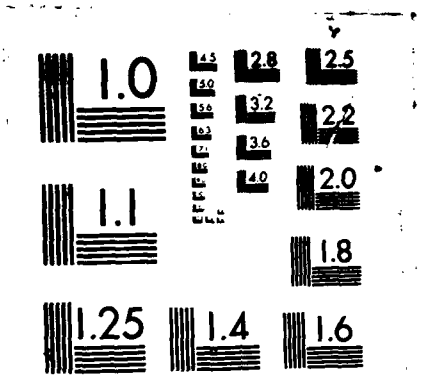
AD-A183 505 SOFTWARE ARCHITECT'S WORKSTATION A TOOL FOR ADA (TRADEMARK) PROGRAM DEVELOPMENT(U) NAVAL UNDERWATER SYSTEMS CENTER NEWPORT RI P J FORTIER ET AL. 01 JUL 87 1/1

UNCLASSIFIED NUSC-TD-6630 F/G 12/5 NL

END
9-87
DTIC



(12)

NUSC Technical Document 6630
1 July 1987

Software Architect's Workstation: A Tool For Ada Program Development

P. J. Fortier
P. A. Bergandy
Combat Control Systems Department

N. Bryden
Syscon Corp.

R. Charette
Computer Science Corp.

S. Trager
Softech, Inc.

Naval Underwater Systems Center
Newport, Rhode Island / New London, Connecticut

PREFACE

This report was prepared under NUSC Project No. 63756a, Task Area No. E1D of the "Software Technology for Adaptable Reliable Systems (STARS) Program," principal investigator P.J. Fortier (Code 2222), NUSC program manager T.P. Conrad (Code 2211). The sponsoring activity is the Naval Sea Systems Command, program manager P.J. Andrews (SEA-61R2).

REVIEWED AND APPROVED: 1 JULY 1987

J.R. Short
Head, Combat Control Systems Department

AD-A183 505

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3 DISTRIBUTION / AVAILABILITY OF REPORT Approved for public release; distribution unlimited.
2b DECLASSIFICATION / DOWNGRADING SCHEDULE	
4 PERFORMING ORGANIZATION REPORT NUMBER(S) TD 6630	5 MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION Naval Underwater Systems Ctr 6b OFFICE SYMBOL (If applicable) Code 2222	7a. NAME OF MONITORING ORGANIZATION
6c. ADDRESS (City, State, and ZIP Code) Newport Laboratory Newport, RI 02841-5047	7b ADDRESS (City, State, and ZIP Code)
8a. NAME OF FUNDING / SPONSORING ORGANIZATION Naval Sea Systems Command 8b OFFICE SYMBOL (If applicable) SEA-61R2	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (City, State, and ZIP Code) Washington, DC 20362	10 SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. / PROJECT NO / TASK NO. / WORK UNIT ACCESSION NO

11 TITLE (Include Security Classification)
SOFTWARE ARCHITECT'S WORKSTATION: A TOOL FOR ADA PROGRAM DEVELOPMENT

12 PERSONAL AUTHOR(S)
Fortier, P.J., Bergandy, P.A., Bryden, N., Charette, R., and Trager, S.

13a TYPE OF REPORT	13b TIME COVERED FROM ____ TO ____	14 DATE OF REPORT (Year, Month, Day)	15 PAGE COUNT
		87-07-01	24

16 SUPPLEMENTARY NOTATION
An adaptation of a paper originally presented at the Hawaii International Conference on Systems Sciences, January 1987.

17 COSATI CODES			18 SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	Software Systems Development Ada Computer Language
09	02		

19 ABSTRACT (Continue on reverse if necessary and identify by block number)

This report describes the software architect's workstation (SAW) -- a tool constructed to aid designers of Ada code for real-time command, control, and communications systems. The SAW is part of the STARS (software technology for adaptable reliable systems) software engineering initiative. The concepts associated with SAW development, the methodology used, and the techniques that make up the methodology are described. Also, a short example of surface-to-air missile simulation is presented to show the application of the SAW to a real-world situation.

20 DISTRIBUTION / AVAILABILITY OF ABSTRACT ☐ UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT ☐ DTIC USERS	21 ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED
22a NAME OF RESPONSIBLE INDIVIDUAL P.J. Fortier	22b TELEPHONE (Include Area Code) (401) 849-3703 22c OFFICE SYMBOL Code 2222

DD FORM 1473, 84 MAR 83 APR edition may be used until exhausted
All other editions are obsolete

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

SOFTWARE ARCHITECT'S WORKSTATION: A TOOL FOR ADA PROGRAM DEVELOPMENT

INTRODUCTION

The current state of software engineering practice in Department of Defense software development reflects a number of serious shortcomings that result in inadequate software systems. These inadequacies stem from one or a combination of the following:

- Failure to meet the stated requirements of a design
- Inadequate or incorrect system design
- Inadequate system performance
- Failure to properly address the man-in-the-loop interface adequately.

A first step in correcting these shortcomings is more rigorous and structured engineering of software systems instead of the typical software development process.

A more structured approach to software development (i.e., a methodology) must provide integrated techniques that allow for the capture of the pertinent aspects of the real-world system, along with the ability to refine and transform these aspects into software specifications and design.

The software architect's workstation (SAW) is a tool comprising an integrated set of methods and techniques to address the various portions of the software system development process (figure 1).

