

F.1.3.2 Create Button

- Create Diagram

  1. Create A-0 when there are no diagrams present on screen, no diagrams in project environment, project loaded.
  2. Create child diagram when project loaded, one or more diagrams present in environment, and no diagrams on screen.
  3. Create child diagram when project loaded, one or more diagrams present in environment, diagram on screen, and click on box with no child diagram.
  4. Create child diagram when project loaded, one or more diagrams present in environment, diagram on screen, and click on box with child diagram.
  5. Create child diagram when project loaded, one or more diagrams present in environment, diagram on screen, but no click on box.

- Create Box

  1. Create box partially outside window limits.
  2. Create box on top of other box.
  3. Create box on top of line segment.
  4. Create box next to invalid ICOM connection.
  5. Create box next to valid ICOM connection.
  6. Create box next to valid CALLS connection.
  7. Create box in window limits with valid name and number.
  8. Create box in window limits with valid name but invalid number.
  9. Create box in window limits with valid number but not valid name.

- Create Line Segment

  1. Attach to available box.
  2. Attach to available simple turn or junctor connection.
  3. Create without attaching to anything.
  4. Attach to invalid connector.
  5. Create with endpoint outside window limits.
  6. Attach to two simple turns that belong to two different data elements.
  7. Attach to two junctors.

8. Attach to two boxes.

- Create Arrow

    1. Attach to available line end.
    2. Attach to invalid line end (already taken by another arrow, simple turn or junctor).
    3. Create without attaching to anything.
    4. Attach to simple turn or junctor.

- Create Simple Turn/Junctor - The only valid case for these two is to be attached to a line that has an open end. Any other case would make it an invalid operation.
- Create Squiggle

    1. Create left to right and up
    2. Create left to right and down
    3. Create right to left and up
    4. Create right to left and down

- Create Label - automatically created when line segment is created.
- Create Note, Metanote, Footnote, or FEO

    1. Create within window limits
    2. Create outside window limits

F.1.3.3 Delete Button

- Delete Diagram

    1. Delete when there are boxes with children diagrams.
    2. Delete when there are boxes with no children.

- Delete Box

    1. Delete when there are children diagrams.
    2. Delete when there are no children.

- Delete Line Segment

    1. Delete when attached to simple turn.

2. Delete when attached to junctor.
3. Delete when attached to box.
4. Delete when attached to arrow.

- Delete Arrow

1. Delete when There is a CALLS record associated.
2. Delete when no CALLS record associated.

- Delete Simple Turn

1. Delete when attached to two lines.
2. Delete when attached to one line.

- Delete Junctor

1. Delete when attached to more than one line.
2. Delete when attached to one line.

- Delete Squiggle - Since not attached to anything, there are no invalid conditions tested, except for not finding a squiggle in the first place.
- Delete Label - automatically deleted when line segment is deleted.
- Delete Note, Metanote, Footnote, or FEO

1. Delete when found.
2. Delete when not found.

F.1.3.4 Update Button

- Update Diagram, Note, Metanote, Footnote, or FEO - The update button test ensures that a valid character string is input. Zero characters or a blank string is not acceptable.

F.1.3.5 Move Button

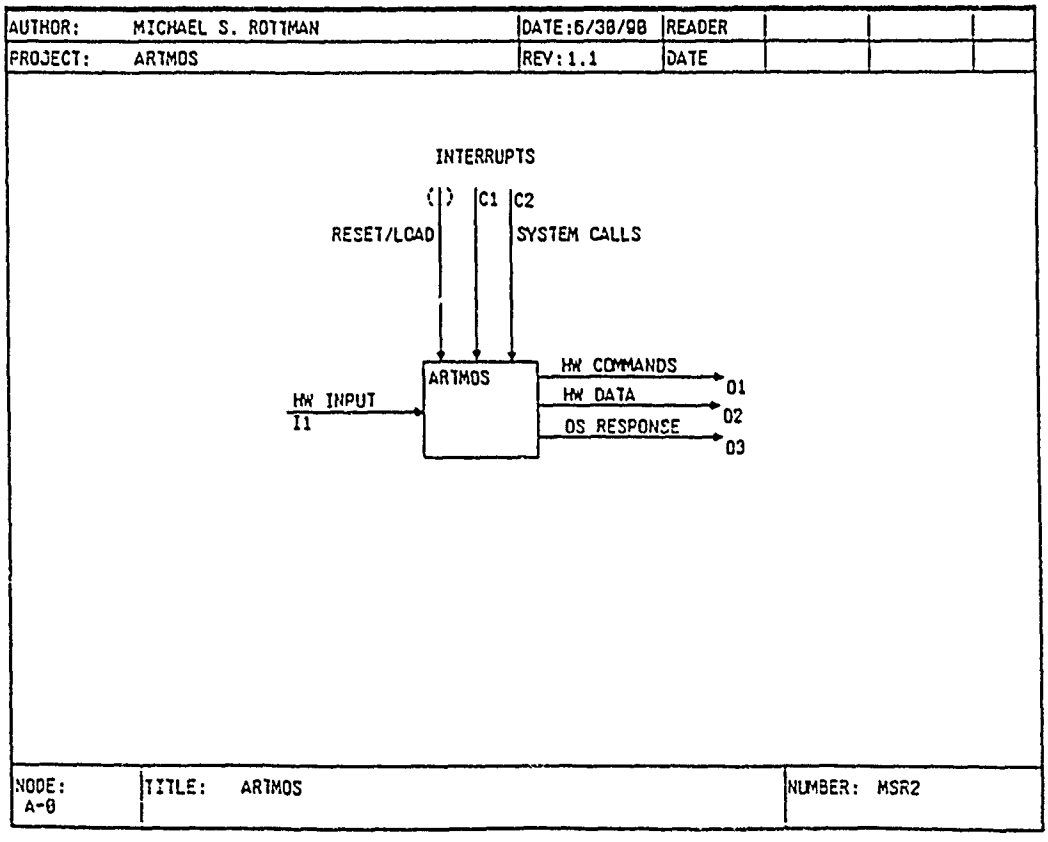
- Move Label, Note, Metanote, Footnote, or FEO - Ensure that the object was moved to a location within the drawing window limits.

F.2 Integration Test Case

The integration test case is based on an $IDEF_0$ project description developed by Rottman (22). This project description is comprehensive in the types of objects and relationships it contains.

A-0 ARTMOS

This is the SADT "context diagram," which shows an overview of the entire operating system as it relates to the "outside world." The OS, once triggered, receives interrupts, system calls, and hardware input and produces hardware commands, data and OS response.



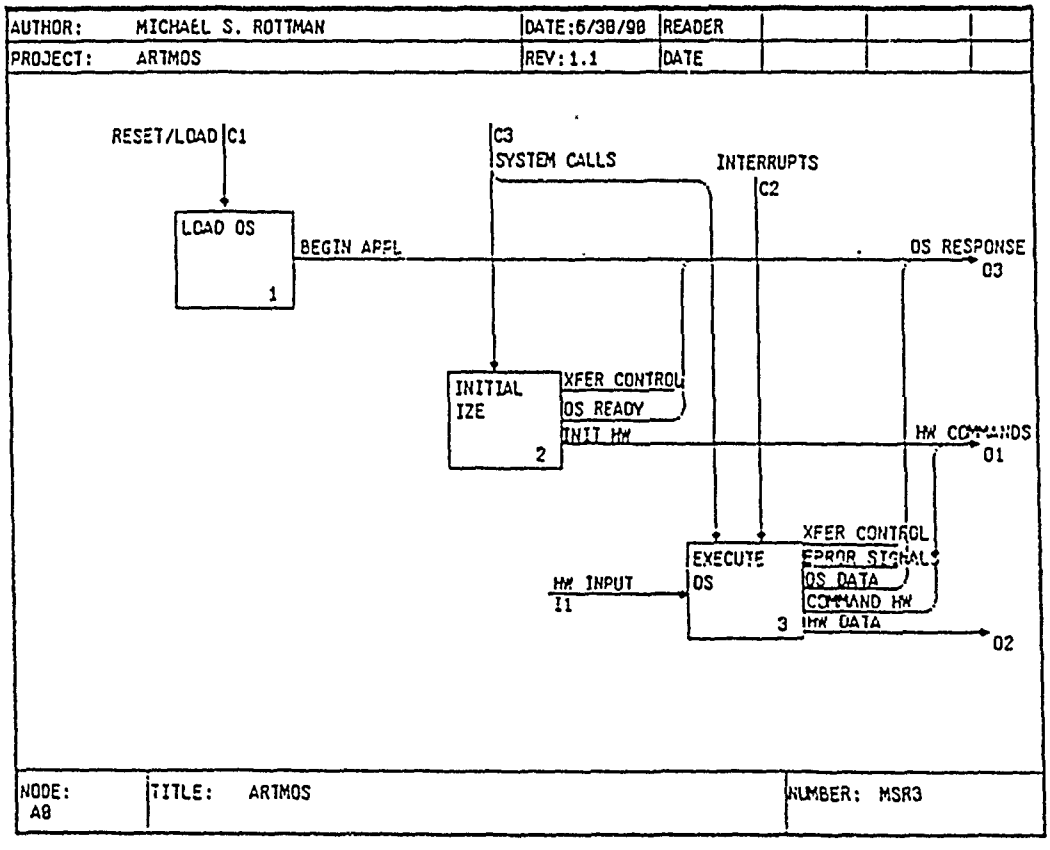
A0 ARTMOS

Abstract: This diagram decomposes the view presented in the A-0 diagram. The three activities are loading the OS, initializing the system, and executing the OS.

A1 This activity loads the actual OS and application software. When the code has been loaded, the application software is activated. The complexity of this activity will depend on the specific needs of the target hardware.

A2 This activity performs the necessary actions to place the OS and system hardware in a known state prior to execution of the main application and OS code. This involves initializing hardware resources and creating data structures, both on the local processor and for the system itself.

A3 This activity is the actual "operating system." It performs the multitasking of application tasks and handles system calls and interrupts.



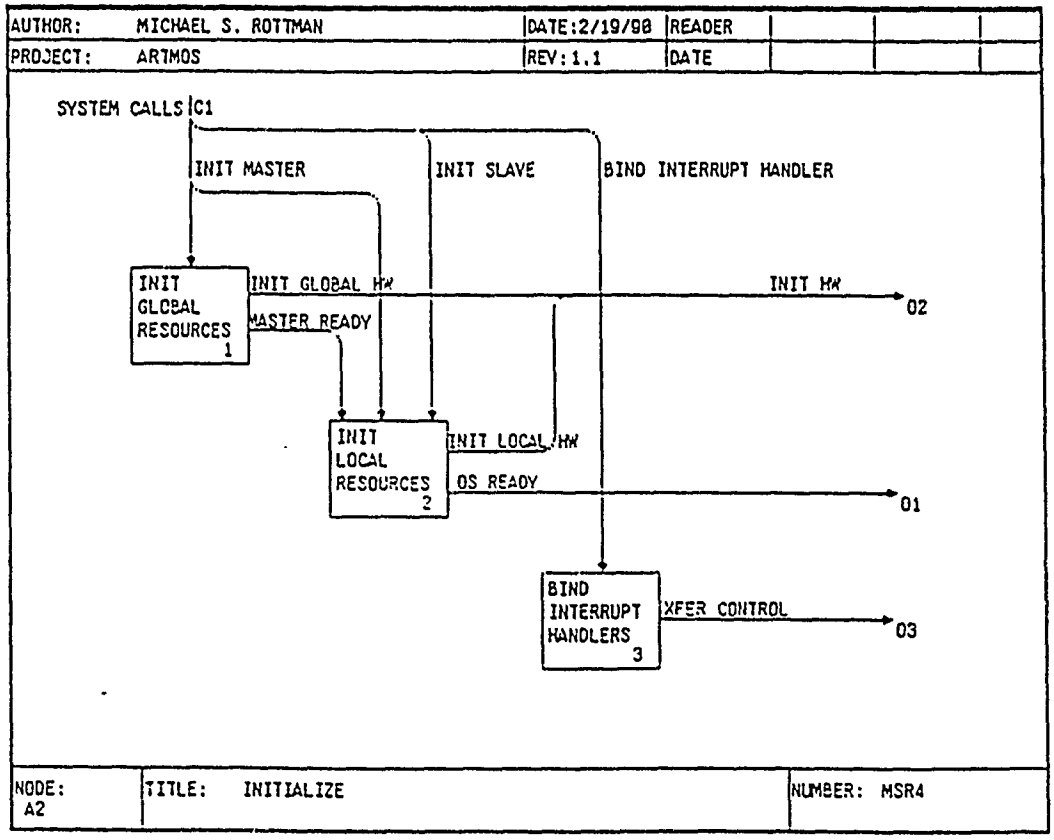
A2 INITIALIZE

Abstract: This diagram shows that initialization consists of initializing both the global system resources and the local processor resources, as well as binding any application-provided interrupt handlers.

A21 this activity initializes all hardware and software resources shared by the processors to place them in the desired state. One processor is designated as the master processor, and the master is the only processor to trigger this activity.

A22 This activity initializes all hardware and software resources local to a processor, and takes place on all processors (including the master). This activity cannot occur until A21 INIT GLOBAL RESOURCES has completed.

A23 This activity maps application-provided interrupt handlers to local hardware interrupts. This gives the application designer the flexibility to supply his own handlers if necessary.



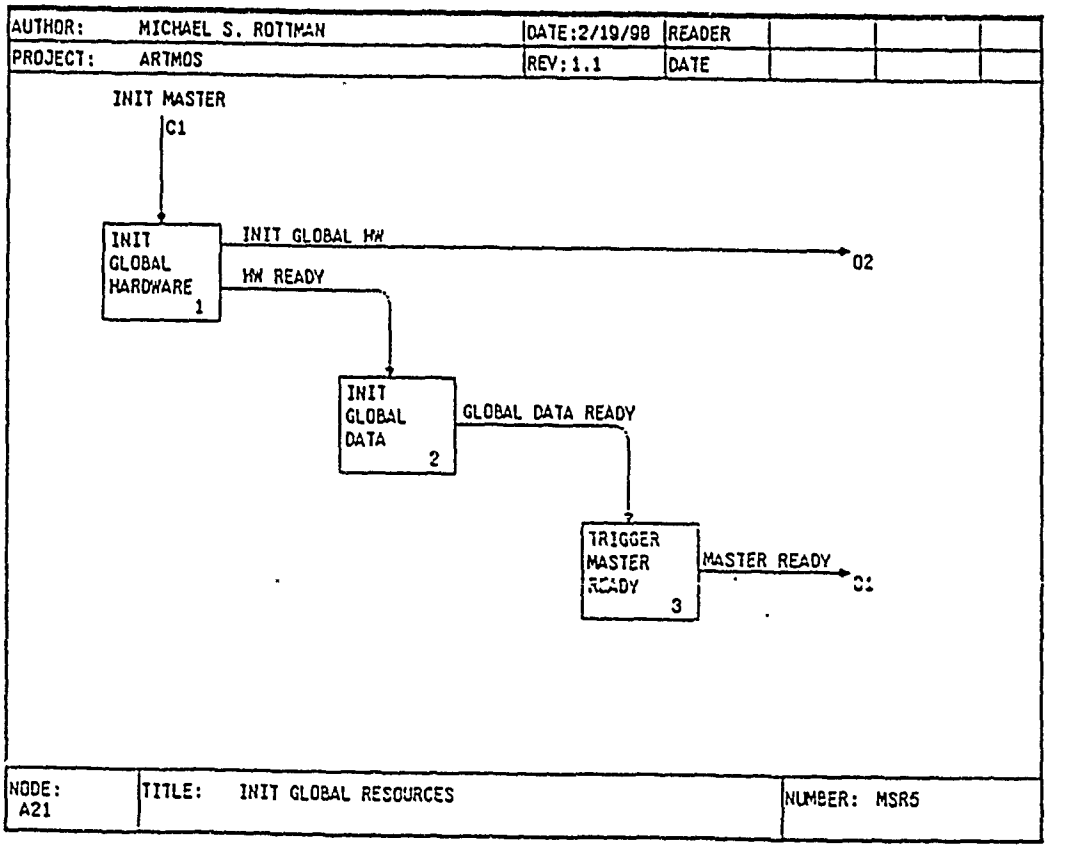
A21 INIT GLOBAL RESOURCES

Abstract: This diagram decomposes the process of initializing global or system resources into the initialization of all hardware devices and creation of any data structures shared by the multiple processors. This activity occurs on only one processor, the master.

