RL-TR-92-148
Final Technical Report
June 1992 1992

C3I REUSABLE SPECIFICATIONS

International Software Systems, Inc.

David Burlingame

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

Rome Laboratory
Air Force Systems Command
Griffiss Air Force Base, NY 13441-5700

This report has been reviewed by the Rome Laboratory Public Affairs Office (PA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

RL-TR-92-148 has been reviewed and is approved for publication.

APPROVED: William E. Rzepka

WILLIAM E. RZEPKA
Project Engineer

FOR THE COMMANDER: John A. Graniero

JOHN A. GRANIERO
Chief Scientist
Command, Control & Communications Directorate

If your address has changed or if you wish to be removed from the Rome Laboratory mailing list, or if the addressee is no longer employed by your organization, please notify RL (C3CB) Griffiss AFB NY 13441-5700. This will assist us in maintaining a current mailing list.

Do not return copies of this report unless contractual obligations or notices on a specific document require that it be returned.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave Blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	JUNE 1992	Final Dec 88 - Dec 91

4. TITLE AND SUBTITLE

C3I REUSABLE SPECIFICATIONS

5. FUNDING NUMBERS

C - F30602-88-C-0029
PE - 62702F
PR - 5581
TA - 22
WU - 23

6. AUTHOR(S)

David Burlingame

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

International Software Systems, Inc.
9430 Research Blvd
Bldg 4, Suite 250
Austin TX 78759-6543

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Rome Laboratory (C3CB)
Griffiss AFB NY 13441-5700

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

RL-TR-92-148

11. SUPPLEMENTARY NOTES

Rome Laboratory Project Engineer: William E. Rzepka/C3CB/315-330-2762

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release - distribution unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

This effort developed Proto+, a CASE tool and a methodology that provide automated software tool support and guidance for requirements engineering and prototyping. Proto+ enables an analyst to establish and execute focused requirements and prototyping efforts to fully understand requirements, constraints, and functions allocated to the system. Proto+ is a rapid prototyping environment that supports an analyst in the definition and evaluation of prototype software system specifications and design. The System Specification and Design Language (SSDL) is a language that provides the formal foundation for Proto+. Software systems defined with SSDL are evaluated through interpretive execution. At any time during the prototyping process, the analyst can execute the prototype. A complete prototype includes three related representations. A dataflow-like representation of the system defines the data transformations and control flow in terms of process nodes, data stores, communication connections, and ports. Each process node in the hierarchy may be further defined through refinement. The second representation is defined by the process node behaviors. The behavior of each leaf process node consists of a set of data transformation rules between the process node's input ports and output ports. The final representation of a complete prototype is the information model through which an analyst represents real-world objects manipulated by the prototype. An object-oriented information model is supported by Proto+.

14. SUBJECT TERMS

Requirements engineering; rapid prototyping; executable specifications; reusable specifications

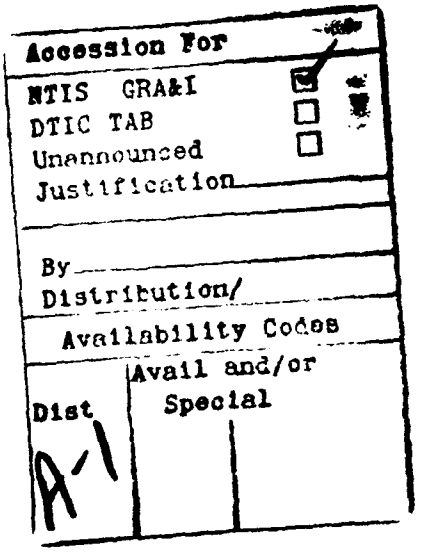
15. NUMBER OF PAGES

36

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

Table of Contents



Executive Summary

The traditional approach to software development (waterfall model), includes separate and distinct phases for software requirements analysis, design, implementation, and testing. Under this approach, requirements analysis is restricted to identifying the necessary software system capabilities. Determining how the software system requirements are satisfied is postponed until the design and implementation stages. In addition, software requirements validation (determining if the software system satisfies the requirements) is often not conducted until acceptance testing.

The traditional software development approach is adequate for developing small to medium-scale data-processing systems, where system requirements, constraints and functionality are well understood and formulated before software design and implementation. For large, complex, real-time systems, however, the traditional software development approach fails because software requirements cannot be completely and correctly identified without constructing and evaluating some part of the system's functionality. The adequacy, feasibility, and cost of the requirements cannot be completely and correctly determined without experimenting and evaluating a prototype of the software system that will ultimately implement the target software system.

The rapid-prototyping software development approach encourages end-user review and evaluation of the software requirements by experimenting with an early system prototype that constitutes one or more possible software solutions that satisfy the requirements. Requirements errors resulting from analyst, end-user, and developer misunderstandings and miscommunications can be detected early. Experience with rapid prototyping shows that identification of additional requirements and correction of existing requirements results from end-user interaction with the prototype.

Rapid prototyping is most effective when supported by computer-aided software engineering (CASE) tools. Adequate software tool support results in rapid prototyping consuming less development time and resources. The productivity of rapid prototyping is further enhanced if software component reuse is supported. In addition, software prototypes and systems constructed using well tested software components are of higher quality.

Proto+ includes a CASE tool and a methodology that provide automated software tool support and guidance for requirements engineering and prototyping. A key factor in the successful delivery of any software system is a clear understanding of the requirements by both the developers and customers. Proto+ enables an analyst to establish and execute focused requirements and prototyping efforts to fully understand requirements, constraints, and functions allocated to the system.

Proto+ is a rapid prototyping environment that supports an analyst in the definition and evaluation of prototype software system specifications and designs. The System Specification and Design language (SSDL) is a language that provides the formal foundation for Proto+. Software systems defined with SSDL are evaluated through interpretive execution. At any time during the prototyping process, the analyst can execute the prototype.

A complete prototype includes three related representations. A dataflow-like representation of the system defines the data transformations and control flow. This dataflow representation is referred to as the functional prototype. A functional prototype includes process nodes, data stores, communication connections, and ports. Each process node in the hierarchy may be further defined through

refinement.

The second representation of a complete prototype is defined by the process node behaviors. The behavior of each leaf process node consists of a set of data transformation rules between the process node's input ports and output ports. The data transformation is defined through a high-level structured language and includes support for assignment, condition, and iteration. In addition to the behavior constructs, math library functions and user-defined functions may be included in the behavior rules.

The final representation of a complete prototype is the information model through which an analyst represents real-world objects manipulated (get and set an object's attributes) by the prototype. An object-oriented information model (classes, subclasses, instances, and inheritance) is supported by Proto+.

The Proto+ tools are graphic-based, multi-window tools implemented on top of the Sun Microsystems Open Window environment. The tools support an object-oriented interaction method in which the end-user selects an object (with the mouse) and then applies an action to the object. All Proto+ tools are designed to minimize the degree of textual input.

A key strength of Proto+ is the ability to execute a specification. The Proto+ Interpreter supports interpretive execution of a prototype. In addition, an analyst may debug a specification by setting breakpoints, suspending and resuming execution, and examining any of the objects included in a prototype specification to determine their current state.

The final delivery of Proto+ includes the Proto+ methodology, the Proto+ tool set, a training course designed to acquaint new users with the Proto+ tools, and an example in the multiple-target tracking domain developed with the Proto+ tools.

1. General

The following paragraphs present the organization of the Final Technical Report for Proto+, relevant project references, and acronyms used in this document.

The project sponsor is Rome Laboratory (RL, formerly Rome Air Development Center), Command, Control, and Communications (C^3) Directorate, Software Technology Division (C^3C), Software Engineering Branch (C^3CB). C^3CB is also the user and operating center that will initially execute the completed CASE tools.

1.1 Organization of the Final Technical Report

The Final Technical Report for Proto+ provides a summary and analysis of the C^3I Reusable Specifications research and development contract (F30602-88-C-0029) also know as the Proto+. Section 1.2 summarizes key areas of the Proto+ statement of work. Project references and terms are presented in sections 1.3 and 1.4.

Section 2 summarizes results of a previous research and development effort - Proto (F30602-85-0129) that developed an initial version of CASE tool technology for requirements engineering and rapid prototyping. Proto+ subsequently extended and enhanced this initial technology. Section 2 concludes with a summary of the Proto+ research objectives.

Section 3 describes the technical approach of the Proto+ research project. Section 4 provides the project details and results including the Proto+ architecture and a brief explanation of the System

Specification and Design Language (SSDL). This section then illustrates the capabilities of the Proto+ tools. Section 5 summarizes the conclusions of the research effort and identifies specific areas for further research and development.