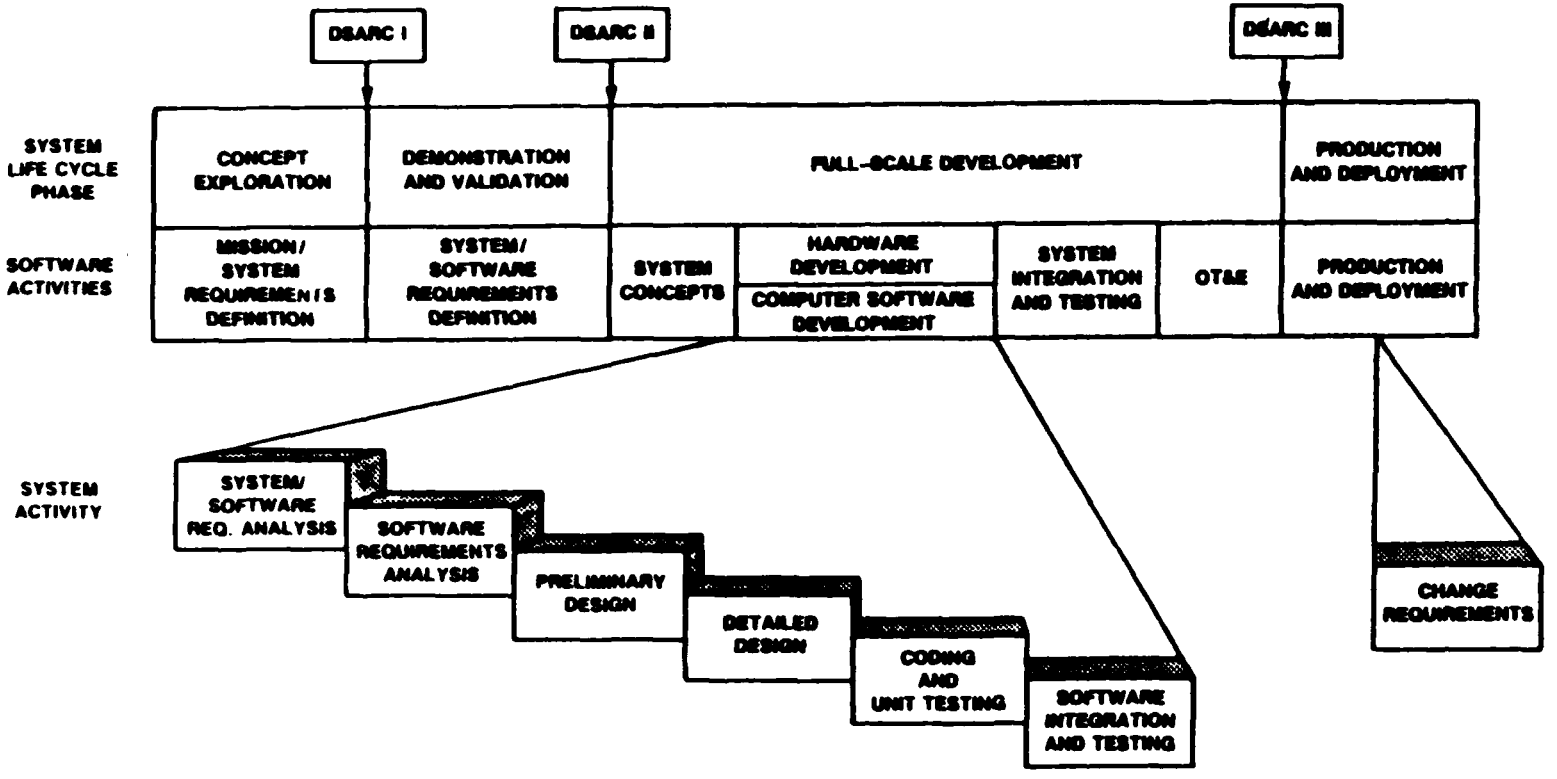


Figure 1. Computer Software Development Process

SOFTWARE BUILDING METHODOLOGY

METHODOLOGY REQUIREMENTS

A software building methodology must successfully produce and transform real-world system requirements into a precise statement of the software system's external behavior. It must possess a creative aspect that can help derive the specifications, as well as an easily used clerical aspect that can document them. Moreover, the methodology must fully describe the interfaces, modes of operation, and functions of the system. Important characteristics are that it be minimal (produce a blackbox view), understandable, accurate, precise, and easily adaptable.

The methodology must permit realization, in some abstract form, of the software system as described in the specifications. It must be able to support abstraction, decomposition, and the notion of hierarchy; it must also provide traceability and correctness analysis. Both the creative and clerical aspects of the methodology must be supported by techniques such as information hiding, abstraction of data types, stepwise refinement, data flow analysis, and graphic decomposition.

METHODOLOGY COMPOSITION

The SAW's methodology is based on one that has recently been successfully developed (references 1, 2, 3) to fulfill the above criteria, and that is currently being extended by the Naval Underwater Systems Center (reference 4). The methodology is an integrated approach for performing system/software requirements analysis and for producing software requirements, specifications, and designs for Ada* programs. It currently consists of five methods:

- The Air Force's IDEF modeling technique
- The Naval Research Laboratory's Software Cost Reduction Project (SCRP) techniques
- The Air Force-developed SAINT simulation language
- The Ada Iconic Language Builder
- The Combat System Software Simulator.

IDEF Modeling Technique

The IDEF (integrated computer-aided manufacturing definitional methods) comprises a set of structured definitional and analysis techniques for

*Ada is a trademark of the U.S. Department of Defense (Ada Joint Programming Office).

performing system analysis. This method provides a means of understanding and managing systems complexity and structure via functional, data flow, and behavioral perspectives. These perspectives allow for the modeling of a system in terms of its functions, their interfunction relationships and interfaces, information flow and content within the system , along with dynamic interaction of the system with its environment.

IDEF, which is based on Softech's structured analysis and design technique (SADT)™ (reference 5), has been used to model numerous systems. The technique uses concepts in topdown organization, modularity, hierarchy, and active relationships to describe systems under study.

The functional aspects provide a means of visualizing the structure of a system and the static relationships between system functions. Each function is described in terms of its activity, inputs to the function, outputs from the function, and controls or constraints imposed on the function (figure 2). To keep the level of detail manageable on a screen (or page), the IDEF technique limits the number of functions to six (SADT™ developed this restriction) (reference 5).

Such a functional description with its inputs, outputs, and controls allows for describing relationships between functions such as precedence, domain, and parallel or feedback conditions. These relationships are based on how the functions are linked and associated with one another on the screen (figure 3).

The behavioral view provided by IDEF defines the temporal nature of the system being modeled. It provides a means of defining how the functions interact with one another and their environment over time. As in the functional view, IDEF uses the box as its main descriptor. The box in this case has a different meaning. Its main components are the activation of activity description, and the input and output event descriptions (figure 4).

The input events are used to initiate the activity, and the output events result from the occurrence of the activity. To make the functional and behavioral views meaningful, the behavioral model descriptions must be associated with those of the functional model. That is, an activity represents an action on a corresponding function(s) in the functional model. A more detailed description of this technique can be found in reference 3.