A211 This activity "programs" the hardware devices and mechanisms needed by the entire system to place them in a known state prior to execution. An example of global hardware could be the communications network linking the processors.

A212 This activity creates and initializes any dynamic data structures used by the processors. One examples of global data structures could be a table showing the status of the different processors or number of errors by each processor.

A213 This activity serves to notify the other processors that global initialization has completed, and they can proceed with local initialization.



A22 INIT LOCAL RESOURCES

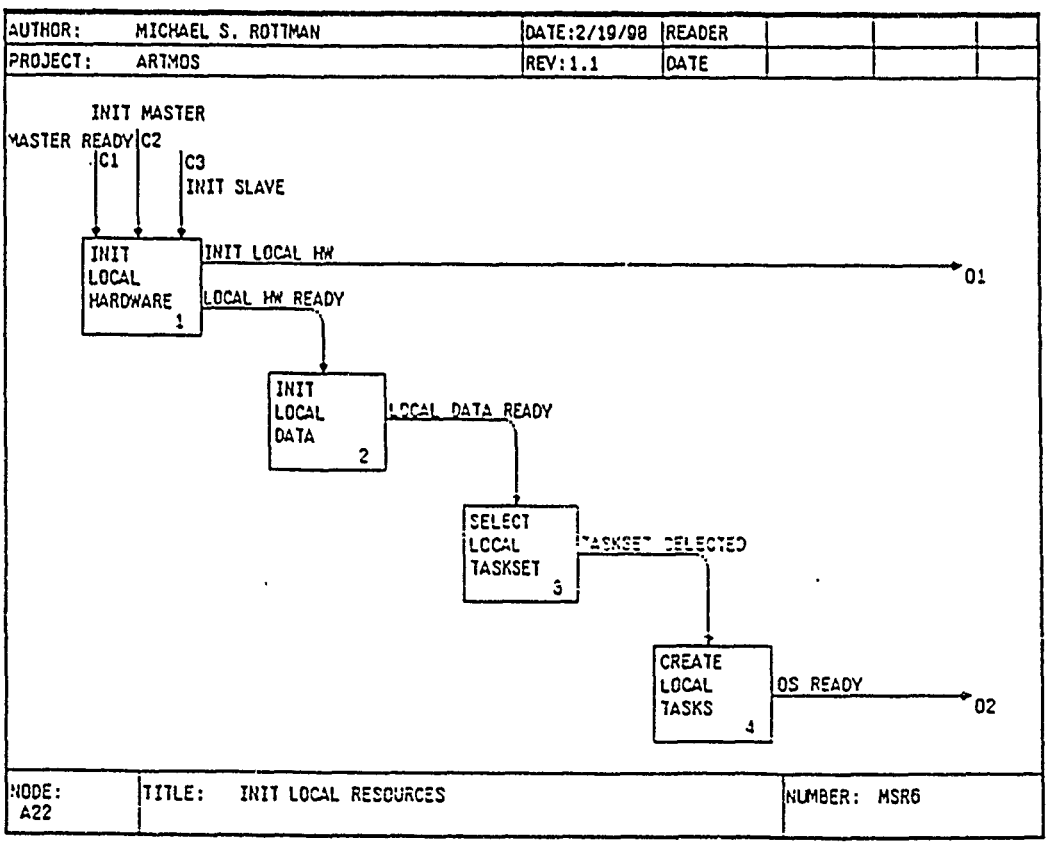
Abstract: This diagram decomposes the process of initializing local resources into the initialization of all hardware devices and creation of any data structures needed by the local processors. This activity occurs on all processors, including the master.

A221 This activity "programs" the hardware devices and mechanisms needed by the local processor to place them in a known state prior to execution. Examples of local hardware devices include timers, interrupt controllers, and input/output devices.

A222 This activity creates and initializes any dynamic data structures used by the local processor. Examples include task management queues or lists and status tables.

A223 this activity evaluates the system configuration and selects a set of tasks for the local processor to manage. The task sets are static: they do not change unless the configuration changes, and processors in the same configuration always get the same set of tasks.

A224 This activity "creates" the local tasks based on the selected task set. A TCB is obtained and initialized for each task in the task set, then loaded into the appropriate task management list. When all tasks are created, OS readiness is signaled.



A224 CREATE LOCAL TASKS

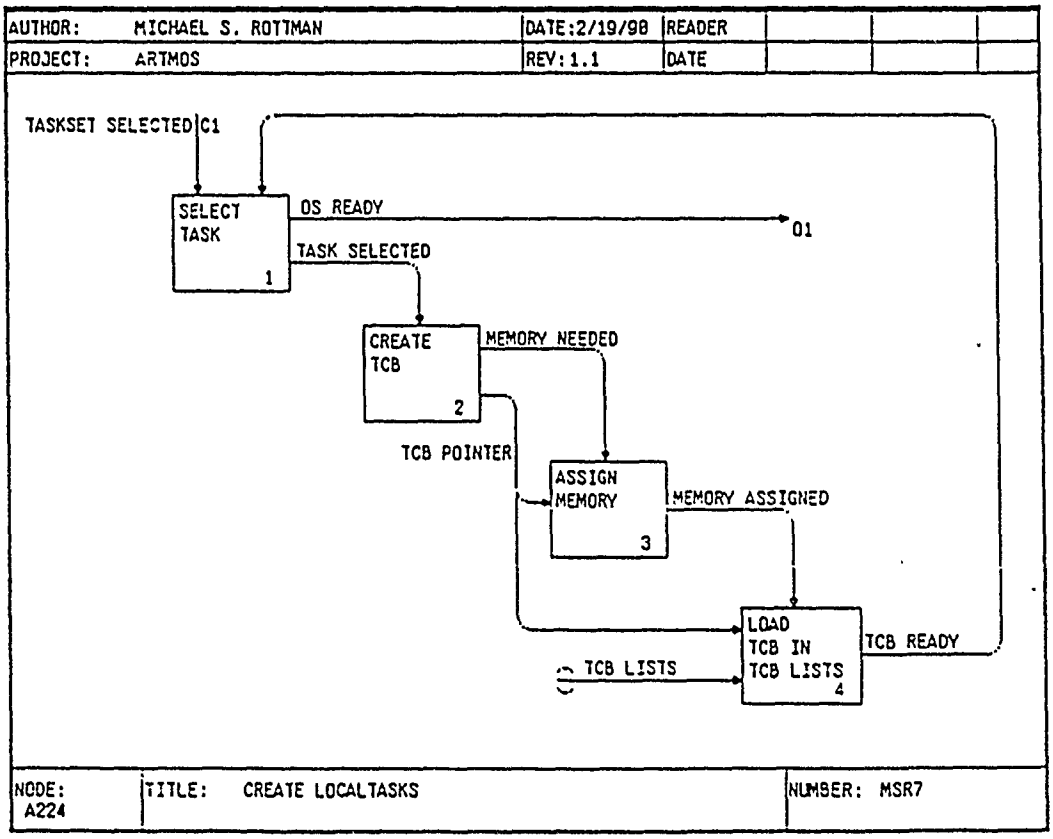
Abstract: This diagrams shows how application tasks are "created" and initialized. For each application task assigned to the local processor, this activity finds an available TCB, allocates the amount of memory the task needs, and loads the TCB into the appropriate task management list.

A2241 This activity controls the creation of tasks. It sequences through the set of tasks, selecting each task for creation and passing task information to the next activity.

A2242 This activity gets an unused TCB and initializes it with information specific to the task selected by A2241.

A2243 This activity is responsible for memory management. Memory is allocated to the selected task according to its needs.

A2244 This activity loads the initialized TCB into the appropriate task management list (ready or sleeping).



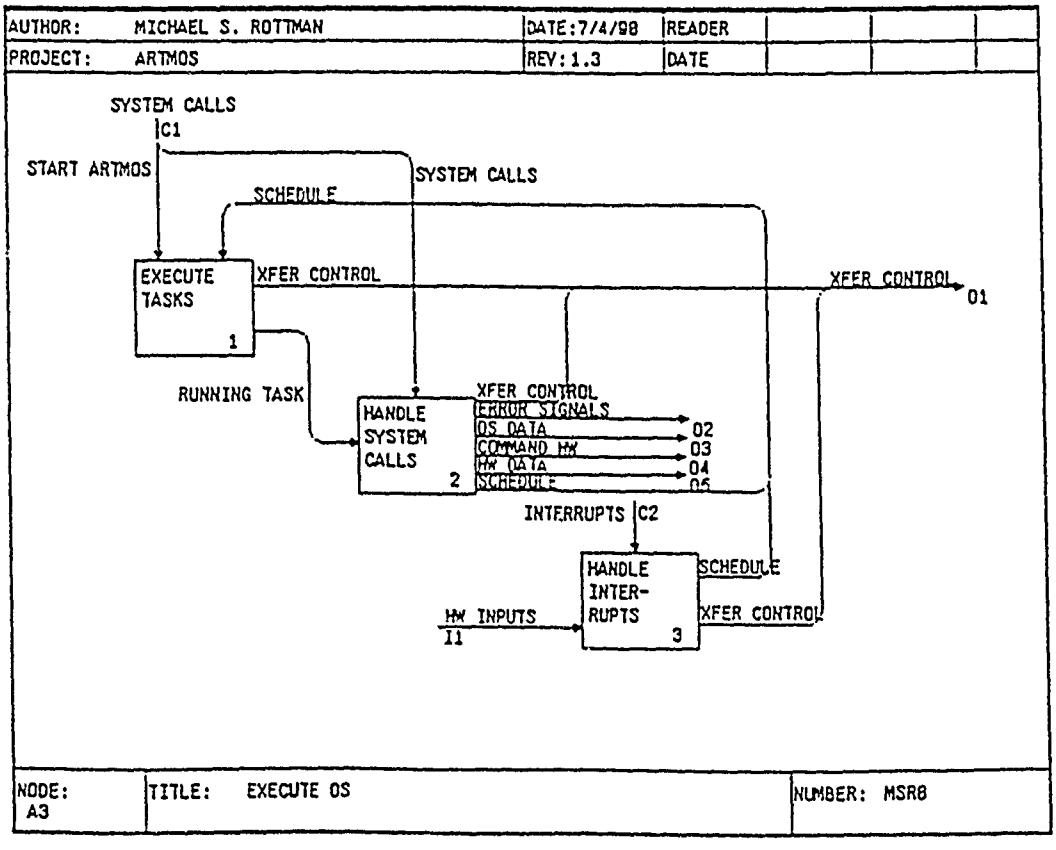
A3 EXECUTE OS

Abstract: EXECUTE OS is the heart of the ARTMOS. It is responsible for the multi-tasking of application tasks and management of system resources. Other major functions of EXECUTE OS are the handling of system calls from the application tasks and interrupts from the hardware.

A31 This activity is responsible for scheduling ready tasks to the local processor for execution. This activity is triggered by the START ARTMOS system call, or whenever a system call or interrupt has changed the set of ready tasks or caused the currently running task to release the processor. In the first case, A31 checks the list of ready tasks. If any have a higher priority than the currently running task, the running task is preempted. Then (or when a task has released the processor) A31 selects a task from those ready to run, restores the task's context (state), and passes control to the application task.

A32 This activity handles all application system calls for task, communications, semaphore, interrupt, and time management. Commands to the hardware and responses to the application are generated as needed.

A33 This activity handles any local processor interrupts, and to a large degree is dependent upon the specific target hardware. Some systems will need more or different handlers than others. A clock interrupt is assumed.



A31 EXECUTE TASKS

Abstract: This diagram shows the steps required for the execution of application tasks. When the set of ready tasks has changed, the ready list is evaluated to determine if a higher priority task is available to preempt the running task. If so, or if the current task has released the processor, a task must be selected from the set of ready tasks, its context must be restored, and control must be passed to the application task.

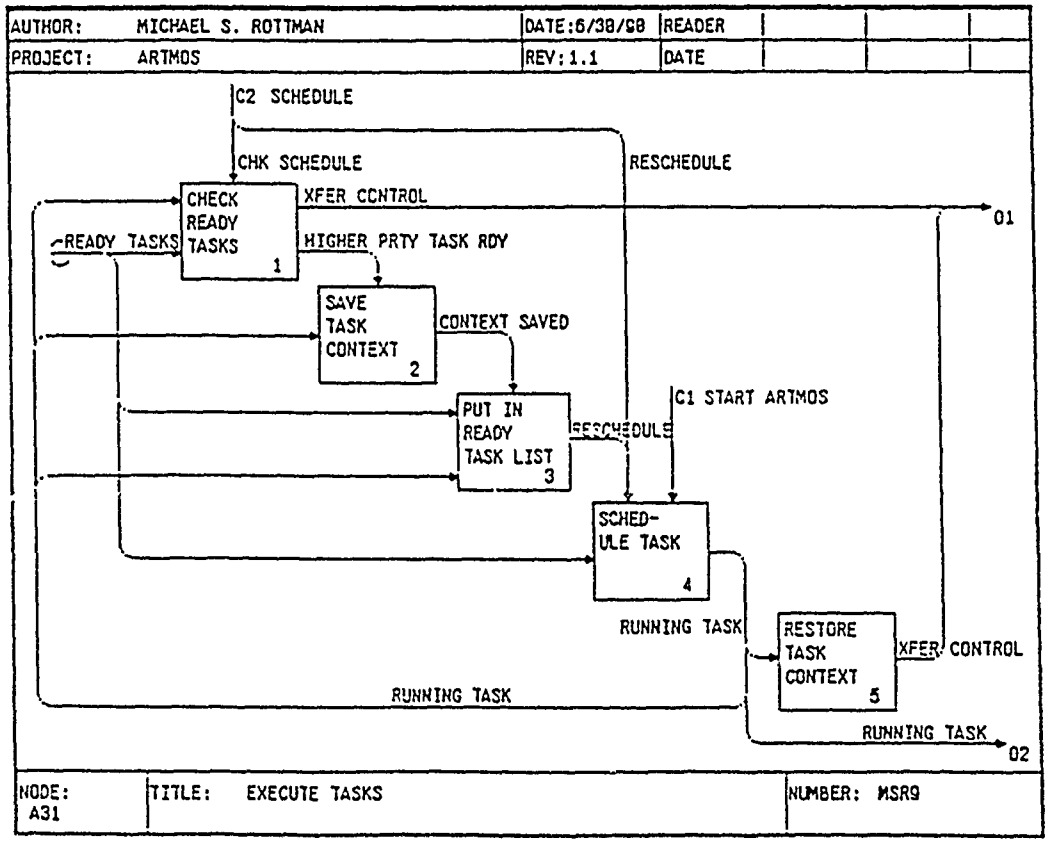
A311 This activity checks the ready list to see if any ready task has a higher priority than the running task. If so, the running task is preempted.

A312 This activity saves the context of the running task.

A313 This activity puts the running task back in the ready task list.

A314 This activity performs the actual selection of the ready task to execute. A specific scheduling algorithm will be identified in the design phase.

A315 When a task is switched out of the processor, the exact state of the processor (the context) is saved. The next time the task executes, this activity restores the context, so that the task can continue as if never interrupted. Task context may include status flags, data registers, and so on.



A32 HANDLE SYSTEM CALLS

Abstract: This diagram decomposes the process of handling system calls into functional areas based on the type of call. The system calls fall into the categories of task management, communications, semaphore management, interrupt management, and time management. Many of the calls prompt some hardware command or response from the OS, and all either return control to the application task or trigger a scheduling operation.