1.2 Summary of Statement of Work

The first of two key technical objectives involved development of an effective prototype development methodology that encompassed reuse of design specifications and emphasized evolutionary development of functional prototypes. The second key technical objective required the development of CASE tools that supported the enactment of the prototyping tasks identified in the methodology. Both the methodology and supporting CASE tools should effectively support rapid prototyping with an emphasis on the development and use of reusable design specifications.

The specific technical objectives of the methodology include supporting a general concept of systems development incorporating reuse and functional evolution. The methodology assumes that an integrated Requirements Engineering Testbed (RET) containing CORE Analyst [i], Proto+, and Requirements Engineering Environment/Rapid Interface Prototyping [n] (REE/RIP) tools will eventually be available to the systems analyst. The methodology should provide guidance for using each of the tools. The Requirements Engineering Environment Development (REED), an on-going R&D effort, will integrate these tools [q].

A set of CASE tools that supports the development of prototype design specifications represented graphically will be developed. Functional decomposition, specification of control flow, behavioral prototyping, and definition of real-world objects should also be supported by the CASE tools. During the construction of a prototype, an analyst should be able to search reuse libraries for components that can be tailored and incorporated in the developing prototype specification. At any point during the prototyping construction process, an analyst should be able to execute the prototype to evaluate performance, functionality, and behavior issues.

To support the Proto+ technology transfer, a user's manual will be developed and training sessions for Rome Laboratory (RL) personnel will be provided at each incremental delivery of the Proto+ tool set.

1.3 Project References

All references are unclassified.

1.3.1 Miscellaneous References

a. Statement of Work for the C^3I Reusable Specifications Contract, Air Force Contract No. F30602-88-C-0029, Griffiss AFB, NY, September 1988.

b. Functional Description for Proto+ (C^3I Reusable Specification). ISSI-C88A0002A

c. System/Subsystem Specification for Proto+ (C^3I Reusable Specification). ISSI-C888A00003A.

d. Program Specification for Proto+ (C^3I Reusable Specification). ISSI-C88A00004A.

e. User Manual for Proto+ (C^3I Reusable Specification). ISSI-C88A00005A.

f. Data Base Specification for Proto+ (C^3I Reusable Specification). ISSI-C88A00009.

g. Data Base User's Manual for Proto+ (C^3I Reusable Specification). ISSI-C88A000010.

h. Proto+ Methodology (C^3I Reusable Specification). ISSI-C88A01002.

1.3.2 References Applicable to the History and Development of the Project

i. "Analyst: CORE/Macintosh," Systems Designers Scientific, Camberly, Surrey, England, June, 1986.

j. Hartman, D.; Konrad, M.; and Welch, T., "VHLL System Prototyping Tool Final Report," Air Force Contract No. F30602-85-C-0129, Griffiss AFB, NY, December 1988.

k. ISSI, "RADC System Software Requirements Engineering Testbed Research and Development Program," June 1988.

l. ISSI, "SSDL Specification Language", Technical Report, ISSI-88A01011, May 1989.

m. Luqi and Valdis Berzins, "Rapidly Prototyping Real-Time Systems," *IEEE Software*, September 1988.

n. Software Requirements Specification for the Requirements Engineering Environment Development. ISSI-B89A00006.

o. Rzepka, W. and Ohno, Y., "Requirements Engineering Environments: Software Tools for Modeling User Needs," *IEEE Computer*, April 1985.

p. Statement of Work for VHLL System Prototyping Tools, Air Force Contract No. F30602-85-C-0129, Griffiss AFB, NY, August 1984.

q. Requirements Engineering Environment Development (REED), Air Force Contract No. F30602-89-C-0200.

1.3.3 Applicable Standards

r. DoD-Std-7935.1, "Automated Data Systems (ADS) Documentation Standards," April 1984.

1.3.4 Documentation Concerning Related Projects

s. Mullery, G.P., "CORE - A Method for Controlled Requirement Expression," Systems Designers Ltd., Camberly, UK, February 1979.

t. Rzepka, W. and Daley, P., "A Prototyping Tool to Assist in Requirements Engineering," 19th Hawaii International Conference on System Sciences, Honolulu, HI, January 1986.

u. Stephens, M. and Whitehead, K., "The Analyst - An Expert Systems Approach to Requirements Analysis," 8th International Conference on Software Engineering, London, UK, August 1985.

1.4 Terms and Abbreviations

CASE	Computer Aided Software Engineering
COTS	commercial off-the-shelf
GO	graphic object
GUI	graphical user interface

ISSI	International Software Systems, Inc.
MTT	multiple-target tracking
OM	object manager
ONTOS	COTS object manager
Proto	ISSI rapid prototyping tool; existing VHLL system prototyping tool - Air Force Contract No. F30602-85-C-0129
Proto+	Name of the enhanced VHLL system prototyping tool - Air Force Contract No. F30602-88-C-0029
RADC	Rome Air Development Center
REED	Requirements Engineering Environment Development - Air Force Contract No. F30602-89-C-0200
RET	Requirements Engineering Testbed
RL	Rome Laboratory
SSDL	System Specification and Design Language
UIM	User Interface Manager
UI	user interface

2. Project Overview

This section provides an overview of the Proto+ R&D project. The overview includes a project history that summarizes the objectives of the RADC System/Software Requirements Engineering Testbed Research and Development Program [k]. The results and suggested areas of additional research that resulted from the Proto project (Air Force Contract No. F30602-85-C-0129) are also summarized.

2.1 Project History

During 1988 a panel of experts in the areas of software engineering, software life-cycle models, requirements engineering, formal specification languages, and CASE tools met to develop a long-range plan for Rome Laboratory's Requirements Engineering Testbed (RET) program. The panel documented their research and development suggestions in the technical report "RADC Systems/ Software Requirements Engineering Testbed Research and Development Program" [k]. This report identified two research and development tracks:

- Evolutionary Track which identified R&D efforts to extend current techniques and tools to support rapid prototyping of interfaces, functionality, scenario development, reuse, and requirements analysis methods.
- Formal Language Track that involved research toward the development of a single formal language for expressing goals, requirements, constraints, and solution architectures.

Proto was one of the initial tools developed in support of the Evolutionary Track. The primary objective of Proto included providing functional prototyping capabilities through which an analyst could define, execute, evaluate, and validate software system specifications prior to extensive design and implementation efforts.

Proto+ represents a subsequent R&D effort funded by RL to address specific technical issues and areas not addressed by Proto. Section 2.3 summarizes the Proto+ technical objectives.

2.1.1 Summary of Proto Capabilities

Proto defined the initial implementation of a prototyping tool developed by ISSI for RL (Air Force Contract No. F30602-85-C-0129). Proto is used by a systems analyst to:

- create a functional specification
- refine the functional specification into a functional prototype that can be interpreted
- validate the specification, to verify the functionality is usable in the context of the target system, by interpreting the functional prototype before knowledgeable end-users of the proposed system

A Proto system specification is a hierarchical dataflow graph, where each node can be pg2

of a refinement which is itself a dataflow graph. The nested dataflow graph receives all its inputs from and sends all its outputs to the ports of its enclosing node (parent node). At the lowest level in a graph, a node can be a compiled object (compiled function) or a Script language body. Compiled functions are modules written in a programming language (C for Proto). Script bodies contain statements of the interpreted Script language, a simple structured programming language.

Proto was implemented on an Apollo DN3000 Unix workstation. The windowing environment and graphical user interface support was provided by the Apollo graphic and windowing system.

2.1.2 Summary of Suggested Proto Enhancements

Upon completion and delivery of Proto, the following areas for additional research were identified:

- Proto should be enhanced to enable specification of multi-threaded systems.
- The Proto Interpreter should be enhanced to allow execution of multiple processing paths.
- The Proto scheduling algorithm should be enhanced and users should have more control over the scheduling algorithm.
- A flexible cut and paste capability should be provided in the Graph Editor.
- Reuse libraries should be enhanced to allow multiple reuse libraries for a variety of domains.
- Enhanced reuse tools should be provided for defining, searching, browsing, and tailoring reusable components.
- Proto should be enhanced to use a common object manager to store all requirements engineering data.
- The Proto tools should be re-hosted on a more commercially viable platform.