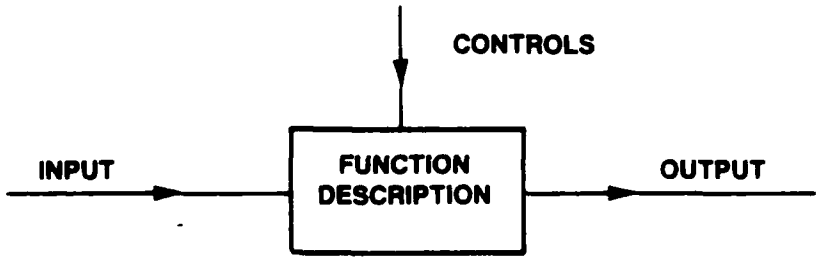


Figure 2. Syntax of IDEF Functional Component

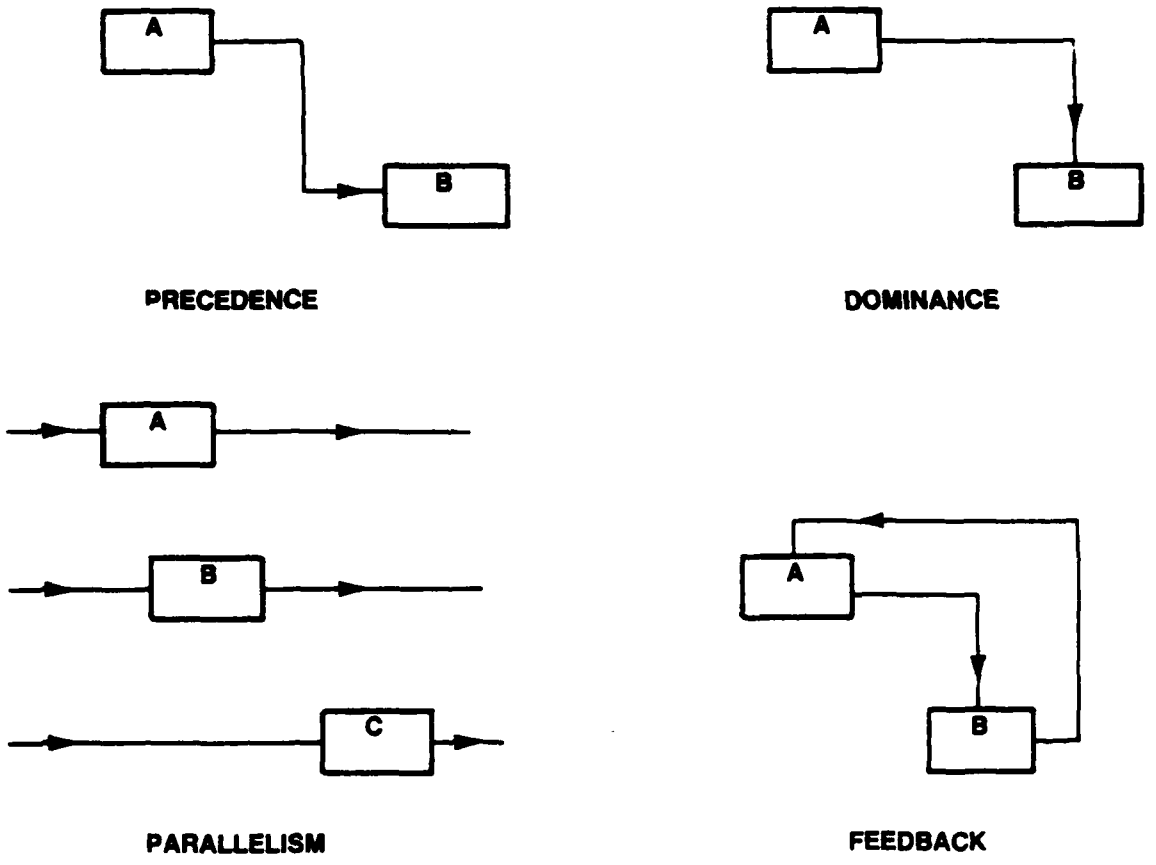


Figure 3. Data Flow Relationships

Figure 4. Activation of an Activity

SCRP Technique

The software cost reduction program (SCRP) (reference 6) is a set of formalized methods for specifying, designing, and documenting system operational requirements and for identifying missing or conflicting specifications. The emphasis in this technique is to have in place the questions one needs to answer in system development before beginning to collect and associate the answers. The technique has four major concepts:

- Separation of concerns
- Formal specification
- Abstract interfaces/information hiding
- Documentation.

The goal in separation of concerns is to link together (via the data base) information that needs to be linked and keep separate items that should not be linked. This concept limits or isolates the impact of changes on a system. Formalization of specifications defines a formal nomenclature by which one can describe the elements of a system. This allows removing the ambiguity of English text-only specifications. The third concept, information hiding, provides a means of isolating unrelated details from higher level components, thereby providing for ease of change and evolution. Finally, documentation (through a data base) allows organizing the class and content of information necessary to specify a system. Only through rigorous application of these concepts to a system can true benefits be realized.

SAINT Language

SAINT (systems analysis of integrated networks of tasks) is a simulation language developed in the late 1970s (reference 7). SAINT provides a means of modeling systems as a network of tasks that perform jobs and produce or consume resources.

To make this tool a part of the methodology, its elements must be mapped to those of the previous methods. A SAINT network of tasks is equivalent or can be mapped to a set of diagrams (functional and behavioral) in IDEF. In addition, IDEF resources equate to SAINT resources, and information flow in IDEF equates to attributes in SAINT. Using these relationships, one can construct a SAINT model that can be used to examine the present specification of a system. The model will give insight into correctness of the system and will indicate areas in need of refinement or change.

AICON Technique

AICON (Ada iconic builder technique) provides the capability to perform high-level conceptualization of Ada software designs using graphic techniques. The tool is based on Buhr's work (reference 8) on graphic design of Ada programs. The tool extracts information developed in the IDEF and SCRP models to construct the initial Ada code. This is currently performed through the mapping of IDEF functions and SCRP activations to Ada packages, tasks, etc, via a data base of stored items or through viewing previously stored information and manually extracting the pertinent aspects for the Ada description.

The basic notation available for use at this level consists of a set of editing and query modes to construct, change, or examine designs using icons and text.