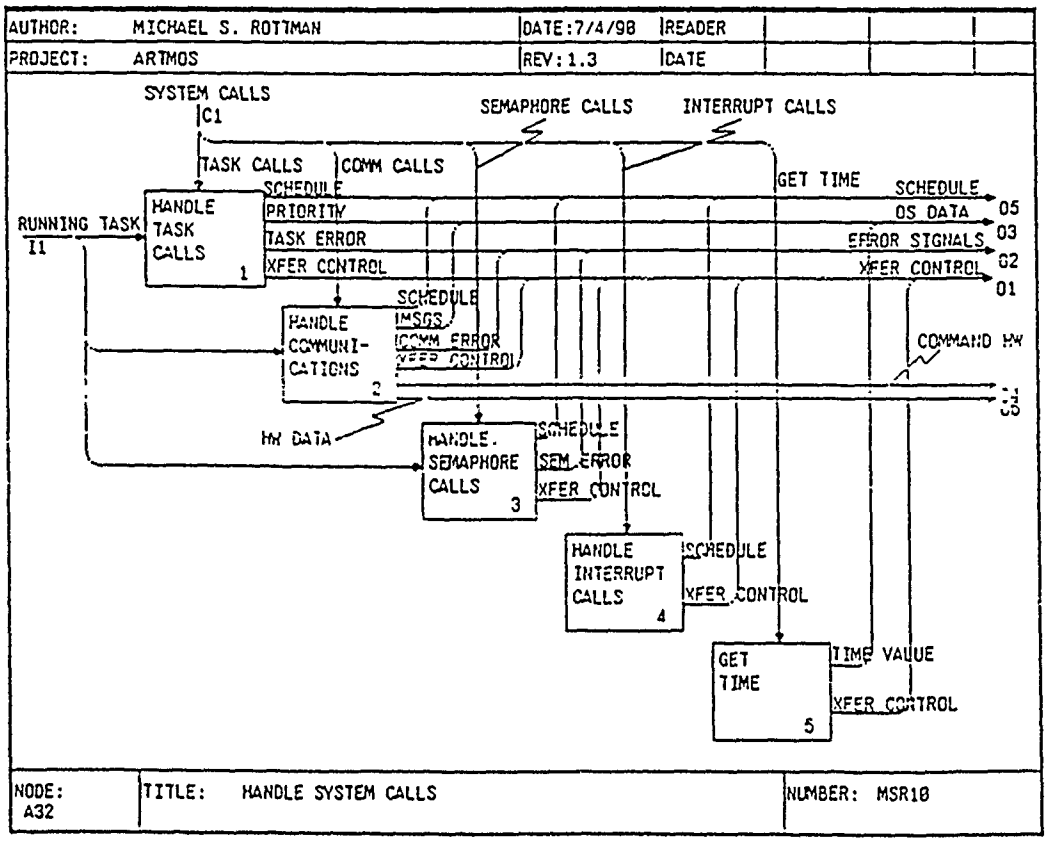
A321 This activity handles requests from a task for information about itself or to change its execution in some way. A task can terminate or delay its execution, change its operating mode, or ask its priority.

A322 This activity performs all commuications between tasks. Tasks can synchronously or asynchronously send messages and synchronously receive messages.

A323 This activity provides semaphore capabilities for mutual exclusion on resources local to the processor. No global semaphores are required.

A324 This activity allows application tasks to enable or disable interrupts. In addition, a call handles the return from an interrupt to check if the ready task list has been affected by the interrupt.

A325 This activity supplies the requesting task with the current time value for the local processor.



A321 HANDLE TASK CALLS

Abstract: This depicts the handling of task requests for information or to change their execution in some way. Tasks can only affect their own execution, not that of other tasks. A task can terminate or delay its execution, change its operating mode, or ask its priority.

A3211 This activity allows a task to "kill itself." Termination consists of releasing all resources held by the task and releasing the TCB for other use.

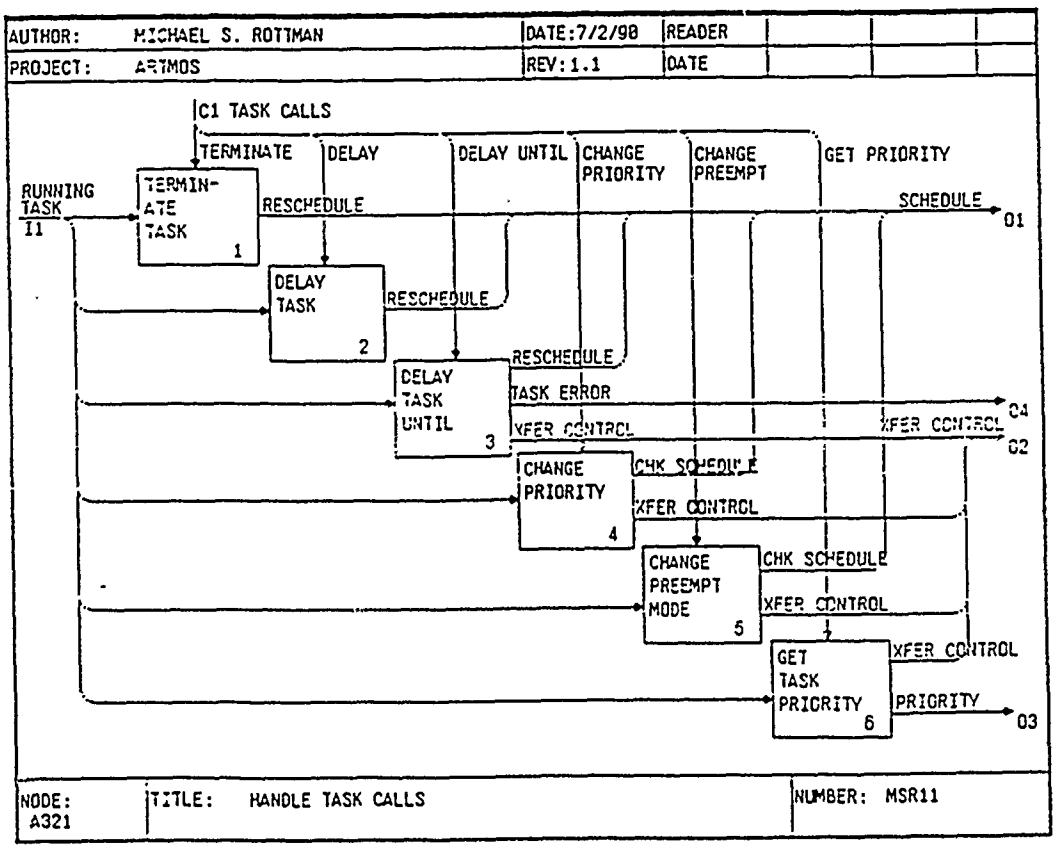
A3212 This activity delays the calling task for a specified time by saving its context and putting it on the sleeping task list or the ready task list, if the delay duration is zero.

A3213 This activity delays the calling task until a specified time by saving its context and putting it on the sleeping task list.

A3214 This activity changes the priority of the calling task. If the priority is lowered, other ready tasks may have a higher priority so EXECUTE TASKS is triggered to check the schedule. If the priority is raised, control is returned to the task.

A3215 This activity enables or disables preemption for the processor. When preemption is disabled, control returns to the calling task. If preemption is enabled, the ready list must be checked to see if a higher priority task is ready.

A3216 This activity returns the priority of the calling task.



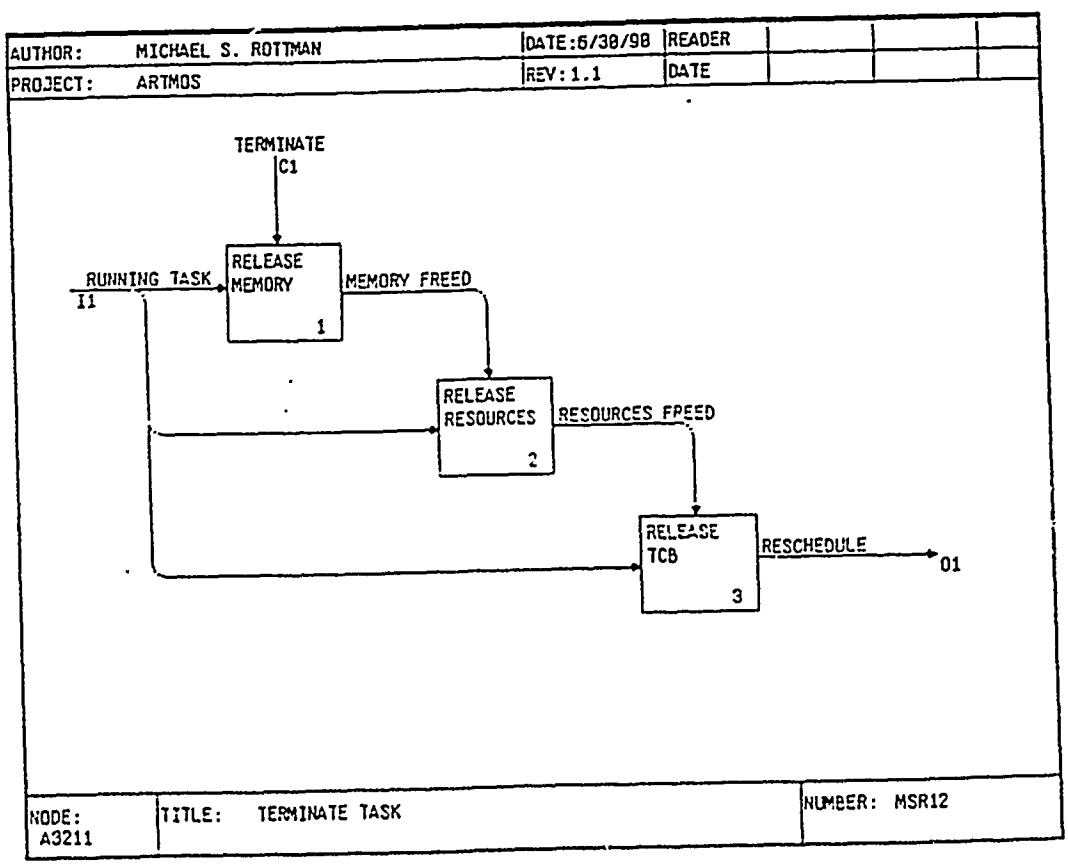
A3211 TERMINATE TASK

Abstract: This diagram shows the components actions required to terminate a task. All memory and resources assigned to a task are released, and the the TCB is freed for other use.

A32111 This activity releases the memory allocated to the task.

A32112 This activity releases any resources allocated to the task, such as guarded by semaphores.

A32113 This activity releases the TCB and triggers a rescheduling operation.



A3212 DELAY TASK

Abstract: This diagram delays the calling task for a specified amount of time. If the delay duration is non-zero, the wakeup time is calculated and set, the task's context is saved, and it is put in the sleeping task list. If the duration is zero, the task is put back in the ready list.

A32121 This activity checks the specified duration to determine whether it is non-zero (whether the task should be saved to the ready task list or the sleeping task list).

A32122 This activity computes the wake up time for the task based on the delay duration and the current time.

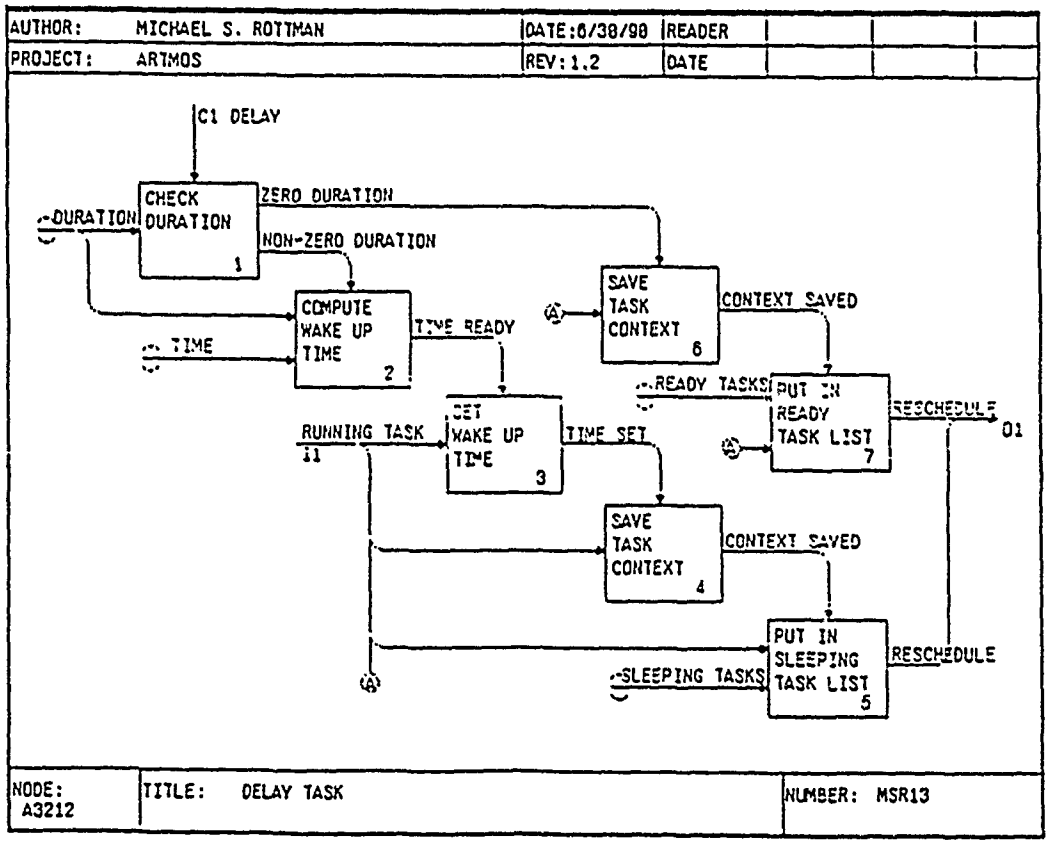
A32123 This activity saves the wake up time in the TCB of the delaying (running) task.

A32124 This activity saves the context of the running task.

A32125 This activity puts the task in the sleeping task list.

A32126 This activity saves the context of the running task.

A32127 This activity puts the task in the ready task list.



A3213 DELAY TASK UNTIL

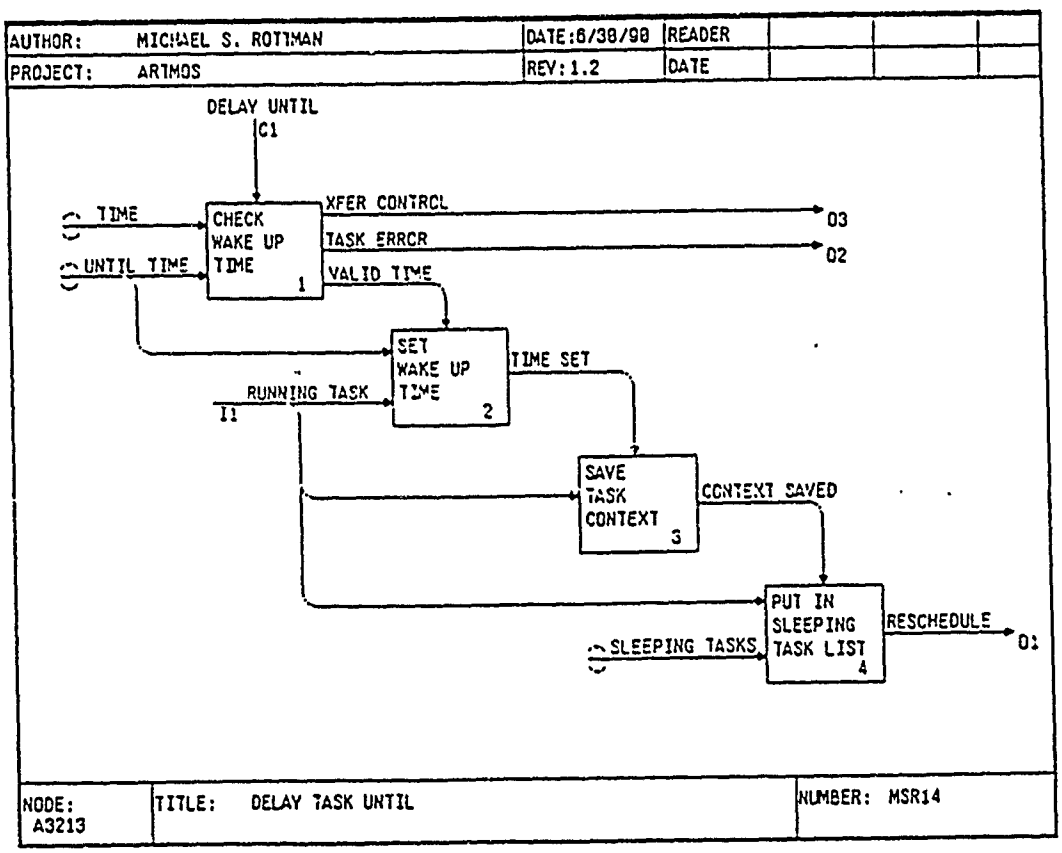
Abstract: This diagram delays the calling task until a specified time. If the wake up time is valid, the time is saved in the TCB and the calling task is switched out.