2.2 Overview of Proto+

Proto+ is a rapid prototyping environment that supports an analyst in the prototyping and evaluation of software system specifications and designs. The formal foundation of Proto+ is the System Specification and Design Language (SSDL). The Proto+ tools implement this prototyping language through both graphic and text constructs.

SSDL is a language for expressing software system specifications and designs. Unlike other lan-

guages, software systems defined in SSDL can be executed. Under this strategy, an analyst can view the results of specifications, which enables correcting designs as soon as possible. This is the essence of the rapid prototyping development paradigm.

A complete prototype is defined by providing three related representations of a system. These three interdependent representations include:

- **Functional prototype** – dataflow hierarchy defining data transformation and control flow
- **Behavioral prototype** – computation or behavior at each leaf process node that transforms a process node's inputs into outputs
- **Information model** – object-oriented data model for representing data manipulated by the prototype

The functional prototype is defined using a graphical language. Through the functional prototype, the user identifies the processes, interfaces, connections, data stores, and refinements. The functional prototype, which provides a dataflow model of the prototype, is defined using the Graph Editor. The Graph Editor is a multi-window graphic editor that provides an object-oriented method of interaction where the end-user selects an object and executes a command on the object (edit, move, delete).

The computation or logic of each of the processes is defined by the behavioral prototype. The behavior of a process is defined using a formal, structured language. The Behavior Editor supports defining and checking of the behavioral prototype. Detailed behaviors of the prototype are entered with the Behavior Editor which supports a high-level structured language for defining detailed computation and the transformation of inputs to outputs.

The final portion of a complete prototype involves the data objects. Through the definition of the data objects, the user defines the data or information model that is manipulated by the functional and behavioral prototypes. The Data Editors provide an object-oriented data model consisting of classes and instances. The real-world objects manipulated by the prototype are implemented as class instances. The Data Editors are also multi-window based and minimize end-user text entry.

A significant capability of Proto+ is that at any time during the design of a prototype, the prototype can be executed. The Proto+ Interpreter supports the execution of a prototype. Through prototype execution an analyst analyzes function, behavior, and performance issues. The Interpreter includes debugging capabilities which enable setting and executing breakpoints, viewing the state of any objects (data, process nodes, data stores, and ports) at breakpoints and during execution, and saving debugging information to text files.

2.3 Proto+ Research Objectives

The Proto+ research objectives include enhancing and extending existing Proto capabilities, adding new capabilities with particular emphasis on improving the usability of the system, providing support for a broader range of prototyping needs, and supporting reuse in a requirements prototyping context.

The overall objectives of the resulting Proto+ R&D project include:

- provide a formal language for specifying prototypes
- provide user friendly tools for building and modifying prototypes
- provide tools to support the use and management of reuse libraries

- provide an interpreter to execute prototypes
- include a DBMS for managing all objects associated with a prototype
- re-host the prototyping tools on a more advanced workstation
- provide a methodology for using the system
- provide a user's manual for the system

3. Technical Approach

This section describes the technical approach adopted to achieve the project's objectives. The technical approach involved evaluating the current prototype (Proto) by both RL and ISSI personnel. Proto+ is only one of several CASE tools included in RET. To effectively use Proto+, the capabilities of Proto+ must be considered with capabilities of other RET tools; therefore, a methodology must provide guidance for when and how to use each of the RET tools. Section 3.2 includes the methodology objectives. Acceptance of a CASE tool requires significant end-user comment and evaluation during tool design and development. Section 3.3 presents the objectives and strategies to ensure final acceptance of the new technology by RL personnel.

To ensure effective transfer of new technology, guidance on how to use Proto+ was provided through user's manuals and training courses. Section 3.4 summarizes the technology transfer objectives and strategies.

3.1 Evaluation of Proto

Evaluation of the current requirements prototyping tool is essential for overall project success. RL personnel had considerable experience with Proto. Critical feedback pertaining to Proto and incremental versions of Proto+ was provided throughout the project. In addition to RL analysis and feedback, key personnel at ISSI also evaluated Proto. The analysis objectives included:

- Evaluate the database needs of Proto and determine the best database model for capturing and maintaining specification information.
- Evaluate different XView implementations to determine the most promising in terms of current capabilities and long-term support for user interface requirements.
- Evaluate the Proto default scheduling algorithm.
- Evaluate the Proto information model and determine whether a generalized object-oriented data model would prove more effective for representing real-world objects.
- Evaluate Proto reuse support and develop a more effective implementation of a reuse facility.

3.2 Methodology Development

The primary objective of the requirements prototyping methodology was to develop a methodology that addressed the front-end of the software life-cycle (requirements analysis and system design). In particular, the methodology should focus on the development of prototypes for analyzing software system requirements, constraints, and trade-offs. The specific strategy included:

- Determine the role that each of the RET tools serves in the requirements prototyping methodology.
- Determine how to integrate the different conceptual models and techniques supported by

each of the RET tools.

- Define strategies to enable all or a subset of the requirements data collected by the different tools to be shared by the tools and/or translated to the different tool representations.
- Describe the methodology steps and tasks in conjunction with tools that support these tasks.
- Develop and deliver the methodology for end-user review and comment at project mid-point.

3.3 Incremental Delivery of Technology

An organization's acceptance of any new CASE tool requires that the organization clearly understand the purpose and operation of the tool. Ensuring adequate time and support for end-user comment and feedback is critical for successful technology transfer. Numerous software development projects have failed because end-users were not able to clearly understand the operation and function of a software tool until final delivery. Requesting significant changes at the end of the project is often not possible because of schedule constraints and because budget and resources have been consumed. The strategy adopted by Proto+ to ensure end-user understanding and encourage end-user critical analysis and feedback throughout the project included:

- Deliver successive tool increments that provide incremental levels of functionality.
- At each delivery of an increment, provide adequate end-user training on the tool's conceptual model, capabilities, and installation.
- During the training courses provide hands-on experience with the tool.
- Provide drafts of user's manuals and installation guides at each increment delivery.
- Ensure that during and after tool delivery sufficient time and effort are allocated for eliciting end-user comments, enhancement suggestions, and software exception collection and analysis.

3.4 Technology Transfer

To ensure successful technology transfer three key strategies that included user's manuals, end-user training, and development of an example in a relevant domain were adopted. The specific strategies for ensuring the development of effective user's manuals included:

- Deliver user manual drafts or updates at each delivery of an increment.
- Ensure that the user's manual includes a short tutorial that illustrates the majority of the tools capabilities.
- Include in the user manual numerous tool screen images to ensure that the end-user clearly understands the look, feel, and function of the tool.

In addition, to the user's manuals, the end-user should be provided with structured training sessions that involve a high-degree of tool interaction. The examples used to illustrate the function of the tool should be simple enough to complete in an afternoon hands-on session. In addition to providing an explanation of how the tool operates, the training sessions should clearly describe the purpose and conceptual model for each of the Proto+ tools.

An example of a subsystem in a domain understood by the end-user should be developed and delivered as part of the project. Developing a real-world example verifies the capabilities of the tool and validates the effectiveness of the tool in a real-world domain.

4. Project Details and Results

This section provides the project details and results. Section 4.1 provides the detailed project objectives organized into seven areas. Section 4.2 then describes the Proto+ software system architecture. The system architecture design includes the software architecture, a summary of the various components, and a description of the dataflow between the components. The set of objects managed by the OM are also identified. Section 4.3 summarizes SSDL, the underlying formal language upon which all of the Proto+ tools are based. The individual Proto+ tools are summarized in Section 4.4. Section 4.5 and 4.6 summarize the results of the methodology development and the user's manual and training course development.

4.1 Detailed Objectives

The Proto+ detailed objectives are described in the following sections.

4.1.1 Language for Specifying Prototypes

The objectives for the prototype specification language include:

- Defined semantics
  The language should have precisely defined semantics to ensure unambiguous interpretation. Ideally, these semantics should have an underlying formal basis.
- Applicable to C^3I domain
  The language should encompass the C^3I problem domain. Supporting the C^3I problem domain requires that the language support prototyping of concurrent cooperating processes and processors.
- Minimize design decisions
  The language should minimize design decisions that have to be made in creating a prototype.
- Graphical syntax
  The language should provide a graphical dataflow, diagram-like syntax permitting hierarchial descriptions.
- Textual syntax
  The language should include a set of textual extensions provided for describing detailed behaviors or computation.

4.1.2 Tools for Building and Modifying Prototypes

A set of integrated, interactive tools should be provided that support the requirements prototyping activities of a systems analyst. Such a tool set should include an editor, an interpreter, and an interactive debugger to aid in the construction and execution of prototype systems.