The basic icons available include the package, task, subprogram, uncommitted module, data, data flow, access connection, sequence number, guard, and labels. Each of these relates to an underlying template that provides a means of describing the object in greater detail. The example given later in this report shows some details of the presentation.

Using the AICON tool, one can interactively develop an Ada program from the IDEF and SCRP description in a topdown fashion.

The use of icons instead of text-only presentation in the development of software provides users with much more feedback (in terms of perceptual information) upon which to judge design decisions. As more use is made of this tool, details of benefits will be published.

CSSIM

CSSIM (combat system software simulator) is an analysis tool that provides the means of analyzing tradeoffs between hardware and software during a systems design (reference 4).. The tool provides the ability to extract information from a design data base, and to model these at varying levels. Users select architectural components to model (e.g., CPUs, networks, sensor devices, secondary storage devices, operating systems, data base systems, etc.) from a component data case. If components are not available, templates are produced that allow users to build their own class of components based on a framework. Once the system hardware has been collected and formed, the user maps his software design onto the hardware system and can then model the operation of his software design instead of the postulated hardware design. Doing this iteratively, for either the hardware or the software, supplies a means of comparatively analyzing various combinations of a design (hardware/ software). This tool provides a means of finding and correcting flaws before producing a final specification and design.

INTEGRATION OF INFORMATION

The goal of employing multiple integrated methods or tools, each of which offers a different view of a system, is to provide a consistent view of all aspects of the system design to anyone using the tools. To provide this integration and consistent view implies a data base. This data base and its multiple views provide a means of integrating the various tool outputs into one system view usable by all (refer to figure 1).

This approach implies the selection of a data model that is easily adaptable to a wide range of internal storage structures. To meet this requirement, the SAW developers have examined and embraced the use of an object-based storage environment that provides a means of associating an object with any data type (graphic, text, icon, table, program, template, etc.).

The use of this type of data base has allowed easy association of the various icons and their templates with one another (e.g., figures 5).

The key elements of this data base are the definition of the core templates and their relationships. The iterative application of the data structure to the offered data, along with their associated activations, provides the integrated environment for SAW to operate within. This data base supplies the framework for management controls on SAW tools. Details of the data base, its schema design, and use will be published in future reports.