A32131 This activity checks the specified wake up time to determine if it is valid (later than the current time). If not, control and an error are returned to the calling task.

A32132 This activity saves the wake up time in the TCB of the running task.

A32133 This activity saves the context of the running task.

A32134 This activity puts the task in the sleeping task list.



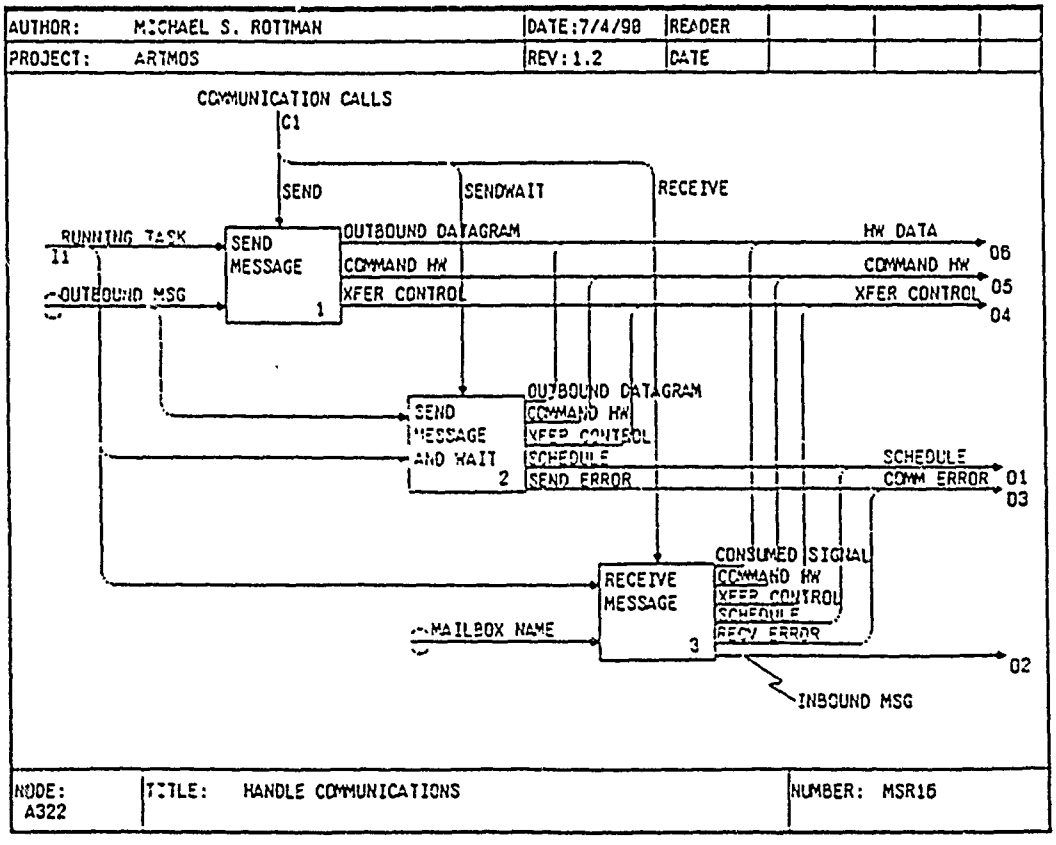
A322 HANDLE COMMUNICATIONS

Abstract: This diagram specifies the intertask communications functionality the ARTMOS must provide to the application. Tasks must be able to send a message to a mailbox synchronously or asynchronously, as well as receive a message when one is available.

A3221 This activity sends a message to a mailbox asynchronously. That is, the message is sent without first checking to see if the previous message has been read (consumed) yet. An outbound message is converted to an outbound datagram for sending, along with appropriate commands to the hardware.

A3222 This activity sends a synchronous message to a mailbox. That is, the message is sent only when the previous message has been consumed. An outbound message is converted to an outbound datagram for sending, along with appropriate commands to the hardware. If the mailbox is not ready within a specified interval, an error code is returned.

A3223 This activity receives a message from a mailbox. If the message is not available (produced) within a specified interval, an error code is returned.

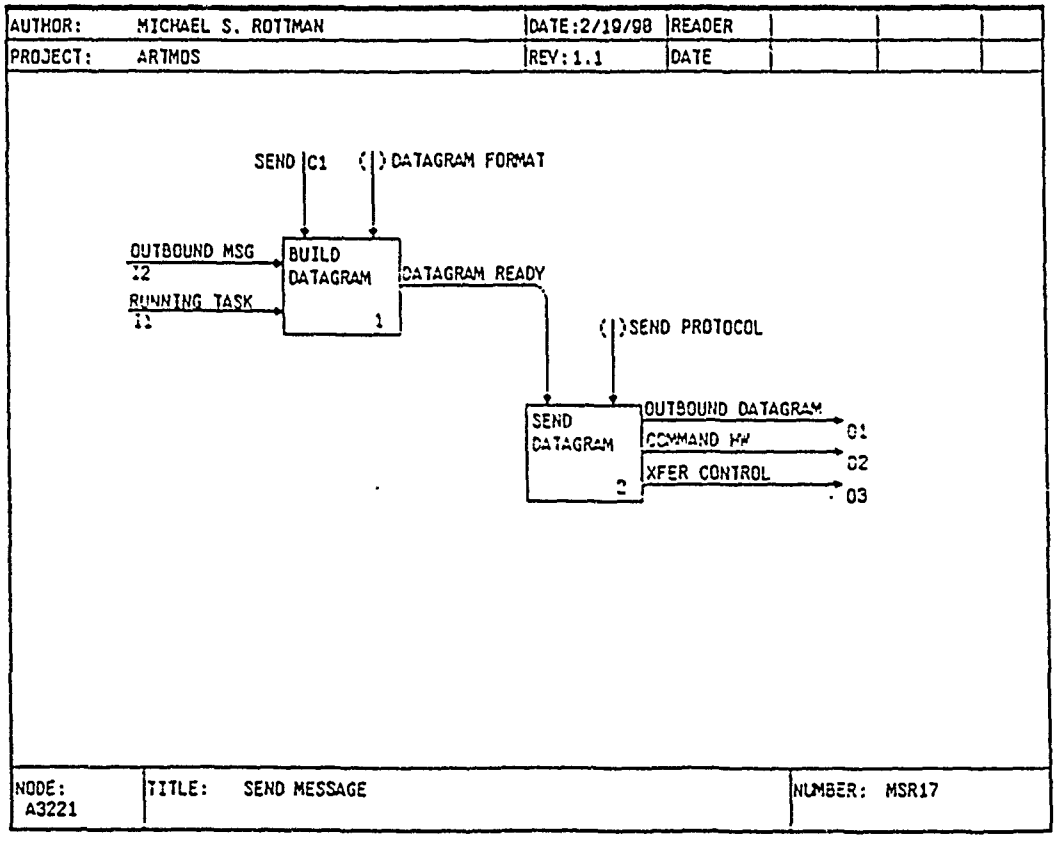


A3221 SEND MESSAGE

Abstract: This diagram shows how a message is sent to a mailbox asynchronously. That is, the message is sent immediately, whether the previous message has been read (consumed) or not. This process involves building an intertask datagram from the outbound message then producing the hardware commands to send the datagram.

A32211 This activity transforms a message into a datagram for sending. The datagram consists of the message, some source information, and some destination information.

A33212 This activity generates the actual hardware commands to send the datagram to the destination mailbox. Specific details of this activity will depend on the needs of the particular target hardware, and so will be discussed in the design phase.



A3222 SEND MESSAGE AND WAIT

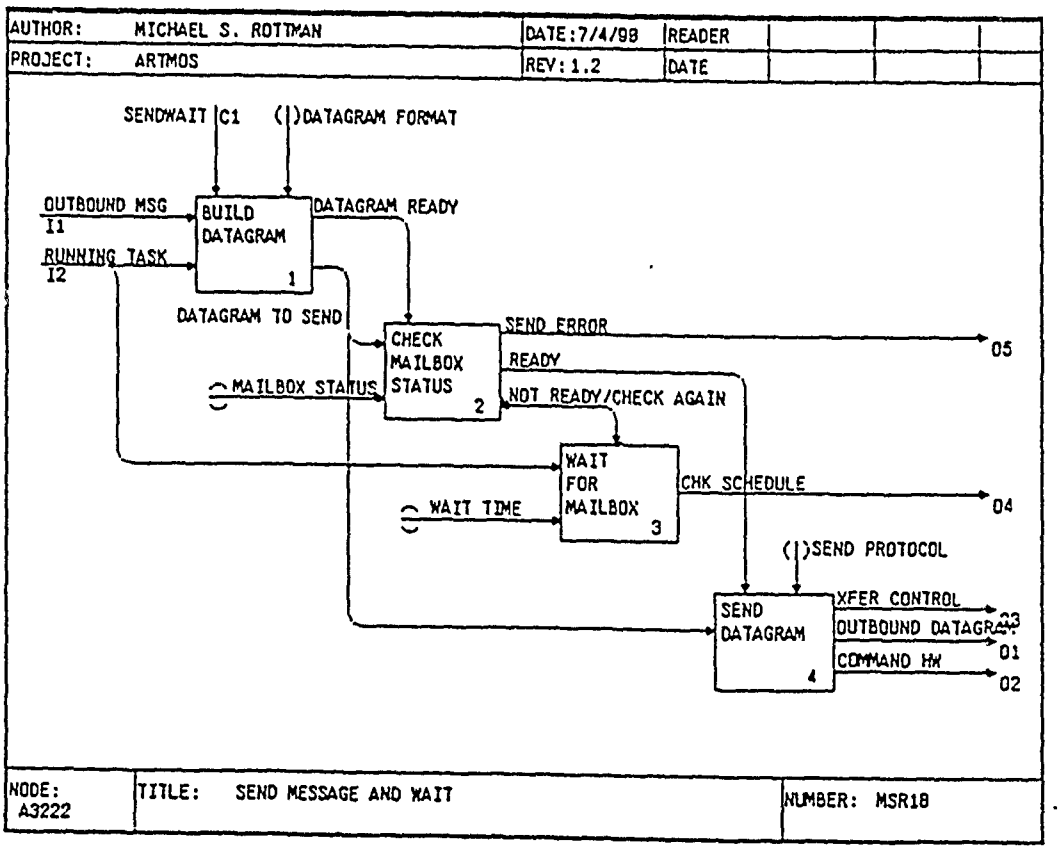
Abstract: This diagram shows how a message is synchronously sent to a mailbox. That is, the message is not sent until the previous message to that mailbox has been consumed, or a specified interval of time has passed. An error code is returned if the message cannot be sent during the interval.

A32221 This activity transforms a message into a datagram for sending. The datagram consists of the message, some source information, and some destination information.

A32222 This activity checks the status of the destination mailbox. If the mailbox is ready (the previous message has been consumed), the datagram is sent. Otherwise, the sending task will wait for a specified amount of time.

A32223 This activity "blocks" the running task for a specified amount of time. No details can be provided concerning the way the task is blocked until the design phase. It could put the task on a blocked list, it could poll, or so on.

A32224 This activity generates the actual hardware commands to send the datagram to the destination mailbox. Specific details of this activity will depend on the needs of the particular target hardware, and so will be discussed in the design phase.



A3223 RECEIVE MESSAGE

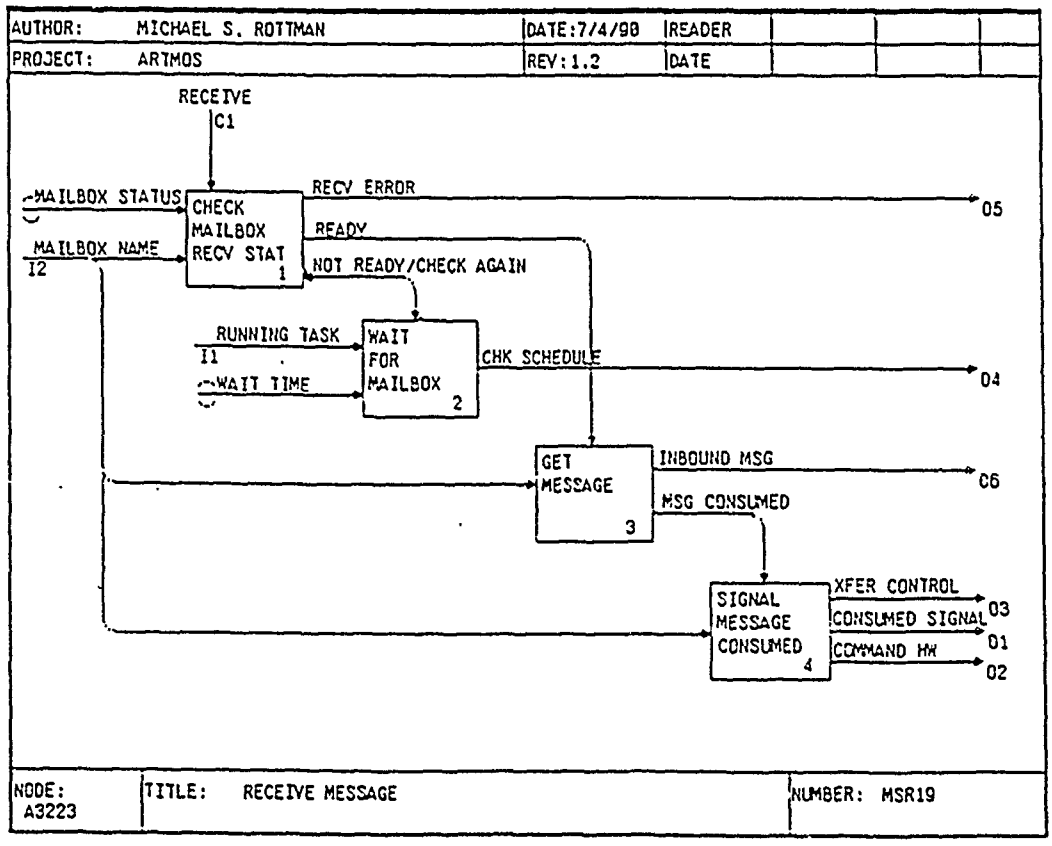
Abstract: This diagram describes the requirements for receiving a message from a mailbox. If the message is available, it is removed from the mailbox and passed to the running task, with a "message consumed" signal sent back to the mailbox. If the message is not available (produced) within a specified interval, an error code is returned.

A32231 This activity checks the status of the source mailbox. If the mailbox is ready (the awaited message has been produced), the message is received. Otherwise, the receiving task will wait for a specified amount of time.

A32232 This activity "blocks" the running task for a specified amount of time. No details can be provided concerning the way the task is blocked until the design phase. It could put the task on a blocked list, it could poll, or so on.

A32233 This activity removes the message from the source mailbox and passes it to the running task.

A32234 This activity generates a "message consumed" signal and the necessary hardware commands to send the signal to the mailbox.