The set of tools should include the following:

- Graph editor
  The system should provide an easy to use graph editor for building prototype systems.
- Multiple representations
  The system should provide appropriate representations for portions or aspects of a prototype. Key representations include graphical representations that define the dataflow and

state (control) information, an interpretable, high-level textual representation that defines the detailed computation or algorithm of the prototype system, and an object-oriented data model (classes, subclasses, instances) for defining the data objects manipulated by a prototype system.

- Graphics support features
  The system should provide support for managing the layout of multiple dataflow graphs including moving, shrinking, and scaling the graphs that enable the systems analyst to extract the necessary view of a particular multilevel hierarchical dataflow description.
- Support tools
  The system should provide tools to assist the analyst in ensuring the correctness of prototypes. Examples of such tools are consistency and completeness checkers.
- Close coupling between tools and database
  The system should provide a close coupling between the Proto+ tools and the Proto+ database so that tools can share objects with consistent interpretations.

4.1.3 Management and Use of Reuse Libraries

The objectives of the reuse libraries and associated capabilities include:

- Library management tools
  The system should provide a collection of tools for inserting and extracting reusable components from reuse libraries.
- Reuse Component Search and Extract Capabilities
  The system should provide a user friendly search capability with which an analyst can locate and extract components from reuse libraries by means of keyword queries.
- Emphasis on reuse of all portions of a prototype
  The system should support reuse of all portions of prototype systems (dataflow diagrams, behaviors, and data objects).

4.1.4 Interpreter

The objectives for the Interpreter include:

- Interpreter for directly executing prototypes
  An interpreter that can directly execute prototypes should be provided. Ideally, an analyst should be able to suspend the interpreter during a run, analyze the state of the prototype, and resume execution.
- Debugging features
  The system should provide a collection of features to assist the analyst in debugging prototypes. Typical features include breakpoints, pause/resume of execution, visual indications of which functions are being interpreted (through highlighting), and support for examination of data during interpretation.
- Instrumentation
  The system should provide tools and features with which the analyst can instrument a prototype. The goal of these tools is to help the analyst visualize the operation and state of a prototype.

4.1.5 Data Base Management System

The objectives of the DBMS include:

- Manage objects
  All objects associated with a prototype should be stored and managed by the DBMS. As objects are referenced they should be brought from disk to memory in a manner that is transparent to the end-user.
- Object-oriented
  The DBMS should support an object-oriented design paradigm required by the prototype-building tools.

4.1.6 Execution Platform

The current execution platform of Proto cannot include the significant improvements of advanced hardware architectures (RISC) and the resulting improvement in performance. In addition, the improvements in Graphical User Interfaces (GUI) provided by the various standards and user interface utilities (Open Look and XView) are not available on the current Proto platform. A key objective of Proto+ will be to host the prototyping tools on Sun Microsystems Sun 4 and SPARCstation workstations. In addition to providing increased performance, the family of Sun workstations shows the most promise for remaining a commercially viable line of workstations.

4.1.7 Proposed Methods and Procedures

The methods of the previous prototyping project - Proto (Air Force Contract No. F30602-85-C-0129) primarily support functional prototyping. The Proto methods will be significantly extended and modified under the Proto+ project to include an extensive requirements prototyping methodology. The Proto+ methodology will describe in detail how an analyst with the Proto+ tools develops a complete prototype.

The organized set of methods will be documented in the Proto+ Methodology Report. The key factors that must be addressed by the Proto+ methodology include:

- Concurrent Execution Model
  Proto+ supports a concurrent execution model allowing C^3I prototyped systems made up of multiple communicating processors to be defined. Proto+ supports a more realistic and flexible execution model enabling the definition of prototype systems consisting of multiple processes running on multiple processors.
- Extended Prototyping Capabilities
  Proto+ addresses both functional and behavioral prototyping. In addition, a more flexible and realistic data model is provided for capturing the data objects of a prototype system.
- Prototyping Language Extensions
  The prototyping language supports multiple behavior rules, a richer set of C^3I modeling primitives, and support of multiple execution threads.
- Reuse
  Since reuse support is a key objective of Proto+, the Proto+ methodology should provide significant guidance in defining, using, and tailoring reusable components.
- Integration with REE/RIP and Analyst

The methodology will assume that a systems analyst is working in an integrated environment that includes CORE Analyst (*The Analyst: An Expert Systems Approach to Requirements Analysis*), REE/RIP and Proto+. The methods of Proto+ will address when and how each of these tools contributes to the definition of a complete prototype system.

4.2 Explanation of Software Specification Language (SSDL)

SSDL provides the formal foundation of Proto+. A software system specification is defined as a hierarchical dataflow graph consisting of process nodes, data stores, communication connections, and ports (interface between process nodes and connections). Each process node in the hierarchy may be further defined by a refinement represented by a dataflow graph (also known as a subgraph). The subgraph receives all inputs and sends all outputs to ports of the enclosing process node (parent process node). The behavior of a leaf process node (non-refined node) is defined through a high-level structured language that includes support for assignment, condition, and iteration.

A complete prototype is defined by three related representations of a software system. These three interdependent representations include:

- Functional prototype - consists of a hierarchical dataflow graph that identifies the data transformations and flow of control.
- Behavioral prototype - captures the detailed transformation of inputs to outputs at each leaf process node.
- Information model - includes the definition of the data objects required and processed by the process nodes and store nodes.

A functional prototype identifies the process nodes that perform system functions. Data is transient and flows between process nodes over the connections as messages. Messages represent transient data and data stores define state data (maintained between process node executions) that may be accessed by the process nodes. Two types of data exist in SSDL - activation data that flows over the connections arriving at the process node's input ports and reference data that resides in data stores and may be accessed by process nodes through access connections.

The detailed behavior or computation of any of the leaf process nodes (non-refined) is defined through a set of behavior rules. A behavior rule defines the data transformation in which inputs are transformed into outputs.

SSDL supports an object-oriented information model. Objects include attributes that define characteristics. In addition, objects may inherit attributes from other objects. Typically, an end-user defines the data objects of a software system through a set of classes and subclasses where the data objects represent instances of the classes and subclasses. The attributes of the data objects are modified during prototype execution by the process node behaviors. Through the use of objects, an end-user defines the real-world objects (person, airplane, account) manipulated by the prototype.

SSDL supports three kinds of value holders: ports, data stores, and local variables. By default, value holders are type-less and can assume values of any type. End-users can also impose type constraints on value holders; type checking is done dynamically during interpretation. Data stores are the only value holders that preserve state across process node executions. Ports are denoted as being either input ports or output ports. Input ports have attached first-in-first-out (FIFO) queues that

store incoming data. Ports are visible to their containing process node and a process node's decomposition. Visibility of local variables is strictly limited to their containing process node.

The basic computation model is one of concurrent, cooperating processes that is message driven. The model allows addressing of timing issues and behavior issues as well as functional prototyping.

Message passage between process nodes is defined through connections. Several types of connections are defined for modeling different message processing protocols. Asynchronous communication between process nodes is defined through stream connections. Synchronous communication is supported through synchronized connections. During execution, the Interpreter generates acknowledgments to messages arriving over synchronized connections. Finally, a sample-and-hold communication protocol is provided through sample connections. A sample connection includes a single element buffer that is overwritten by each new message arriving at the process node over a sample connection.

A behavior of a process node consists of one or more behavior rules that are dependent upon input ports. A behavior rule includes a trigger (guard) and an action. The general structure of a process node behavior is defined as follows:

```
initial
   -- Port queue and data store initialization.
end initial

behavior
   <trigger1> :- <action1> !
   <trigger2> :- <action2> !
   ...
   <triggerN> :- <actionN> !
end behavior
```

If a trigger expression is empty, the behavior rule is an independent behavior, otherwise, the behavior rule is a dependent behavior. Independent behavior rules do not depend on input data and are always eligible to execute. Conditional trigger expressions for dependent behaviors may be constructed with the following forms:

```
accept <input port name>

<trigger expression> or <trigger expression>

<trigger expression> and <trigger expression>
```

Triggers specify how data messages are accepted, or received, by a process node (**accept** operations) and the input data conditions under which behavior rule is activated. Even though the triggers of multiple behavior rules within a process node may be true simultaneously, at most one behavior rule within a process node can be executing at any given time. In this manner, use of the behavior rule construct allows specification of non-deterministic selection of alternative actions, which subsequently influences scheduling decisions during interpretation.