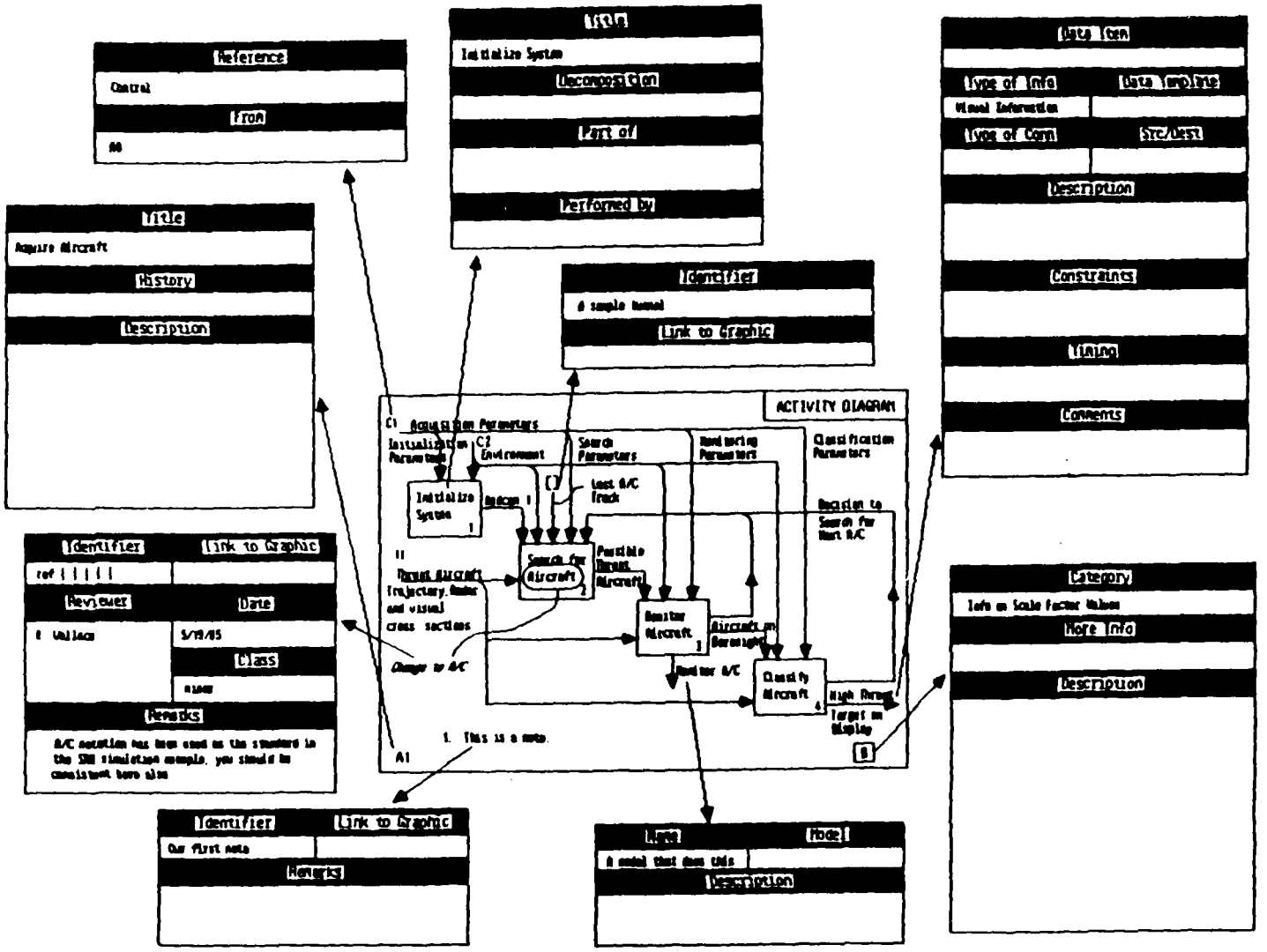


Figure 5. Sample of Active Associations

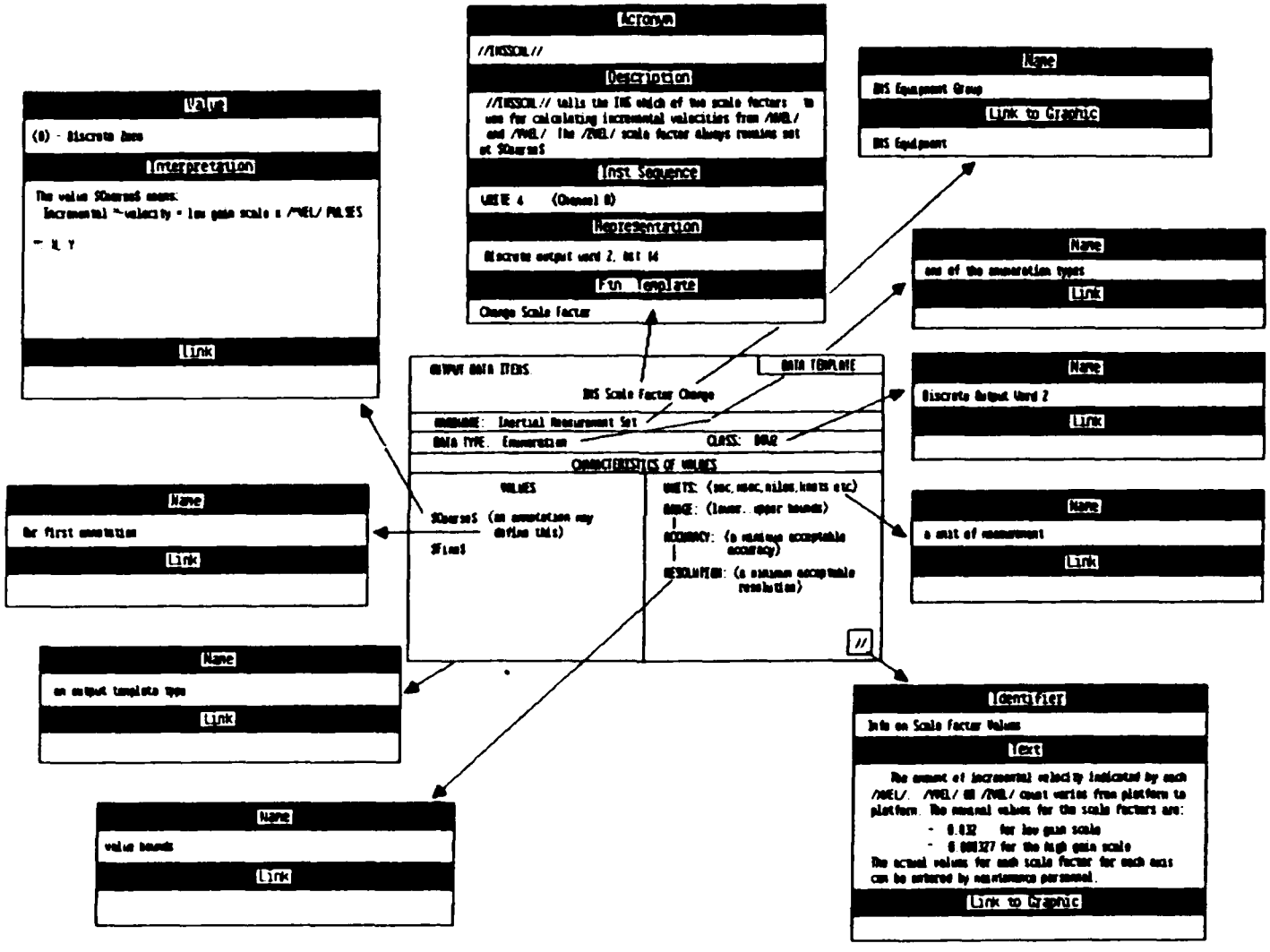


Figure 5. Sample of Active Associations (Cont'd)

USER INTERACTION

The user interface is of extreme importance to the success of the SAW as a software architect's tool. This interface comprises the method and the mechanisms used to provide information to the users. These mechanisms for manipulating information must be consistent, friendly, helpful, and understandable for best results.

To achieve these objectives, the SAW uses an object orientation for the screen interface. Functions available are shown either as icons (as in the Apple Macintosh™) or as pulldown menus. The icons and menus provide the available options to users. To examine the screen, users employ a mouse to move the cursor about and select the proper function to perform or object to manipulate. Once an option has been selected, the next level of options is brought up into active mode. This process occurs for all levels of the methods and across methods. Once selected, objects can be expanded to view internals or contracted to further recess detailing.

The following example involving the surface-to-air missile (SAM) simulator will help to clarify and highlight some of the qualities of the SAW technique.

The SAM simulator (figure 6) consists of a simulated track TV camera, track radar, acquisition radar, and three control consoles requiring a three-person crew. The crew responsibilities are, in a general sense, divided into three areas; the Fire Control Officer is the commander and is responsible for aircraft acquisition, missile launching, and overall control of the system. The other two operators, elevation and azimuth, are responsible for tracking the threat aircraft in either the 0-plane or the azimuth-elevation plane so that effective threat evaluation and missile launch/intercept can take place.

During the incoming attack, the crew is responsible for evaluating C^3 (command, control, and communications) information, for acquiring the aircraft with the acquisition radar, for transitioning from the acquisition mode to the track mode, for tracking the aircraft with the track radar, and for launching missiles at the aircraft. These activities are optimized by effectively evaluating C^3, TV, and radar data, and by communications among the crew.

To design such a system, a user would begin by formulating an understanding of the system, and then developing the IDEF activity and behavior diagrams in a hierarchical topdown fashion (e.g., figure 7). All items on the screen are viewed as objects and therefore have associated with them their own context. By defining/selecting an item, a software architect can construct or examine underlying templates that are automatically provided upon item creation as part of its data base object definition (see figure 5). The user iteratively develops the diagrams in a topdown fashion and fills in the generated templates to construct a nearly complete description of the system being studied.