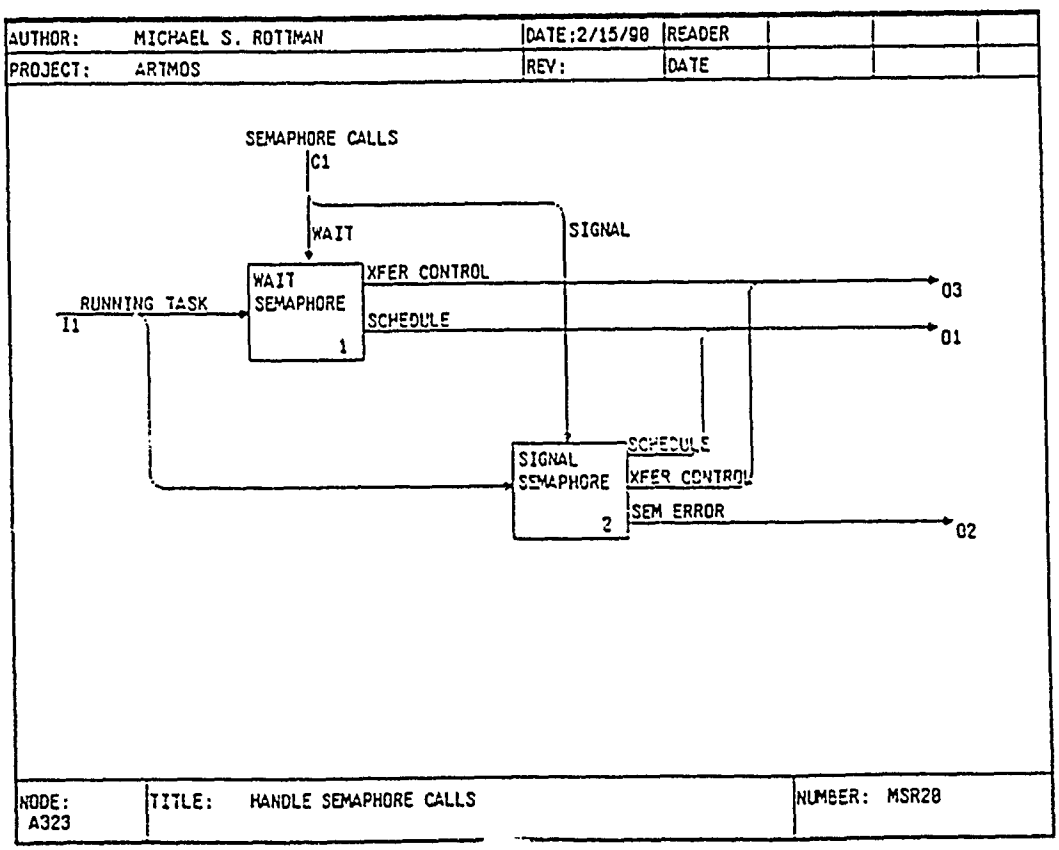


A323 HANDLE SEMAPHORE CALLS

Abstract: This diagram specifies the semaphore management actions the ARTMOS must provide for mutual exclusion of local resources. The application tasks will have the capability to *wait* for a semaphore or *signal* that the semaphore is available.

A3231 This activity checks if the semaphore is available. If it is, it is assigned to the running task and control passes back to the task. Otherwise, the task blocks until the semaphore becomes available.

A3232 This activity releases the semaphore. If tasks are waiting for the semaphore, one is unblocked and added to the ready list. Control returns to the running task if the new task if equal or lower priority.



A3231 WAIT SEMAPHORE

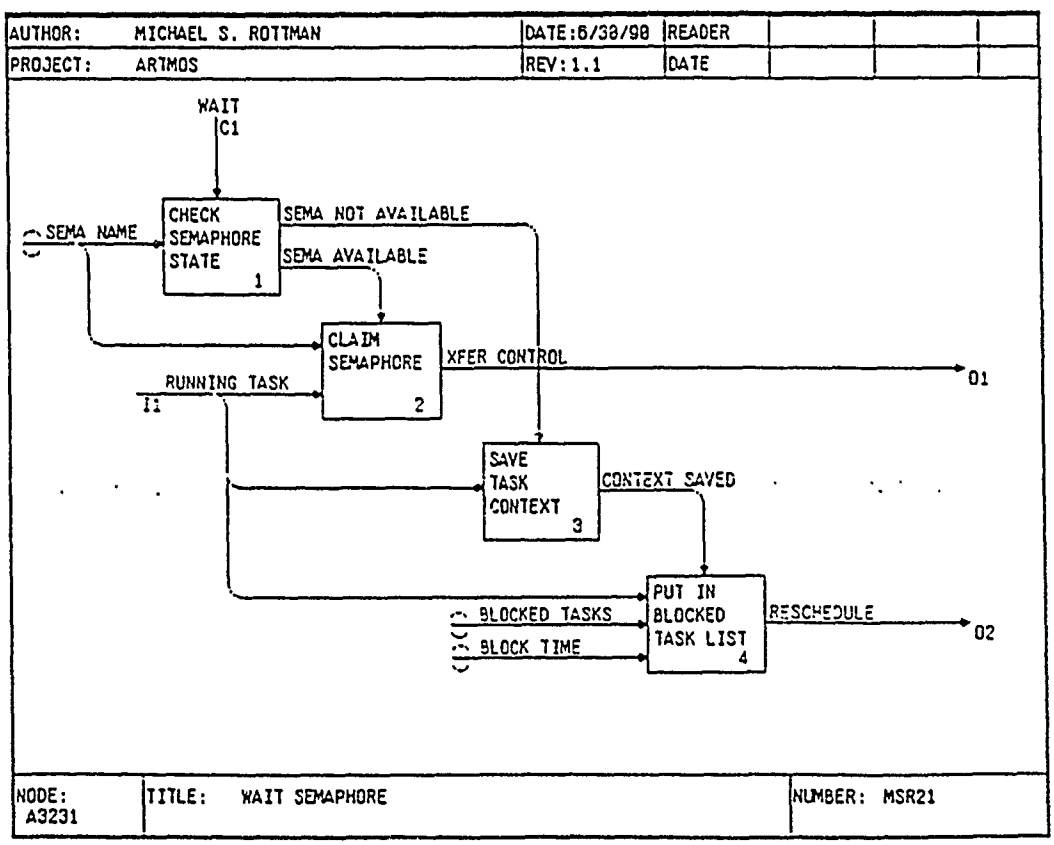
Abstract: This diagram specifies the nature of the semaphore em wait operation. The invoking task continues operation if the semaphore is available, and blocks otherwise.

A32311 This activity checks if the specified semaphore is available. If so, the task is given the semaphore; otherwise, the task blocks.

A32312 This activity assigns the semaphore to the running task and returns control to it.

A32313 This activity saves the context of the running task.

A32314 This activity loads the task into the blocked task list and triggers rescheduling. A maximum wait time is provided as a parameter to the system call.



A3232 SIGNAL SEMAPHORE

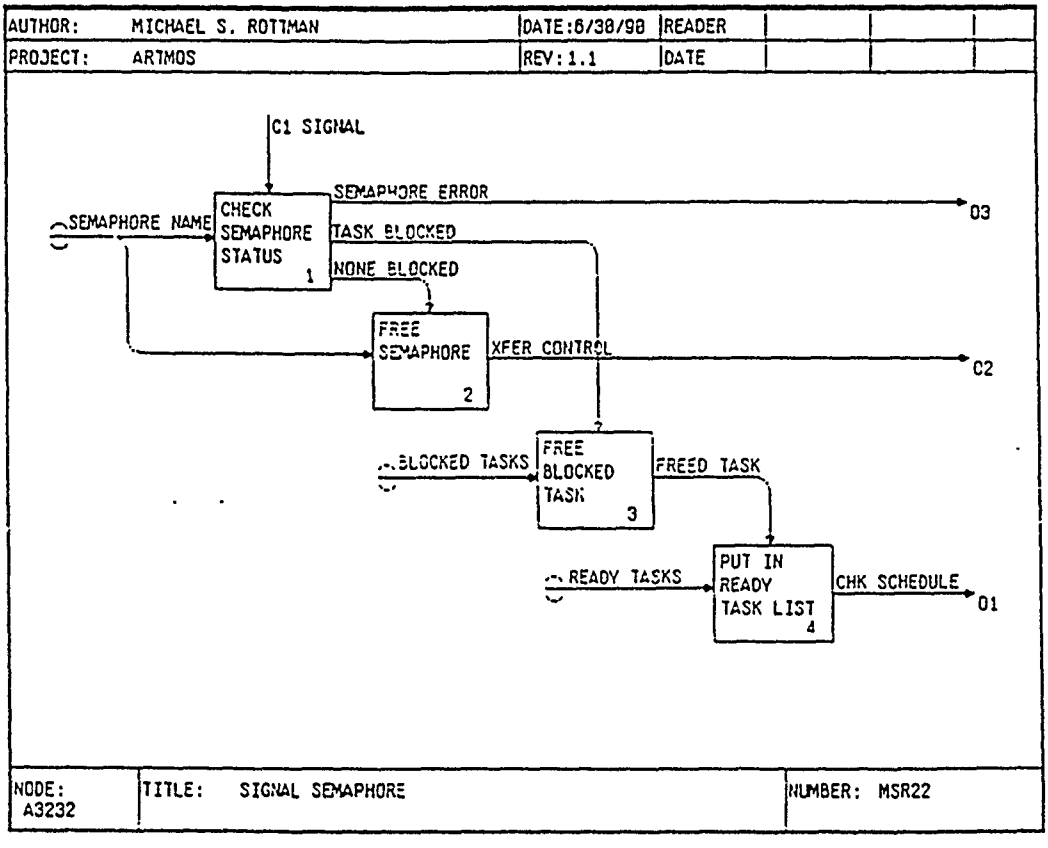
Abstract: This diagram specifies the nature of the semaphore em signal operation. The invoking task releases the semaphore, releasing a (possibly higher priority) blocked task if one exists.

A32321 This activity checks the semaphore status. If the running task does not possess the semaphore, an error code is returned. If any tasks are blocked, one is unblocked; otherwise the semaphore is set to "available" and control returns to the running task.

A32322 This activity sets the semaphore to "available" and control returns to the running task.

A32323 This activity gets the first blocked task from the blocked task list.

A32324 This activity puts the formerly blocked task in the ready task list. EXECUTE TASKS is triggered to check if the new task has a higher priority than the running task.



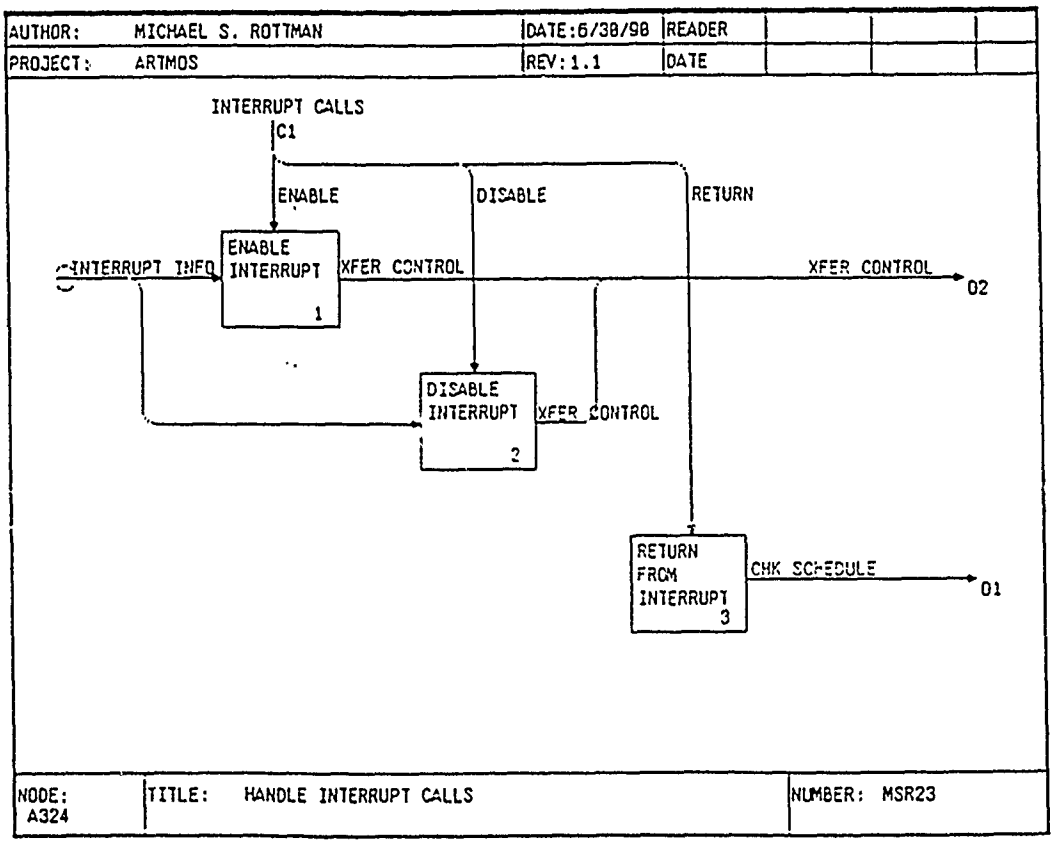
A324 HANDLE INTERRUPT CALLS

Abstract: This diagram specifies the manner in which the application tasks can influence interrupts and how application-provided interrupt handlers must return control to the interrupted task.

A3241 This activity enables the specified interrupt so that it can occur.

A3242 This activity disables the specified interrupt, preventing it from occuring.

A3243 This activity transfers control back to the OS after an application interrupt handler so that the OS can check if a higher priority task was readied by the handler, and preempt the interrupted task if necessary.



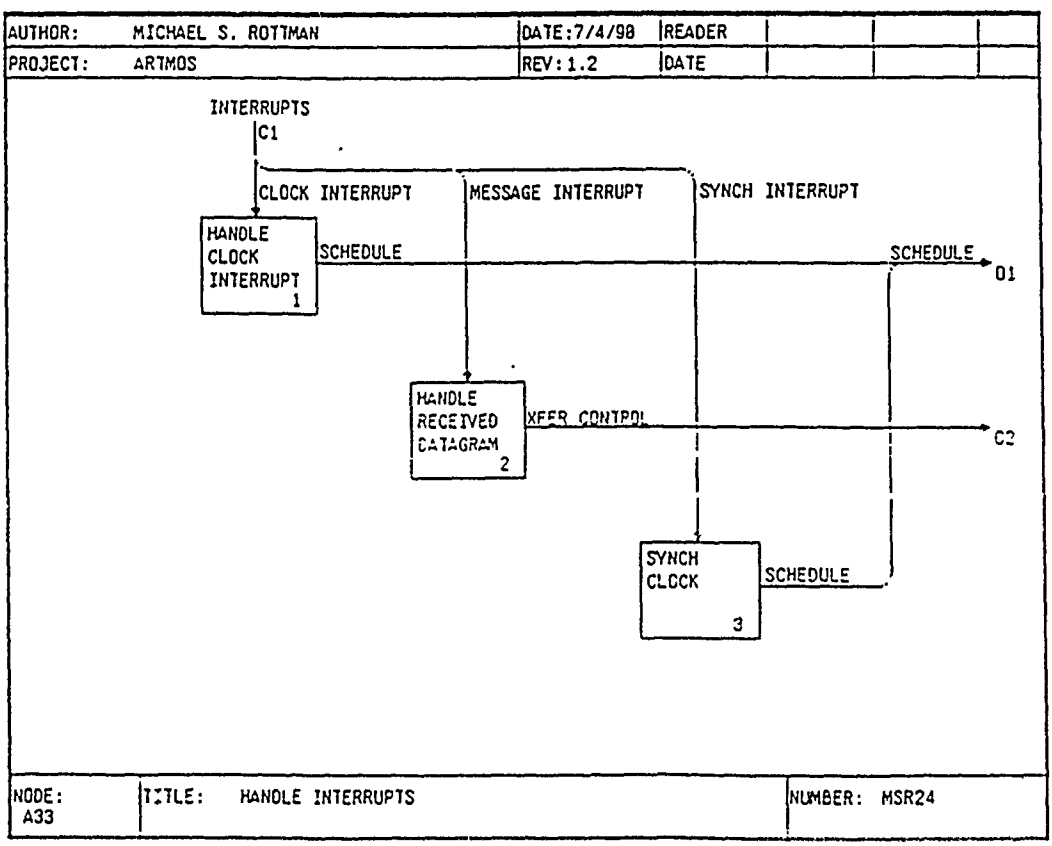
A33 HANDLE INTERRUPTS

Abstract: HANDLE INTERRUPTS shows the different functional requirements for interrupt handling in the ARTMOS. This area by necessity is somewhat sketchy because it depends on the specific interrupt needs of the target hardware. It is assumed that there is a clock interrupt, but receiving a datagram or synchronizing the processor clocks may not be interrupt-triggered. These are shown as interrupts, however, because they happen asynchronously.

A331 This activity increments the local processor clock and wakes up any tasks (blocked or sleeping) that are waiting for the new time.

A332 This activity receives a datagram and posts the message to the appropriate mailbox.