High-level SSDL constructs for specifying behavior rule actions include iteration, conditional statements, assignment, message-passing primitives, value holder accesses, and data object interfaces. Behavior constructs are also available to initialize port queues and data stores, and provide access to procedures written in C. Built-in functions are available for displaying data values, delaying behavior execution an arbitrary number of simulation time units, and accessing the global simulation clock. A functional interface to a comprehensive math library is also provided.

4.3 Proto+ Architecture and Dataflow

Figure 4-1 identifies the various tools and components that comprise Proto+. The Proto+ tools are supported by several utility classes and COTS components. Support for object management is provided by the ONTOS object manager. The user-interface manager (UIM) provides support for all graphical and textual user interface services required by the tools. A two-dimensional graphics package which supports all the graphics manipulation is provided through a class named GO. The Xview Toolkit provides support for all window management. The Proto+ window management protocol is based upon the Open Look UI protocol.

Figure 4-2 provides a description of the dataflow between the Proto+ components. In general three types of objects are manipulated by the tools and managed by the Object Manager. The editors define and edit the objects. The Interpreter manipulates the objects during execution of a prototype. Storage and management of the objects are provided by the Object Manager. The UIM collects all inputs and distributes the inputs to the appropriate tools. The UIM also manages the multi-window display screen. The set of objects manipulated by the Proto+ tools include:

- SSDL Objects -Semantic objects, dataflow connections, and parsed behaviors
- Visual Objects - Graphical topologies and geometries
- Schema Objects - Object-oriented data models

4.4 Proto+ Tool Descriptions

The following section describes the function of the individual Proto+ tools. The set of tools include:

- Project Management Tool - create, delete, rename, and select Proto+ projects. A project in Proto+ includes the top-level graph, all of its subgraphs, behaviors, and local data objects
- Graph Editor - create the functional prototype
- Behavior Editor - define the data transformation through behavior rules
- Data Editors - define the classes, subclasses, instances representing real-world objects. Also define local variables used by process node behaviors
- Reuse Facility - define, search, insert, and extract reusable components (process nodes or process node refinements)
- Dynamic Loading Tool - define C function descriptors that can be referenced by process node behavior rules and executed during interpretation
- Interpreter - execute, set breakpoints, and debug prototypes
- Object Manager - manages (stores and retrieves) all Visual, SSDL, and Schema Objects
- User Interface Manager - provides support for all user interface capabilities.