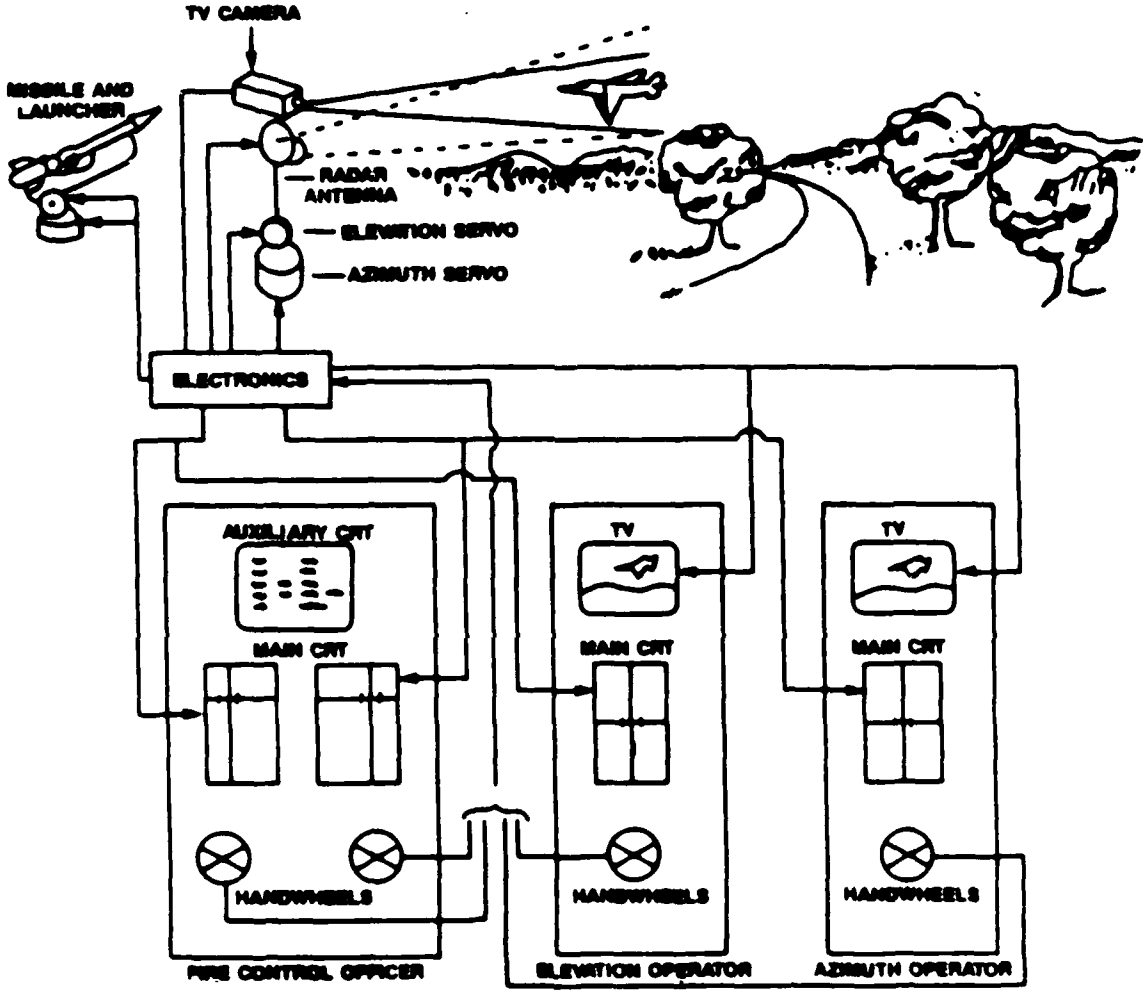


Figure 6. Simulation of a SAM System

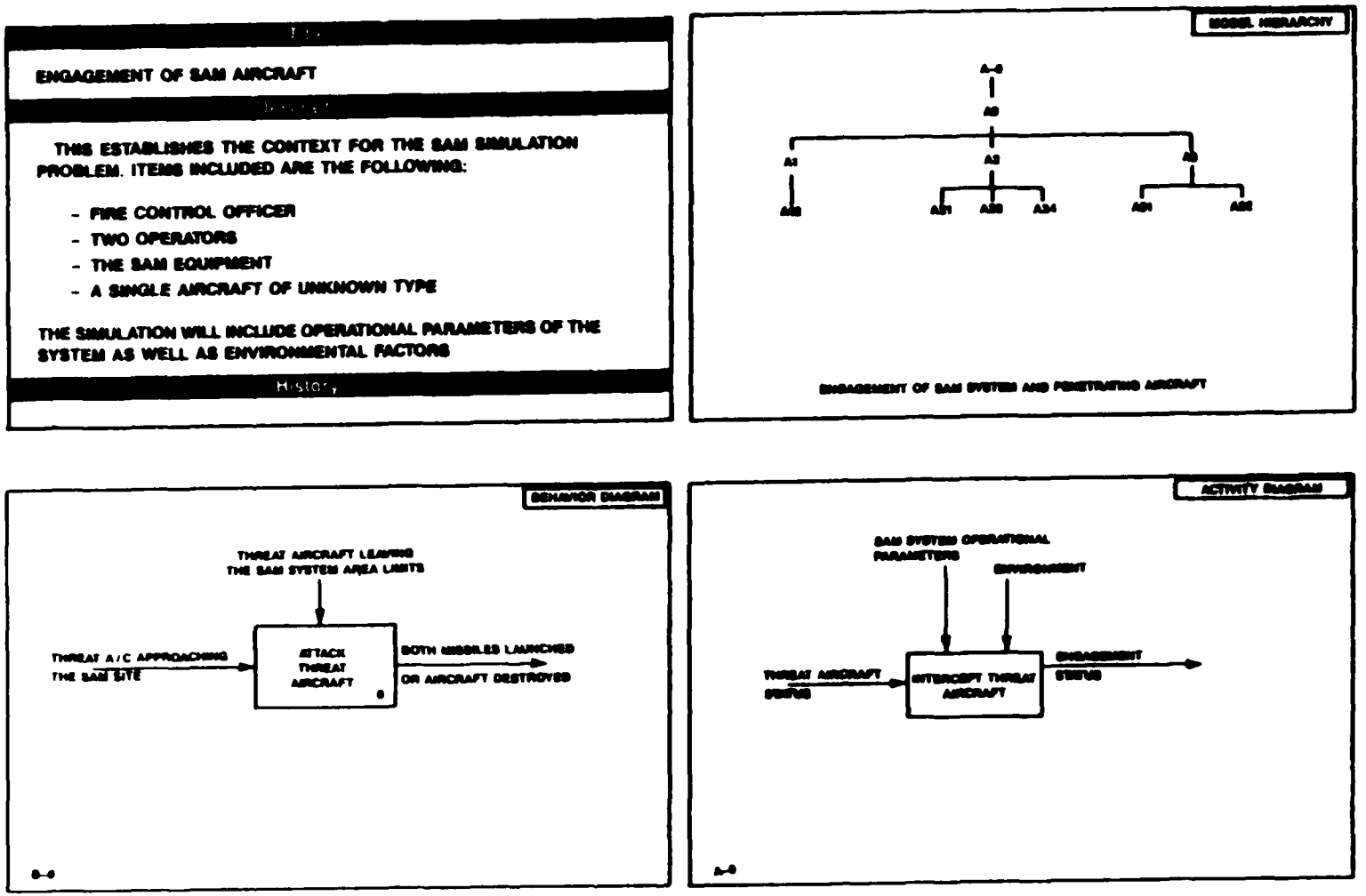


Figure 7. Behavior Diagram, Model Hierarchy, and Activity Diagram for SAM System

To further enhance the completeness of the design, the user develops the SCRP descriptions. These provide a means of capturing information not readily available from the IDEF models; included in this are such factors as:

- Distinguishing characteristics of the computer environment (rotary switch, touch panel, etc.
- Input and output data items
- Modes of operation
- Time-independent description functions
- Timing requirements
- Accuracy constraints on functions
- Undesired event responses
- Required subsets
- Expected types of changes
- Sources for further information.

Figure 8 shows the relationship of IDEF behavior and activity diagram components to SCRP tablets. (Details of SCRP use and interpretations can be found in references 1, 4, and 6.) These relationships can be used to construct SAINT task models to study the operations of the postulated system. Through iterative modeling and refinement of IDEF/SCRP descriptions, a sound design can be realized.