A333 This activity synchronizes the local clock with the rest of the processor clocks, then sees if any tasks are ready to wake up at the new time.



A331 HANDLE CLOCK INTERRUPT

Abstract: This diagram depicts the actions that take place when the local processor clock is incremented.

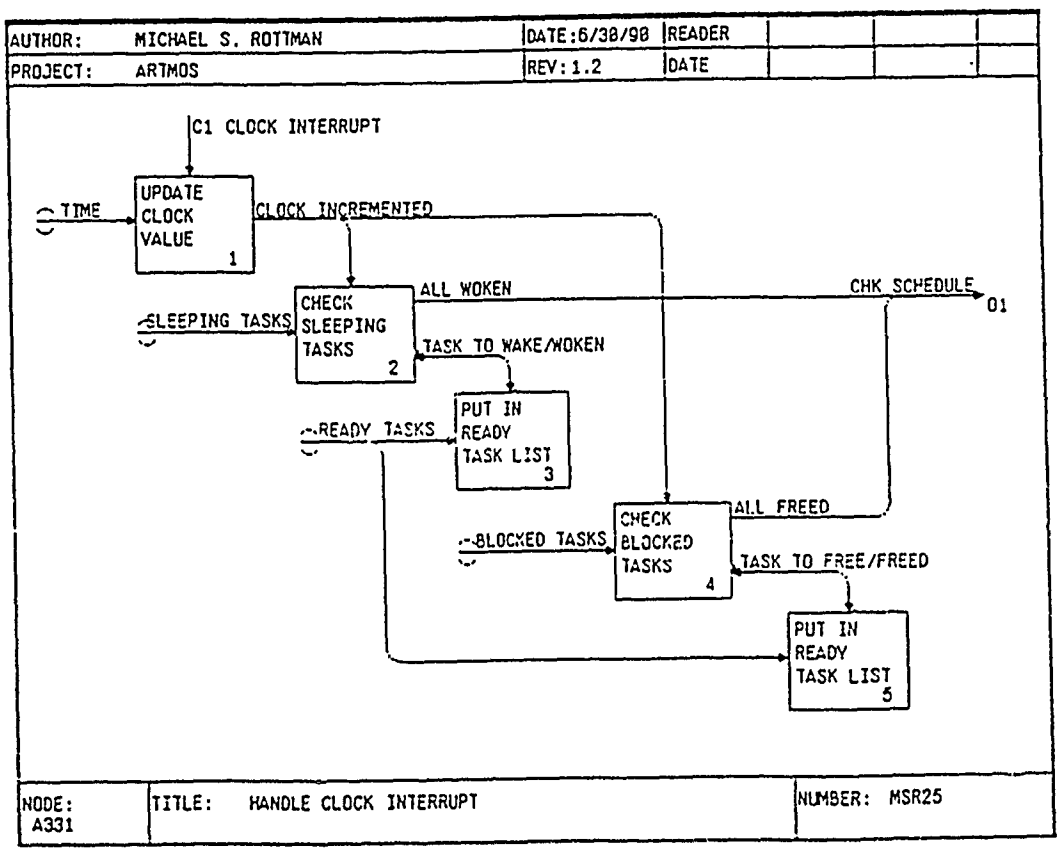
A3311 This activity increments the local clock as appropriate.

A3312 This activity sequences through the sleeping task list, triggerin the awakening of any tasks waiting for the new time.

A3313 This activity puts a task in the ready list.

A3314 This activity sequences through the blocked task list, triggerin the awakening of any tasks waiting for the new time.

A3315 This activity puts a task in the ready list.



A332 HANDLE RECEIVED DATAGRAM

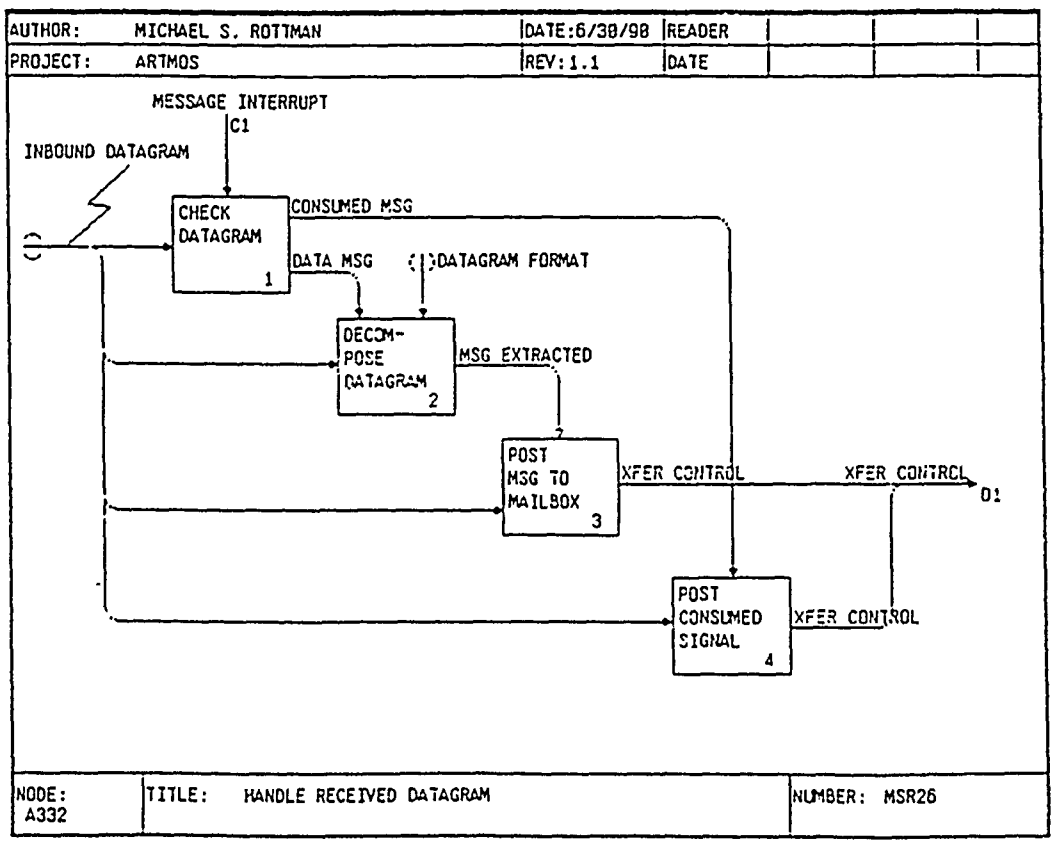
Abstract: This diagram decomposes the process of receiving a datagram. When a datagram arrives at a processor, the message is extracted and placed in the mailbox. If the datagram contains a "message consumed" signal, the appropriate mailbox is "notified."

A3321 This activity checks the arriving datagram to determine if it is a data message or a "message consumed" signal.

A3322 If it is a data message, this activity extracts the message from the datagram.

A3323 This activity loads the extracted message into the specified destination mailbox, then passes control back to the interrupted task.

A3324 If it is a "message consumed" signal, this activity sets the appropriate flag in the mailbox to show that the mailbox is now "available."



A333 SYNCH CLOCK

Abstract: This diagram shows the synchronization of the processor clocks and the actions which must take place as a result of the changing clock value. Until an synchronization algorithm is selected in the design phase, little can be specified. Any waiting tasks whose time has arrived must be awakened, and the interrupted task preempted if any of the new tasks have a higher priority.

A3331 This activity computes an error between the local clock and the other clocks in the system.

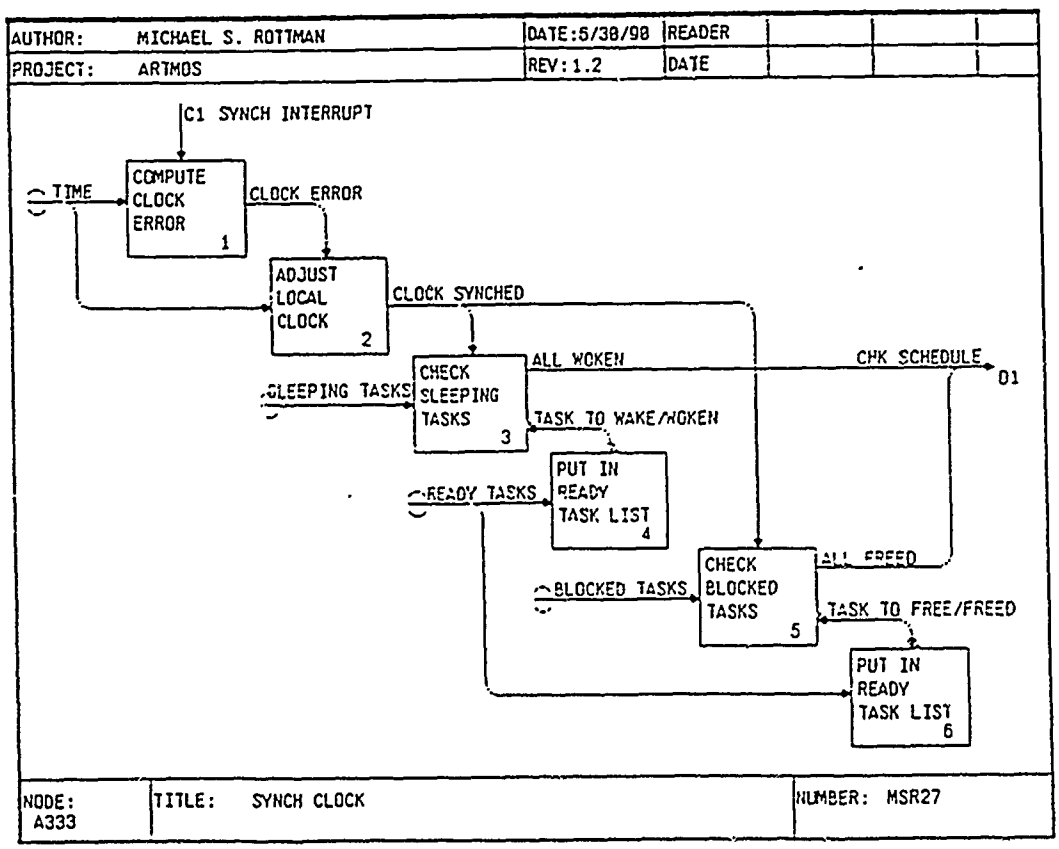
A3332 This activity adjusts the current clock value by the computed error.

A3333 This activity sequences through the sleeping task list, triggerin the awakening of any tasks waiting for the new time.

A3334 This activity puts a task in the ready list.

A3335 This activity sequences through the blocked task list, triggerin the awakening of any tasks waiting for the new time.

A3336 This activity puts a task in the ready list.



Appendix G. *SATool II User Evaluation Form*

CAD-Tool Human-Computer Interface Evaluation[1]

Name (administrative use only): _____.

Estimated time spent with tool/system [:]

Do not write in these spaces
Tool Evaluated:
Class:
Group:
Exper:
First:
ID#:

PLEASE READ BEFORE PROCEEDING:

The following questionnaire is designed to provide user feedback on the human-computer interface of the specified computer-aided design (CAD) tool. Through your responses, we hope to measure your degree of satisfaction with the tool, with primary emphasis on the "user-friendliness" of the human-computer interface.

[1]22 April 1988, U. S. Air Force Institute of Technology, AFIT/ENG(Hartrum)

The questionnaire consists of a set of 11 factors, plus an overall rating. We will determine your satisfaction with the tool based on your response to six adjective pairs used to describe each factor. Each adjective pair has a seven-interval range where you are to indicate your feelings with an "X". Responses placed in the center of the range will indicate that you have no strong feelings one way or the other, *or that you cannot effectively evaluate that given factor.*

Evaluation begin time [:]

1. *System Feedback or Content of the Information Displayed.* The extent to which the system kept you informed about what was going on in the program.

insufficient	☐☐☐☐☐☐☐	sufficient
unclear	☐☐☐☐☐☐☐	clear
useless	☐☐☐☐☐☐☐	useful
bad	☐☐☐☐☐☐☐	good
unsatisfactory	☐☐☐☐☐☐☐	satisfactory

To me this factor is:

unimportant	☐☐☐☐☐☐☐	important

Comments:

2. *Communication.* The methods used to communicate with the tool.

complex	☐☐☐☐☐☐☐	simple
weak	☐☐☐☐☐☐☐	powerful
bad	☐☐☐☐☐☐☐	good
useless	☐☐☐☐☐☐☐	useful
unsatisfactory	☐☐☐☐☐☐☐	satisfactory

To me this factor is:

unimportant	☐☐☐☐☐☐☐	important

Comments:

3. *Error Prevention* Your perception of how well the system prevented user induced errors.

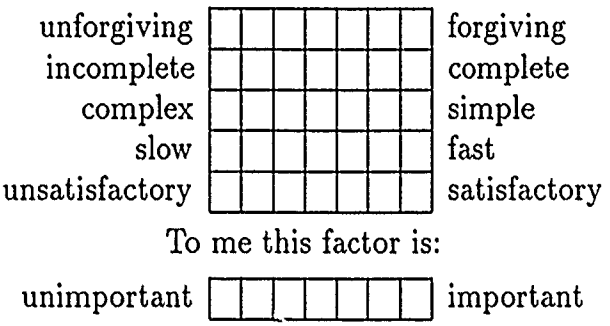
bad	☐☐☐☐☐☐☐	good
insufficient	☐☐☐☐☐☐☐	sufficient
incomplete	☐☐☐☐☐☐☐	complete
low	☐☐☐☐☐☐☐	high
unsatisfactory	☐☐☐☐☐☐☐	satisfactory

To me this factor is:

unimportant	☐☐☐☐☐☐☐	important

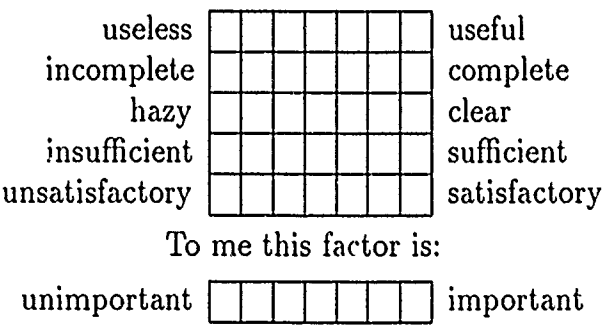
Comments:

4. *Error Recovery.* The extent and ease with which the system allowed you to recover from user induced errors.



Comments:

5. *Documentation.* Your overall perception as to the usefulness of documentation.



Comments:

6. *Expectations.* Your perception as to the services provided by the system based on your expectations.

displeased		pleased
low		high
uncertain		definite
pessimistic		optimistic
unsatisfactory		satisfactory

To me this factor is:

unimportant		important

Comments:

7. *Confidence in the System.* Your feelings of assurance or certainty about the services provided by the system.

low		high
weak		strong
uncertain		definite
bad		good
unsatisfactory		satisfactory

To me this factor is:

unimportant		important

Comments:

8. *Ease of Learning.* Ease with which you were able to learn how to use the system to perform the intended task.

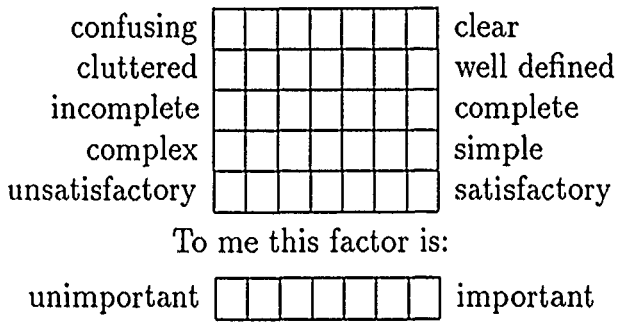
difficult		easy
confusing		clear
complex		simple
slow		fast
unsatisfactory		satisfactory

To me this factor is:

unimportant		important

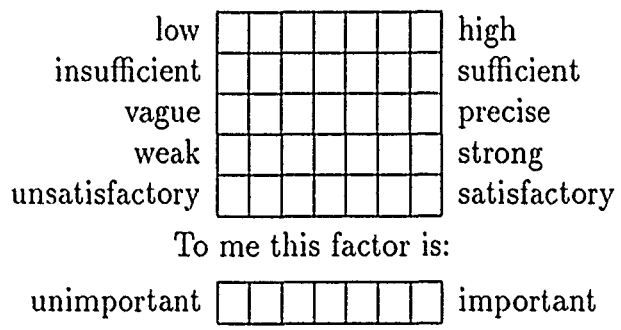
Comments:

9. *Display of Information.* The manner in which both program control and data information were displayed on the screen.