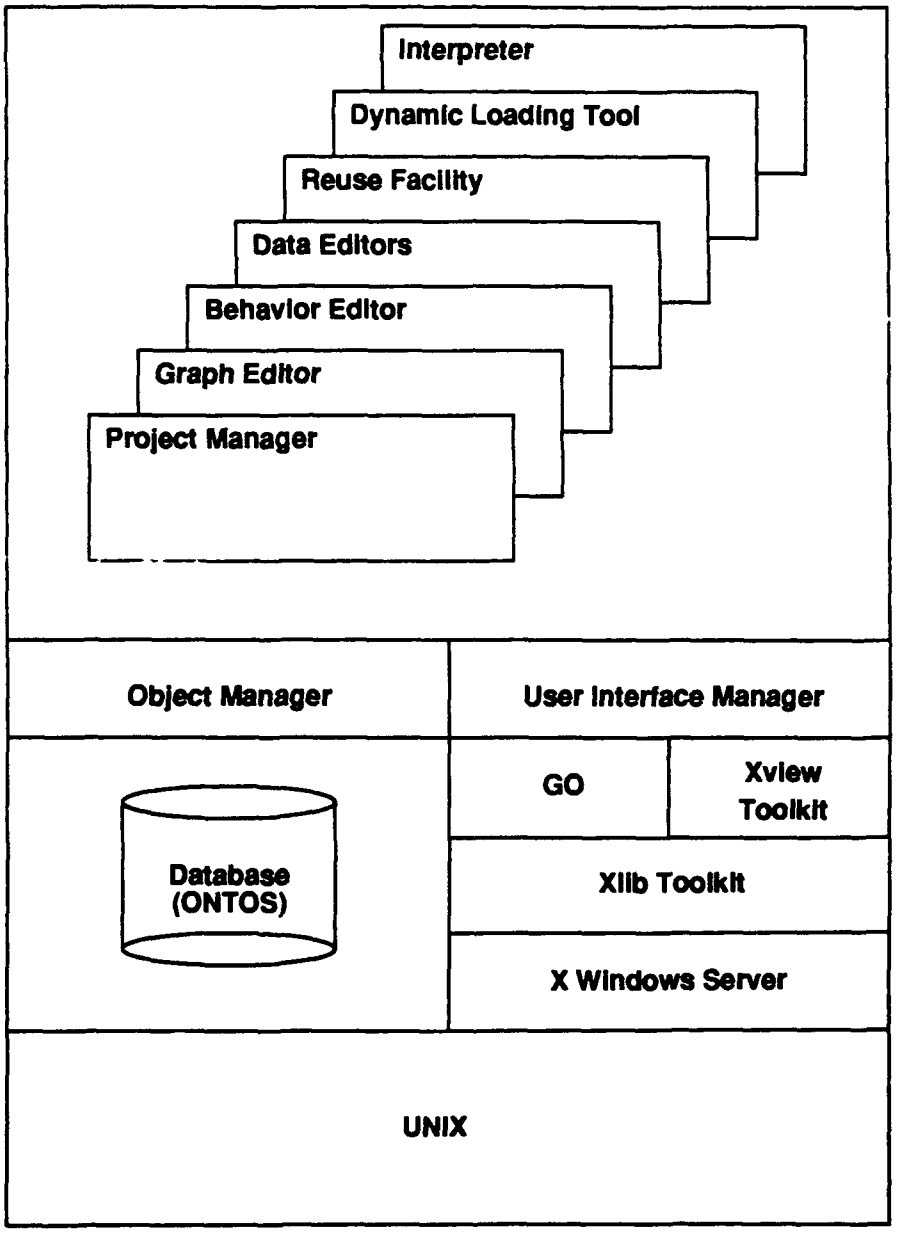


Figure 4-1. Proto+ System Architecture

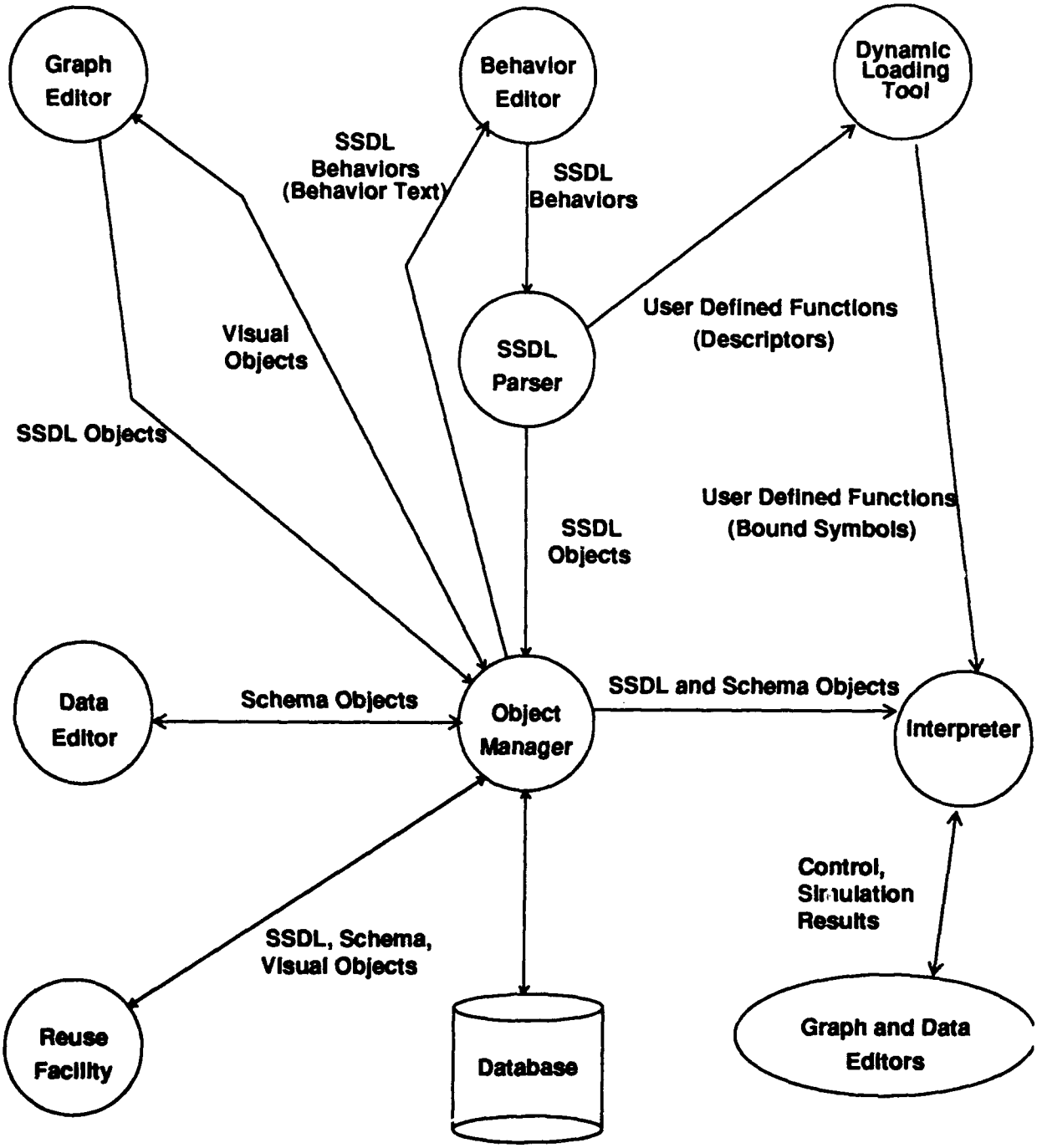


Figure 4-2. Proto+ Dataflow Diagram

4.4.1 Project Manager

The Project Manager provides support for creating and managing projects. A project in Proto+ consists of the top-level graph (highest level), all subgraphs providing refinements of the top-level process nodes, all behaviors, and all temporary data objects required by process nodes. Through the Project Manager tool an end-user creates, renames, deletes, and selects projects. In addition, changes to all projects are saved to the database through the Project Manager. An end-user also invokes other tools through the Project Manager (Graph Editor and Behavior Editor). Figure 4-3 depicts the Project Manager tool window.

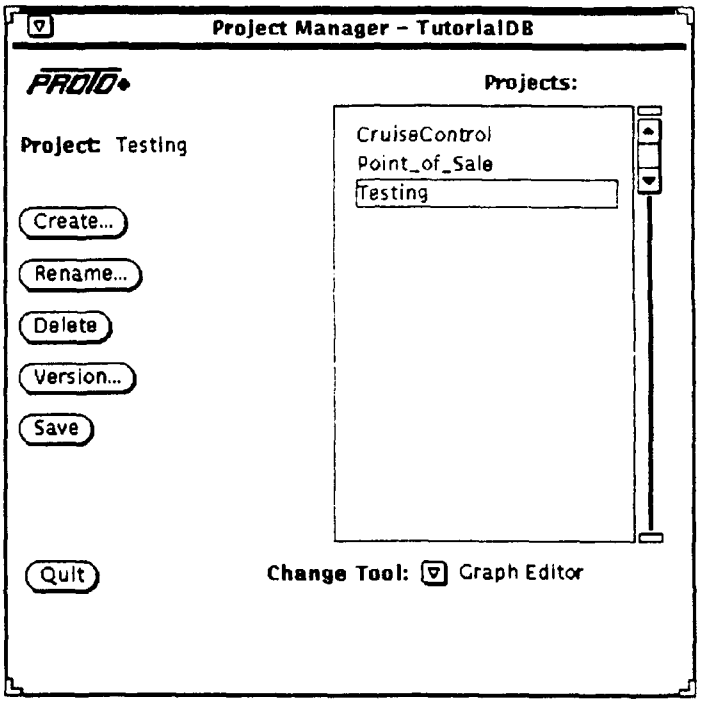


Figure 4-3. Proto+ Project Manager

4.4.2 Graph Editor

The Graph Editor supports the definition of the dataflow portion of a Proto+ prototype. Through the Graph Editor, an end-user defines the process nodes, data stores, connections, ports, and process node refinements (subgraphs). The Graph Editor includes a primary window through which the list of current opened views of a project are listed (see Figure 4-4(a)) and the Graph Editor canvas window (see Figure 4-4(b)). Through the primary window of the Graph Editor the end-user accesses other tools including the Reuse Facility, the Dynamic Loading Tool (Load User Functions, and Data Editors. Through the canvas window of the Graph Editor, the end-user defines the graphic objects that represent the functional prototype (process nodes, data stores, ports, and connections). The Graph Editor also supports the definition of a process node refinement. The capabilities supported by the Graph Editor include cut, copy, paste, panning, zooming, moving, editing, and deleting graphic objects. Figure 4-4(a) illustrates the primary window and Figure 4-4(b) illustrates the Graph Editor canvas window with several graphic objects defined.

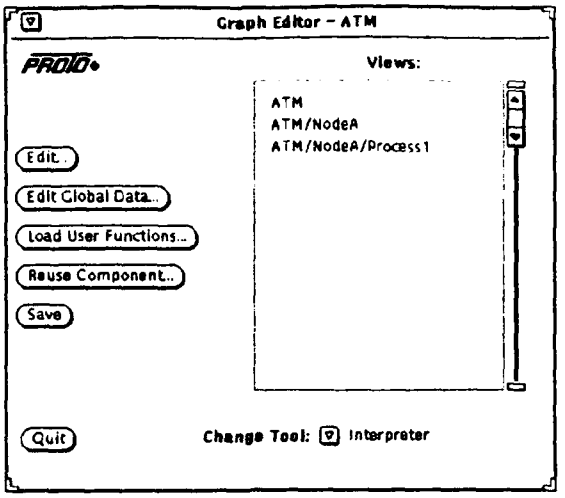


Figure 4-4(a). Graph Editor

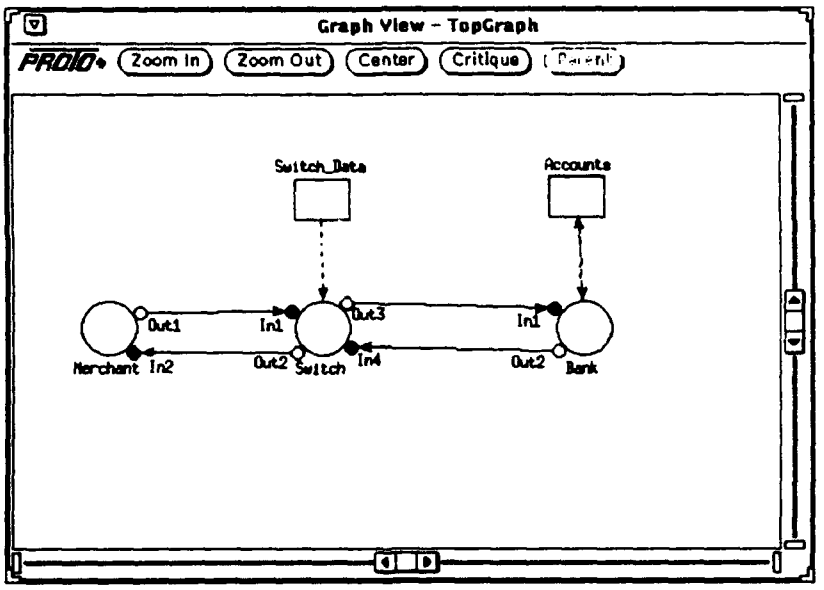


Figure 4-4(b). Graph Editor Canvas Window

4.4.3 Behavior Editor

The Behavior Editor supports the composition of the set of behavior rules defined for a leaf process node (non-refined). The Behavior Editor is primarily a text editor that includes behavior construct templates to assist in the definition of the behavior statements (supply, store_send, condition, and iteration statements). The Behavior Editor includes a parser through which an end-user verifies the syntactic and semantic correctness of a set of behavior rules. The semantic checks include verifying that references to unknown objects (data objects, ports, and data stores) are not included in the behavior. The Behavior Editor provides access to built-in and user-defined functions that have

been dynamically linked with Proto+ (see section 4.4.6 on the Dynamic Loading Tool). Figure 4-5 illustrates the interface of the Behavior Editor in which a pair of simple behavior rules have been defined and checked.