Once a high-level system design is fairly stable, the user develops the detailed software architecture using the AICON tool. For example, using the information associated with activity diagram, a user can construct a high-level design of the system. This high-level design may then be further broken down into submodes with details of the specified interfaces. The graphic capabilities of the AICON tool allow the user to create data flow graphs and structure graphs of the system under design.

By iterating in a topdown fashion, a detailed view of the software system in Ada iconic form (figure 9) can be developed. This detailed view of the Ada program can then be automatically turned into Ada code to examine its correctness, or it can be turned over to CSSIM to be analyzed against the postulated hardware design.

Using CSSIM, the user builds models of the hardware and software architecture and examines various mappings of one to the other. These models allow tuning of the system load before actual construction or coding is performed. CSSIM is in detail in reference 9.

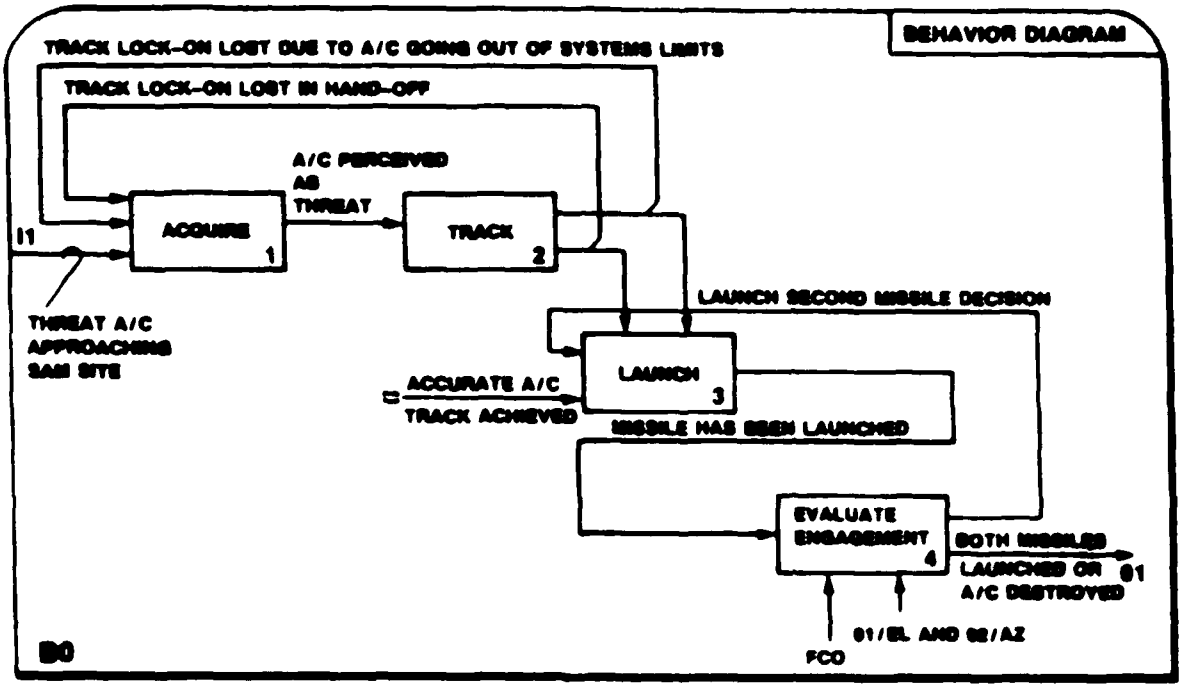


TABLE TITLE: INTERCEPT THREAT AIRCRAFT MODE CONDITIONS			MODE CONDITION
	//THREATIND// =	//ACTRACLKL// =	Other
Acquire	$Nothreat$	$Initial$ OR $Nolock$	//NUMMISLH// < 2
Track	$Threat$	—	//NUMMISLH// < 2
Launch	$Threat$	$Lock$	(//NUMMISLH// = 0) OR (//NUMMISLH// = 1 AND !Second launch dec! = Yes)
EvalEngage	$Threat$	—	(//NUMMISLH// = 1 AND !Second launch dec! = No) OR //NUMMISLH// = 2