Comments:

10. *Feeling of Control.* Your ability to direct or control the activities performed by the tool.



Comments:

11. *Relevancy or System Usefulness.* Your perception of how useful the system is as an aid to a software developer.

useless	☐☐☐☐☐☐☐	useful
inadequate	☐☐☐☐☐☐☐	adequate
hazy	☐☐☐☐☐☐☐	clear
insufficient	☐☐☐☐☐☐☐	sufficient
unsatisfactory	☐☐☐☐☐☐☐	satisfactory

To me this factor is:

unimportant	☐☐☐☐☐☐☐	important

Comments:

12. *Overall Evaluation of the System.* Your overall satisfaction with the system.

unsatisfied	☐☐☐☐☐☐☐	satisfied

(cont'd)

Comments on the Overall System:

Evaluation end time [:]

Total time spent on evaluation [:]

Thank you for your help.

Appendix H. *IDEF$_0$ Diagram Syntax Review*

The original purpose of the IDEF$_0$ methodology was to produce a function model for manufacturing systems. It provided a structured representation of the functions or *activities* and the information or *data elements* which interrelated those functions (18:1-1). However, the nature of the IDEF$_0$ diagram syntax is such that the IDEF$_0$ methodology can also be used to analyze software requirements (10:1). The software specification products of such an analysis are composed of an IDEF$_0$ graphical decomposition and the corresponding data dictionary (10:1). An IDEF$_0$ system model description is composed of a set of hierarchical diagrams similar in structure to data flow diagrams (DFD) (9:12-21). The graphical decomposition of an IDEF$_0$ diagram is based on activities and data elements in much the same way data flow diagrams are composed of process boxes and data flow arrows (9:12-13). The IDEF$_0$ diagram activities are represented by rectangular boxes and the data elements are represented by lines with arrows (Figure 50).

An activity represents a function or action of the system (Figure 50) (10:7). The activity name always starts with a verb (10:68). Each activity box has a single digit integer placed at the bottom right hand corner and represents the activity number. In the event of more than nine boxes in a diagram, lowercase letters can also be used (10:68). Following is a more detailed description of the hierarchical nature of the IDEF$_0$ diagrams.

Figure 51 through Figure 54 show part of a system description for an elevator control system using the IDEF$_0$ methodology (24:23-25). Figure 51 shows the hierarchical decomposition of this system. The first diagram in the hierarchy is known as the A-0 diagram and it contains only one activity with zero or more data elements (Figure 52). This single activity represents the system being modeled. The activity number given to the system activity is A0. The A0 box is broken down in the next level diagram known as the A0 diagram to show a more detailed description of the system (Figure 53). The number for each activity at this level is equal to the number in the bottom right of each box with an "A" placed in front of it. For example the activity number for the activity "Store Request" is A1. Each of the activity boxes in the second diagram is then broken down to show a more detailed description of its function. Figure 54 shows a breakdown of the first activity box from Figure 53. Here each activity inherits the activity number of its parent activity and adds the single digit assigned to it at the end. For example "Manage Destination" has the activity number A12. This numbering system allows an observer to know how many levels down the diagram hierarchy a particular activity is found. Each digit implies a level.

The data element arrows represent the information that is passed from activity to activity and define the relationships between the activities. How a data element

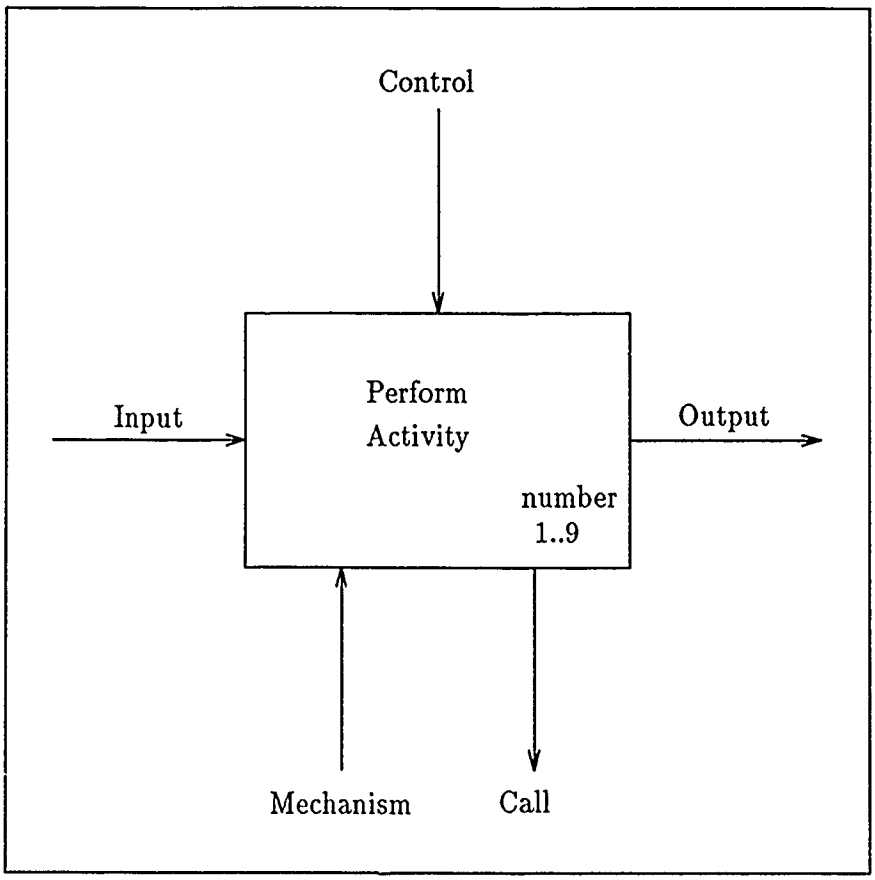


Figure 50. $IDEF_0$ Activity and Data Elements

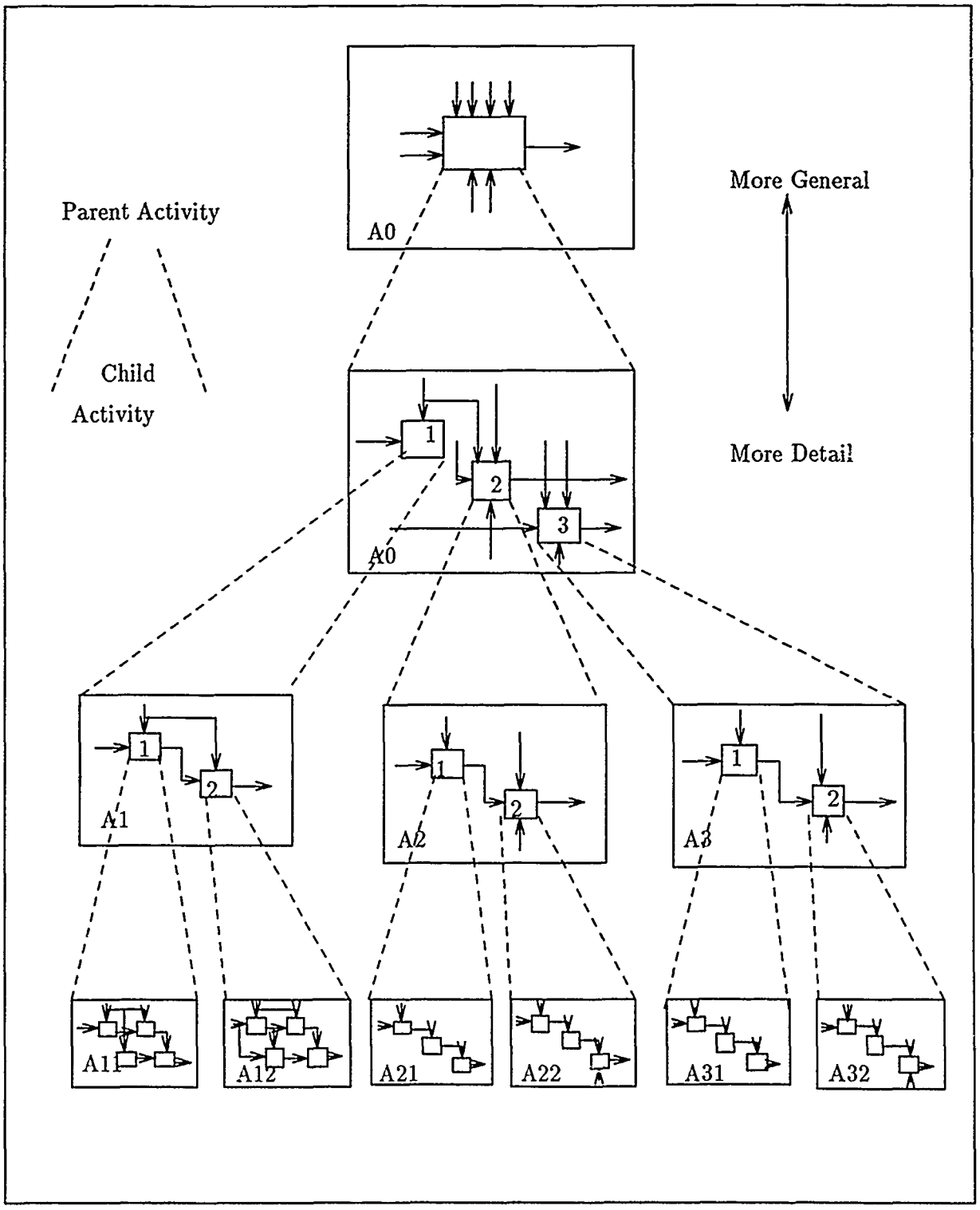


Figure 51. Sample $IDEF_0$ Hierarchical Decomposition

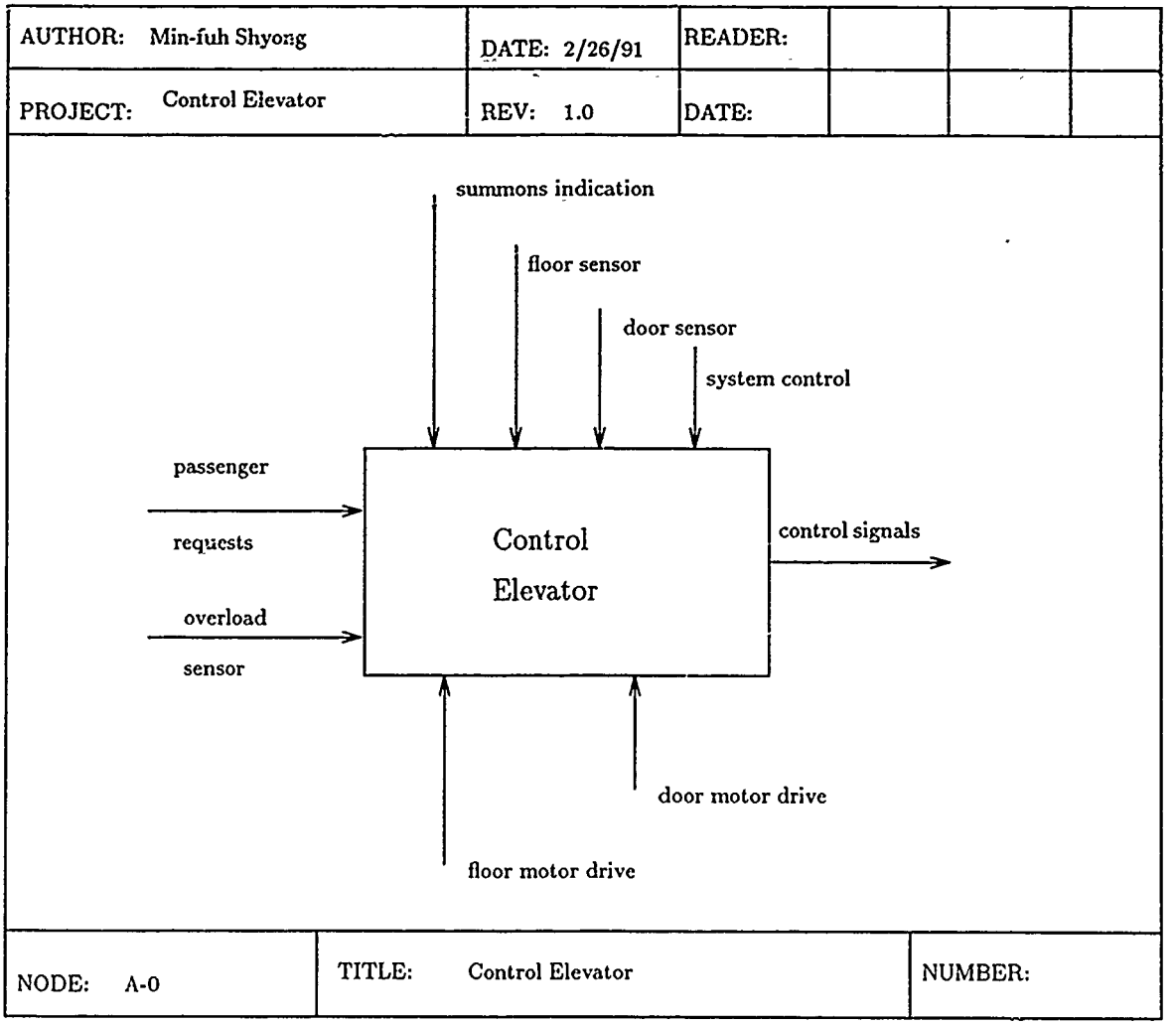


Figure 52. A-0 Diagram for 'Control Elevator'

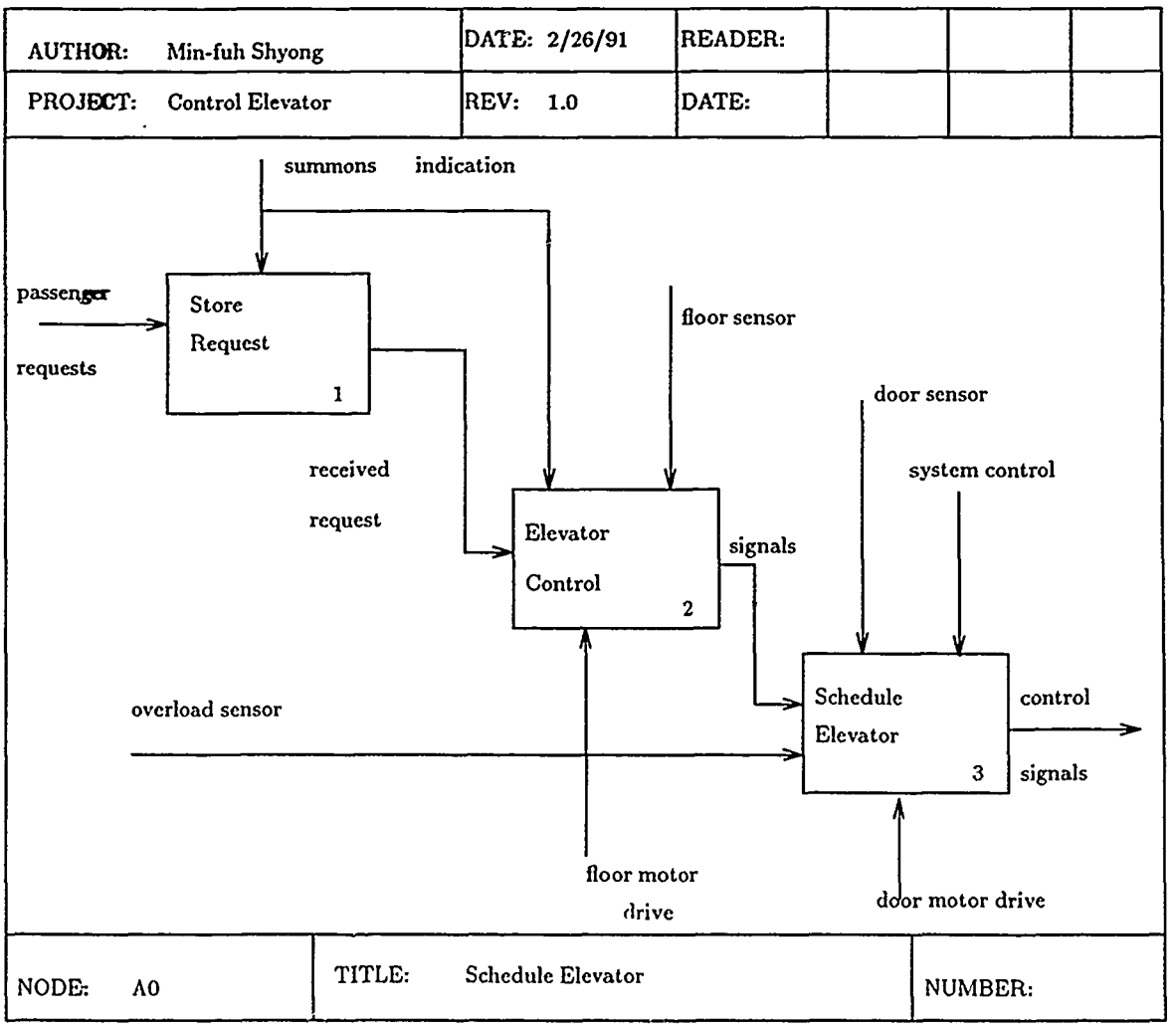


Figure 53. A0 Diagram for 'Control Elevator'