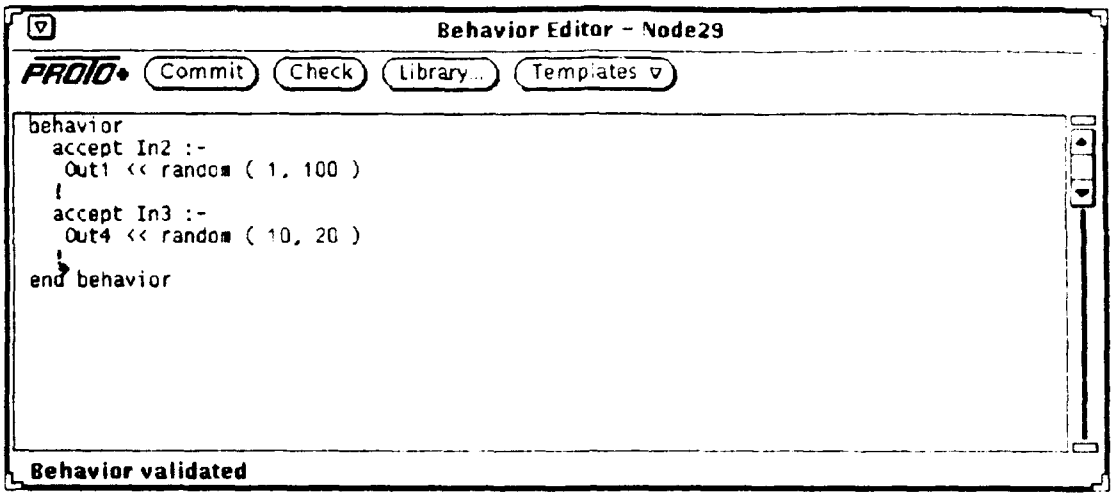


Figure 4-5. Behavior Editor with Multiple Behavior Rules

4.4.4 Data Editors

Through the Data Editors an end-user characterizes the information model. The information model is defined through schemas that include classes and instances. Classes and instances support the representation of real-world objects (person, airplane, account) manipulated by the prototype. Characteristics of objects are defined through attributes. The types supported for attributes include integer, real, string, n-dimensional arrays, and references to other instances. Also through the Data Editors, the end-user may define temporary variables required by process node behavior rules. Figure 4-6 illustrates the interface of the Data Editors. In this figure classes are uncapitalized and instances are capitalized. For example, "city" is a class and "Cleveland" is an instance.

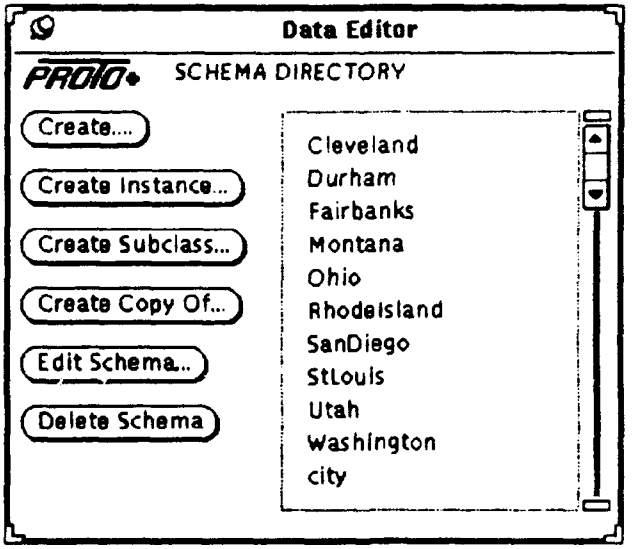


Figure 4-6. Data Editor Window

4.4.5 Reuse Facility

The Proto+ Reuse Facility supports the definition, searching, copy/paste, cut/paste of reusable components between reuse libraries and an opened Proto+ project. A reusable component includes a description, a process node, its behavior, ports, local variables, and any refinements (i.e. subgraphs including process nodes, data stores, ports, and connections). Prior to inserting a reusable component in a reuse library, the end-user must define a description for the process node. The description is subsequently used in searching for an eligible reuse component. A reusable component is identified either directly by listing all of the components in a reuse library or by searching for a component that matches a search string (key words) with a degree of accuracy (percentage of key words that must be matched to be an eligible component). To use a reusable component in a Proto+ project, the component must be copied into the paste buffer. The graph or subgraph in which the component is to be inserted is then opened and the component is pasted into the current editing context.

Through the Reuse Facility, an end-user may open reuse libraries, store components into a reuse library, extract components from a reuse library, and delete components from a reuse library. Figure 4-7 illustrates the interface of the Reuse Facility.

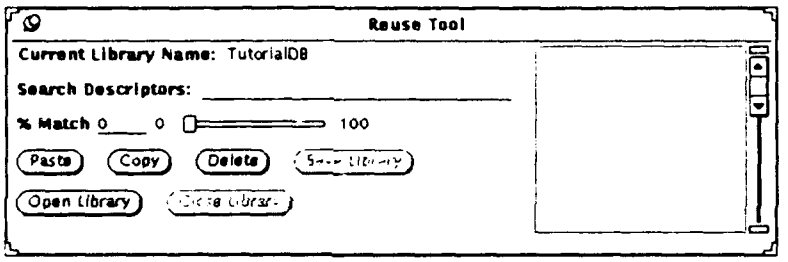


Figure 4-7. Reuse Facility Window

4.4.6 Dynamic Loading Tool

Through the Dynamic Loading Tool an end-user may dynamically bind and link compiled C functions with the current Proto+ tool session. The end-user does not have to exit Proto+ or recompile the tool to include the external functions. An external function is characterized by a function description that includes the name, object code location, function return type and parameters. Once the external functions have been described, the load command links and binds the external functions with Proto+. The end-user may now reference the external functions from any process node behavior. The list of available external functions can be accessed through the Behavior Editor's library command. Figure 4-8 illustrates the interface of the Dynamic Loading Tool.

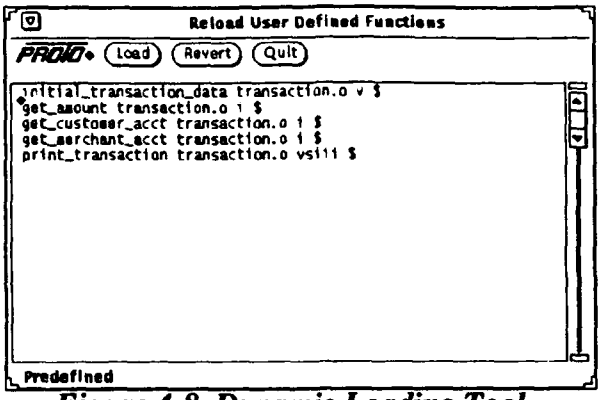


Figure 4-8. Dynamic Loading Tool

4.4.7 Interpreter

The Interpreter provides support for the execution, setting of breakpoints, and debugging of Proto+ systems. The Interpreter animates the prototype by highlighting process nodes as a process node executes and highlighting the connections as messages flow over the connections. Queues associated with the input ports that identify the number of messages waiting to be processed are also visible during interpretation.

The Interpreter also provides debugging capabilities to assist end-users in analyzing prototype execution behavior. An end-user may set a variety of breakpoints and at breakpoints may examine any of the objects in the canvas window to determine the object's current state. Upon completion of Interpreter execution, performance statistics are output to the screen. The performance statistics include process statistics (busy time, idle time, and number of process node executions), processor statistics (busy time, idle time, and behavior rule statistics), and simulation statistics (elapsed simulation time, number of simulation events, and elapsed CPU time in seconds). Figure 4-9(a) illustrates the Interpreter window and Figure 4-9(b) depicts the view of the window as the prototype executes.