Figure 8. Relationship of IDEF Behavior and Activity Diagrams to SCRP Tablets

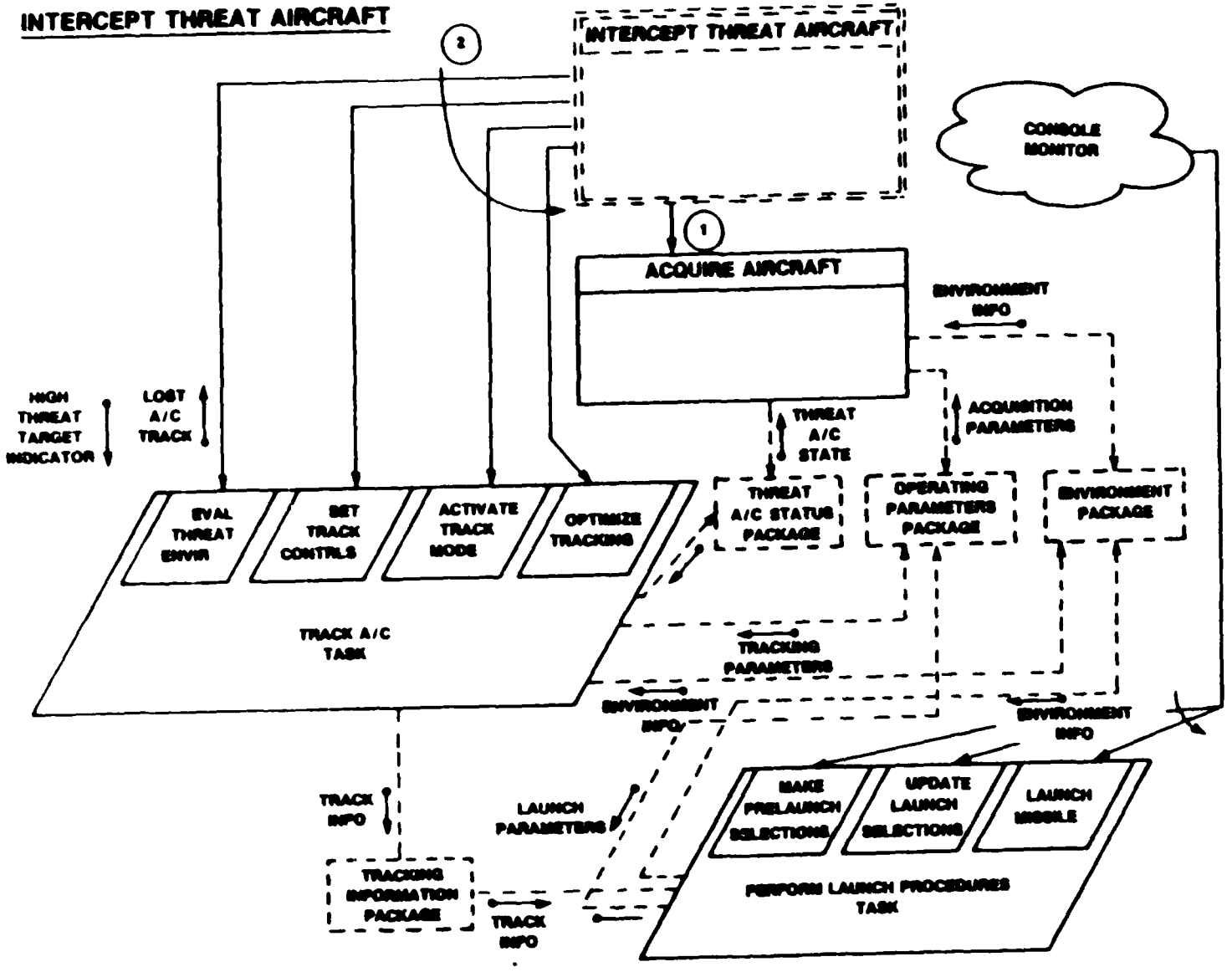


Figure 9. Detailed View of Software System in Ada Iconic Form

SUMMARY

The software architect's workstation (SAW) is a tool constructed to aid designers of Ada code for real-time C^3 systems. The SAW's methodology comprises a number of integrated methods and techniques for evaluating the various portions of the software development process. The methodology provides a means of effectively describing a software system in terms of its structure, behavior, and information flow, and of documenting a wide rang' of information associated with these views.

The biggest challenge in the development of the SAW has been (and will continue to be) the integration of the items within the data base. The object view of data and the qualities embedded within this view have been found to supply the wanted effect. That is, thay have provided a means of storing, retrieving, and associating various data types within the schema model without complex translations and processing.

Future reports will address the problems, solutions, and findings associated with implementing the data base on a workstation, as well as the use of this tool in the requirements analysis, specification, design, and management of real-time C^3 system software for DoD projects.

REFERENCES

1. R. Charette, "SOEM: Putting Theories into Practice," Third International Workshop on Software Specifications and Design, August 1985.

2. R. Charette and R. Wallace, "A Methodology for Addressing System Operability Issues," IEEE/NAECON 85 Proceedings, May 1985.

3. R. Wallace, J. Stockenberg, and R. Charette, A United Methodology for Developing Systems, McGraw-Hill, New York, 1987.

4. P. Fortier and L.D. Juttelstad, "Real Time Hardware/Software Simulation, Design and Use as a Performance Evaluation and Prediction Tool," Proceedings of 16th Hawaii International Conference on Systems Sciences, January 1983.

5. D. Ross, "Structured Analysis (SA): A Language for Communicating Ideas," IEEE Transactions on Software Engineering, vol 17, no. 7, July 1977.

6. K. Henninger, R. Parker, D. Parnus, and J. Shore, "Software Requirements for the A7B Aircraft," NRL Memorandum Report 3876, Naval Research Laboratory, Washington, DC, November 1978.

7. D. Seifert and G. Chubb, "SAINT: A Combined Simulation Language for Modeling Large Complex Systems," Report AMRL-TR-78-48, Aerospace Medical Research Laboratory, Wright-Patterson Air Force Base, Ohio, 1978.

8. R.J. A. Buhr, System Design with Ada, Prentice Hall, Englewood Cliffs, NJ, 1984.

9. P. Fortier, "Generalized Simulation Model for the Evaluation of Local Computer Networks," Proceedings of the 15th HICSS, January 1982.

INITIAL DISTRIBUTION LIST

Addressee	No. of Copies
NOSC	1
NSWC, White Oak	1
DTIC	12

END

9-87

DTIC