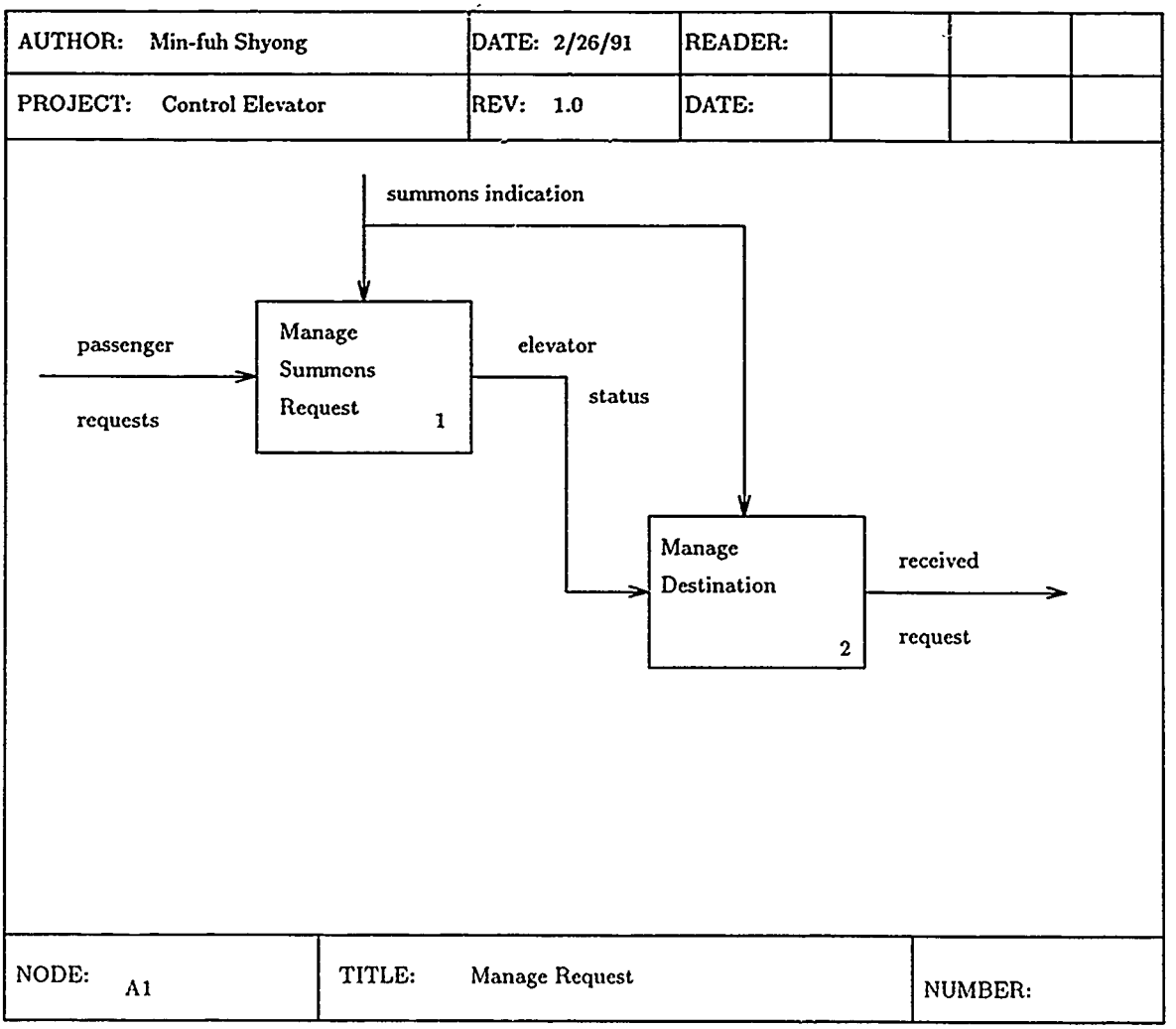


Figure 54. A1 Diagram for 'Control Elevator'

arrow is positioned on an activity determines what function that data element plays for that particular activity (10:68). There are four types of data elements: input, output, control, and mechanism (Figure 50). Arrows incoming at the left side of the box represent inputs. These are converted into outputs that always come out of the right side of the box. The top of the box is reserved for incoming control arrows. Every box has to have at least one control arrow. Input data elements are converted into output data elements depending on the constraints set by the control data elements. The bottom of the box represents mechanisms. Incoming bottom arrows represent means of performing the function. A downward arrow represents a special case of mechanism that indicates the activity is actually a part of another activity or system where its decomposition can be found.

The four types of data elements can be further broken down by the type of arrow drawn (10:68). The seven basic types of arrows are simple, pipeline, branch, boundary, tunnel, to_all, and from_all arrows (Figure 55). A simple arrow shows a direct connection between two activities. The output data element of the first is the input or control data element of the second activity. A pipeline arrow represents a data element that is formed from several data elements (bundle/join) or one that is broken down into subcomponents (fork). A branch shows a data element that breaks into two or more arrows, each carrying the same information. A boundary arrow is connected to one box only and represents the data elements that are shown going into or out of the parent activity. An incoming tunnel arrow is a boundary arrow that has come from an activity other than the parent of the given activity. The outgoing tunnel arrow indicates that the data is going to an activity other than one that is directly decomposed from it. A to_all arrow indicates that the output of a particular activity will be used as a control or input data element by all other activities in the diagram. It has a circle with a letter in the circle used to identify the particular to_all arrow. How this to_all data element is used by the other activities is determined by the from_all arrows. These arrows are composed of an incoming arrow with a circle at one end. In the circle is the identifying character that associates the from_all arrow with a particular to_all arrow. From_all arrows can represent control or input data elements. To_all arrows are always output data elements.

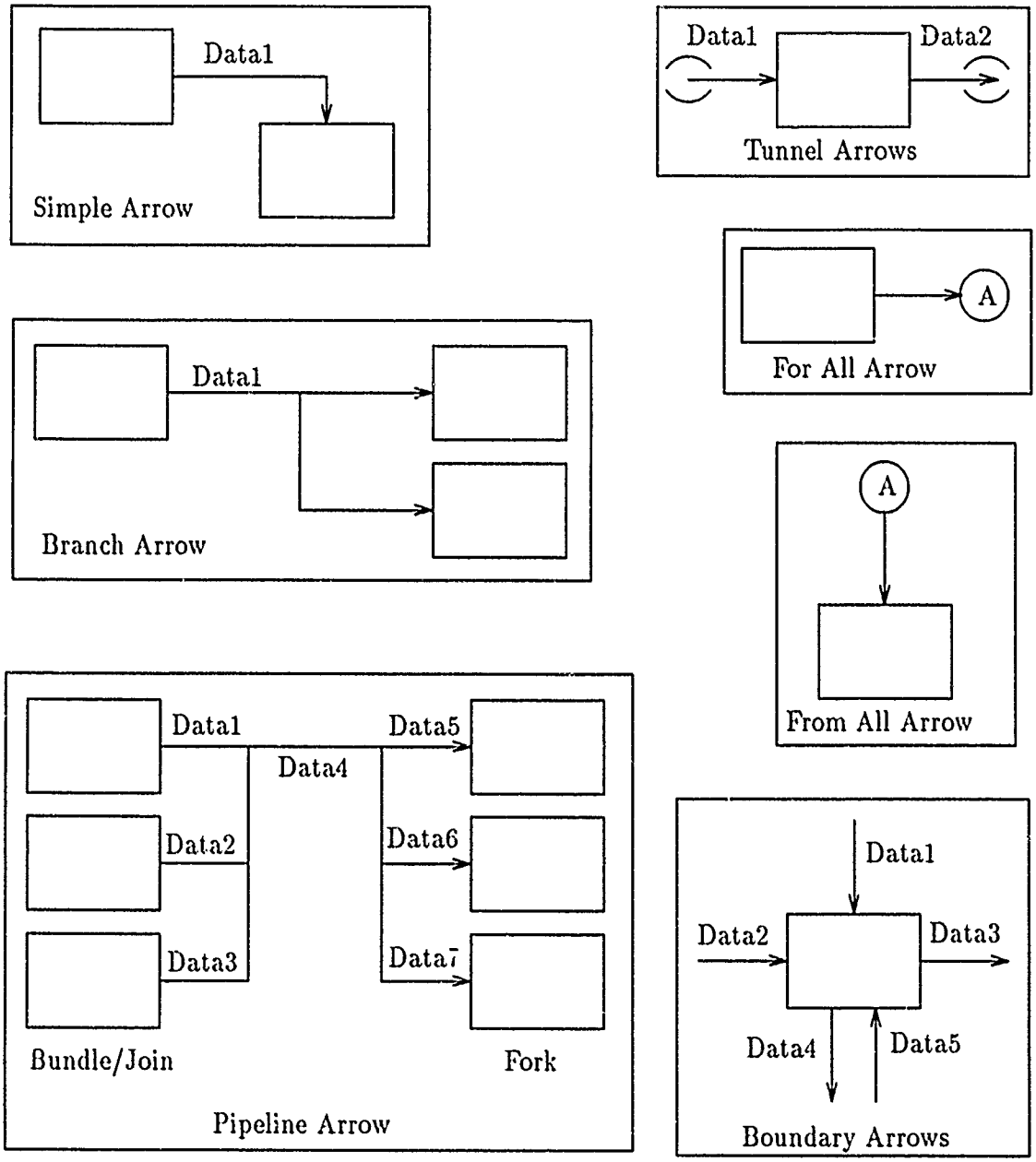


Figure 55. IDEF$_0$ Arrow Types

Bibliography

1. Austin, Kenneth A. and others. "An Entity Relationship Modeling Approach to $IDEF_0$ Syntax," *Proceedings of IEEE 1990 National Aerospace and Electronics Conference NAECON 1990*, *2*:641–645 (May 1990).
2. Batini, Carlo, Enrico Nardelli and Roberto Tamassia. "A Layout Algorithm for Data Flow Diagrams," *IEEE Transactions on Software Engineering*, *12*:538–546 (April 1986).
3. Booch, Grady. *Software Engineering With Ada*. Menlo Park, California: The Benjamin Cummings Publishing Company, 1986.
4. Booch, Grady. *Software Components With Ada*. Redwood City, California: The Benjamin Cummings Publishing Company, 1987.
5. Booch, Grady. *Object Oriented Design With Applications*. Redwood City, California: The Benjamin Cummings Publishing Company, 1991.
6. Brown, Marlin C. *Human Computer Interface Design Guidelines*. New Jersey: Ablex Publishing Corporation, 1988.
7. Dumas, Joseph S. *Designing User Interfaces For Software*. New Jersey: Prentice Hall, Inc., 1988.
8. Foley, Jeffrey W. *Design of a Data Dictionary Editor in a Distributed Software Development Environment*. MS thesis, School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, June 1986 (AD-A172406).
9. Ganes, Chris and Trish Sarson. *Structured Systems Analysis: Tools and Techniques*. St. Louis, Missouri: McDonnell Douglass Corp., 1982.
10. Hartrum, Thomas C. "$IDEF_0$ Requirements Analysis." Class handout describing the use of $IDEF_0$ for software requirements analysis, October 1989.
11. Hartrum, Thomas C. *System Development Documentation Guidelines and Standards*, January 1989.
12. Hoadley, Ellen D. "Investigating the Effects of Color," *Communications of the ACM*, *33*:120–125 (February 1990).
13. Humphrey, Watts S. *Managing the Software Process*. Reading, Masachusetts: Addison-Wesley Publishing Company, Inc., 1989.
14. Hyland, Stephen J. and Mark A. Nelson. "Ada Bindings to the X Window System." Ada computer software source code, 1987.
15. Johnson, Steven E. *A Graphics Editor for Structured Analysis with a Data Dictionary*. MS thesis, School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1987 (AD-A190618).

16. Jones, Oliver. *Introduction to the X Window System.* Englewood Cliffs, New Jersey: Prentice Hall, 1988.

17. Kitchen, Terry L. *An Object-Oriented Design and Implementation for the IDEF$_0$ Esssential Data Model with an Ada Based Expert System.* MS thesis, AFIT/GCS/ENG/90D-07, School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1990.

18. Materials Laboratory, Air Force Wright Aeronautical Laboratories, Air Force Systems Command, Wright-Patterson AFB, OH 45433. *Integrated Computer-Aided Manufacturing (ICAM) Function Modeling Manual (IDEF$_0$)*, June 1981.

19. Matingly, Joseph. *Establishing a Methodology for Evaluating and Selecting Computer Aided Software Engineering Tools for a Defined Software Engineering Environment at the Air Force Institute of Technology School of Engineering.* MS thesis, AFIT/GCS/ENG/91-D, School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1991.

20. Myers, G.J. *Software Reliability, Principles and Practices.* New York: Wiley, 1976.

21. Prototzko, L. Beth et al. "Towards the Automatic Generation of Software Diagrams," *IEEE Transactions on Software Engineering*, *17*:10–21 (January 1991).

22. Rottman, Michael S. *A Common Interface Real-Time Multiprocessor Operating System for Embedded Systems.* MS thesis, School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, March 1991.

23. Scheifler, Robert W. and Jim Gettys. "The X Window System," *ACM Transactions on Graphics*, *5*:79–109 (April 1986).

24. Shyong, Min-Fuh. *An Ada-Based Expert System for the Ada Version of SATool II.* MS thesis, AFIT/GCS/ENG/91-J, School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, June 1991.

25. Smith, Nealon F. *SAtool II: An IDEF$_0$ Syntax Data Manipulator and Graphics Editor.* MS thesis, School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1989 (AD-A215289).

26. Sommerville, Ian. *Software Engineering.* Massachusetts: Addison-Wesley Publishing Company, 1989.

27. Tevis, Jay Evan J. *An Ada-Based Framework for an IDEF$_0$ CASE Tool Using the X Window System.* MS thesis, AFIT/GCS/ENG/90D-15, School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1990.

28. Yourdon, Edward. *Modern Structured Analysis.* Englewood Cliffs, New Jersey: Prentice Hall, Inc., 1989.

Vita

Captain Betty Topp was born March 19, 1963 in Brooklyn, New York. She graduated in 1981 from Colegio San Antonio Abad in Humacao, Puerto Rico and entered the United States Air Force Academy with the class of 1985. On May 29, 1985 she graduated with a Bachelor of Science degree in Computer Science and earned a regular commission into the United States Air Force as a second lieutenant. Her first assignment was to the Command, Control, Communications and Countermeasures (C3CM) Joint Test Force at Kirtland AFB, New Mexico. There she served four years as a data analyst for several field experiments dealing with C3CM critical to the joint missions of the Air Force and Army. Her next assignment was to the Foreign Technology Division at Wright Patterson AFB, Ohio where she served for one year as a senior programmer. She entered the School of Engineering at the Air Force Institute of Technology in May of 1990.

Permanent address: 5495 Gander Rd S.
Dayton, OH 45424