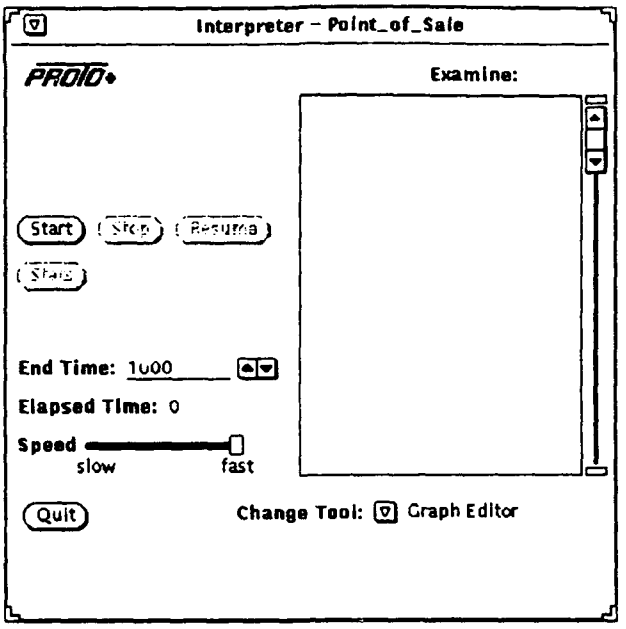


Figure 4-9(a). Interpreter Window

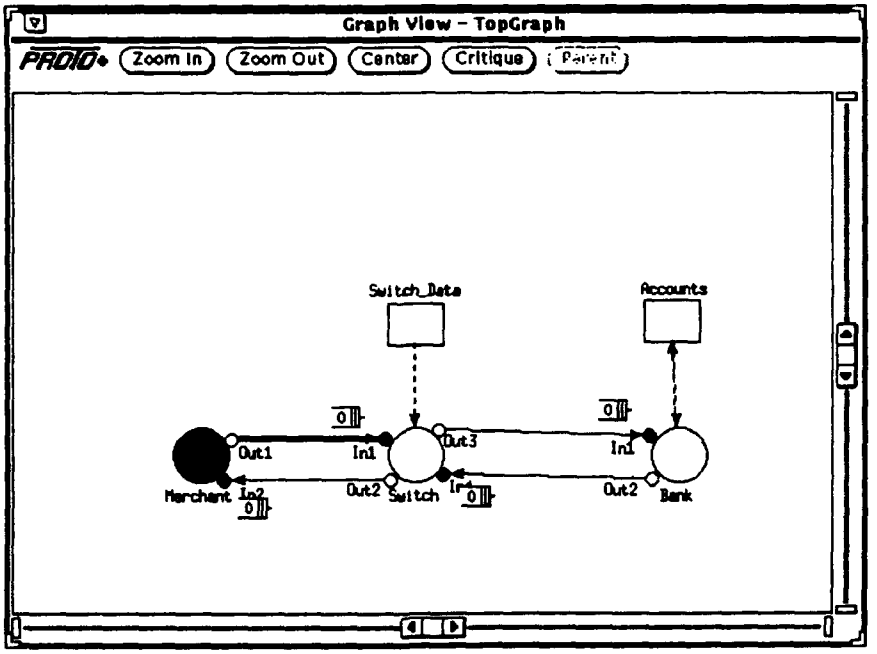


Figure 4-9(b). Interpreter Execution

4.4.8 User Interface Manager

The UIM collects user inputs and distributes them to appropriate tools. The UIM also manages the multi-window display screen. A clean separation between user-interface functions and appli-

cation functions that operate on application data is maintained. During a Proto+ session, all inputs either come from the database or interactive user input from either a keyboard or a mouse. User interaction follows the object-then-action paradigm. With this paradigm, the user is presented with a display containing a collection of objects. A mouse is used to select the desired object, and the system responds by presenting a menu of actions that are acceptable for the selected object. The user then selects the desired action from the menu. Mandatory keyboard use is primarily restricted to entering alphanumeric data strings required by Proto+ tools such as graphic object names, reuse keywords, process node behaviors, and attributes and values of data objects.

Outputs usually are written to the database or presented to the user on a display screen. Outputs to the screen are in the form of graphical displays inside windows, menus, and alphanumeric displays. Descriptive graphical depiction of data is preferred; textual displays are limited to situations where text has served as user input and where text is the most natural notation for presenting output (error messages). File input and output is available from several Proto+ tools (Behavior Editor and Interpreter).

4.4.9 Object Manager

Proto+ incorporates an object-oriented database, or object base, to manage all persistent objects including SSDL objects, Visual Objects, data objects, and behaviors. The Object Manager provides an application-level interface for controlled access to the database.

4.5 Summary of Methodology

A requirements prototyping methodology was developed by the Proto+ contract. The methodology addressed the use of CORE Analyst, Proto+ and REE/RIP. The key steps of the Proto+ methodology included:

- With the support of CORE Analyst develop a domain model that describes the system from a variety of viewpoints.
- Develop a functional decomposition that identifies the major subsystems of the system.
- Perform an initial allocation of requirements, constraints, and functions to the subsystems identified through the functional decomposition.
- Identify the most crucial or least understood requirements, constraints, and functions and analyze these aspects of the system to determine which elements of the system to first prototype.
- Develop a prototyping plan organized around specific prototyping increments.
- Estimate time and resource constraints for each prototyping increment paying particular attention to the first increment.
- Review the plan with the end-user.
- Select and perform one of the prototyping increments.
- Review the results of the prototyping effort to answer original questions and identify new questions.
- Present the prototype to the end-user and either iterate the current prototyping increment if questions must still be answered, or continue with the next prototyping increment.
- Continue selecting, planning, and executing prototyping increments until either all questions

are answered, or schedule and resource limits have been reached.

Two aspects of the methodology were developed under the Proto+ project. The first addressed at a high level the entire requirements prototyping life cycle and provided guidance on when to use the RET tools. The second specifically addressed the requirements prototyping tasks supported by Proto+. In this section of the methodology, details on the use of the Proto+ tools and defining and using reusable components were provided.

The Proto+ requirements protoyping methodology is documented in "Proto+ Methodology" [h].

4.6 Summary of User's Manual

A draft and final version of the Proto+ User's manual were developed during the course of the project. The draft of the user's manual included a step-by-step procedure for developing a prototype. The primary weakness of the first draft was that the document was not adequate as a reference manual. In addition, end-users, although able to complete the tutorial, were not able to easily identify correct sections in the manual to answer specific tool questions.

The final version of the Proto+ User's Manual addressed these weaknesses. A short tutorial of approximately twenty pages was included. The tutorial illustrated eighty per-cent of the tool capabilities. To simplify execution of the tutorial, a seeded Proto+ database was delivered with the final version of the user's manual. Many of the components required by the tutorial were predefined (classes, instances, and external functions) and included in this database. To completely address the needs of a new Proto+ user, the user's manual included installation procedures and reference material on each of the tools.

4.7 Summary of Training Courses

Training courses were provided at each increment delivery of the Proto+ tools. The first training session involved very little hands-on interaction with the tool. A significant amount of training was devoted to explaining the OpenWindows environment, registration of databases, and the conceptual models supported by each of the Proto+ tools. Several in-depth discussions occurred as a result to the first training course. These discussions resulted in changes to the interface and capabilities of Proto+. A oversight on the part of the developers was not providing a clear mapping between existing Proto capabilities and how Proto+ provided the similar or enhanced capabilities.

The second and third training courses devoted an increased amount of time to hands-on interaction with the Proto+ tools. Less training time was devoted to the user interface and basic Proto+ capabilities since end-users had worked with previous versions of the tool.

The objective of the final training course involved training end-users with no experience with Proto+. The final training involved a complete explanation of each of the Proto+ tools. The final training also required that the end-users develop several small examples. Through the set of examples all of the capabilities and tools of Proto+ were exercised.

5. Conclusions

This section summarizes the conclusion of the Proto+ project in the areas of technology advancement and specific areas for further research and development.

5.1 Areas of Technology Advancement

The primary areas that the Proto+ advanced CASE tool and software engineering technology is the area of executable specification languages. Proto+ provides a highly graphic-based model through which software systems are specified. Representing systems through a dataflow, functional decomposition model is quite easy with Proto+. In addition, the ability to model multi-threaded systems is a significant strength of Proto+.

Proto+ provides an effective, graphical, high-level language for defining software system prototypes. The capability to dynamically link, bind, and execute external functions without exiting the Proto+ tool represents another technical strength of Proto+. Proto+ can be viewed as both a requirements prototyping tool and also an integration tool through which tested code modules can be combined and transformed into a software system using a software building blocks strategy.

Proto+ also included enhanced support for reuse. Reuse support has been integrated with the Graph Editor. In addition, Proto+ now supports multiple reuse libraries. Flexible mechanisms for searching and identifying eligible reuse components have also been provided in Proto+.

5.2 Specific Areas for Further Research and Development

The specific areas that should be addressed by subsequent R&D efforts that would improve or enhance Proto+ tools and capabilities include:

- Implement timing constraints and constructs in SSDL.
- Develop a more flexible data model supporting the definition of classes. End-users should be able to specify the scope of classes and instances.
- Ensure that the tools are not coupled to a specific database.
- Address project management issues.
- Ensure that current or enhanced Proto+ tools can scale up to support large software system development.
- Integrate Proto+ with other prototyping and analysis tools including user-interface prototyping tools and performance analysis tools.

U.S. GOVERNMENT PRINTING OFFICE