An Environment for Functional and Performance Prototyping of Parallel Programs

Ramón D. Acosta and Adolfo Guzmán

International Software Systems Inc.
9430 Research Blvd., Echelon IV, Suite 250
Austin, TX 78759

Telephone: 512-338-5719
Fax: 512-338-5757
Email: acosta%issi.uucp@cs.utexas.edu

Abstract

Parallel Proto (PProto) is an environment for constructing prototypes of parallel programs based on functional and performance modeling of dataflow specifications. The system supports codesign and analysis of high-level software and hardware architectures. Facilities provided by PProto include a visual language and an editor for hierarchical dataflow graphs, a resource modeling tool for creating parallel architectures, mechanisms for mapping software components to hardware components, an interactive simulator for prototype interpretation, and a reuse capability. The simulator contains components for instrumenting, animating, debugging, and displaying results of functional and performance models. The PProto environment is built on top of a substrate for managing user interfaces and database objects to provide clear and consistent views of design objects across system tools.

In 1991 Workshop on Hardware/Software Codesign, 13th International Congress on Software Engineering, Austin, TX. MAY 1991.

1 Introduction

Advances in computational power and availability continue to increase the attractiveness of employing parallel architectures in high-performance applications. Unfortunately, architecting of software systems to exploit the performance afforded by these processors remains a difficult challenge for software engineering. It is at the architectural design levels where design decisions can have significant impact on the eventual functionality and performance of parallel systems. Thus, methodologies and tools for analyzing prototypes during early phases of the specification and design process are critical to meeting system requirements.

Parallel Proto (PProto) is a computer-aided software engineering (CASE) environment that aims to overcome difficulties associated with building distributed and parallel software systems by supporting codesign and analysis of high-level software and hardware architectures [Acosta 1990, 1991]. Using prototyping as its primary methodology, PProto addresses the following areas:

- **Specification and Design of Software Prototypes.** Rapid system design is achieved with a visual hierarchical dataflow language that includes facilities for object-oriented data modeling.
- **Architecture Modeling.** A graphic editor is provided for building resource models of many types of MIMD (multiple-instruction, multiple-data stream) machines, including shared-memory and distributed-memory systems. Facilities are available for mapping of software (logical) components to hardware (physical) components.
- **Prototype Execution Using Simulation.** Subsystems to support execution of prototypes include an interpreter, a scheduler, and an architecture modeler. Additional simulation support includes tools for interactive debugging, functional animation, and instrumentation for functional and performance analysis.
- **Design Reuse.** Management and browsing of hierarchical libraries containing reusable specifications and designs is integrated with the system editors.
- **User Interface Management.** Interfaces to editing and simulation tools are managed by a common graphical interface substrate. The user interface manager guarantees a clear and consistent interface across tools.
- **Object Management.** Management of persistent objects is implemented by an object-oriented database commonly accessible to all system tools.

PProto is being developed through a set of modifications and enhancements to Proto+, a prototyping tool for specification and design of software systems based on dataflow concepts [Hartman 1989]. Proto+, in turn, extends the functional prototyping capabilities of the RADC Proto System [Konrad and Hartman 1988] with a model for concurrent processing and communication that provides control over scheduling of operations.

2 Parallel Programming Environments

Current and future advances in multiprocessor technology are expected to increase the need for software engineering environments for parallel and distributed programming. These environments will exploit the computation power of multiprocessors by assisting analysts in explicit specifica-

tion of parallelism, debugging of multiple processing threads, and evaluation of performance and efficiency.

Several research tools for evaluating the functionality and performance of parallel programs have concentrated on low-level measurements of implemented systems [Lehr et al. 1989; Guarna et al. 1989; Miller et al. 1990]. Using invasive procedures, dynamic statistics are accumulated and graphically displayed with visualization techniques. Although this approach can be employed effectively for tuning parallel algorithms, it is inadequate for analyzing high-level architectural tradeoffs.

General purpose simulation tools can also be employed for building parallel systems [Russell 1987; SES 1989]. Use of such tools, however, requires the analyst to handle many more of the generic software/hardware modeling and performance analysis aspects of system design, in addition to domain modeling tasks associated with application specification.

The support for higher-level visual program abstractions of PProto is closer in scope to that of [Yan 1988; Browne et al. 1989; Pease et al. 1991]. These systems contain facilities for concurrent problem formulation, resource management strategies, performance evaluation, and graphical display of results.

3 Prototyping Parallel Programs with PProto

PProto is a tool used by system analysts to accomplish the following:

- Create functional specifications.
- Refine functional specifications into interpretable functional prototypes.
- Interactively debug functional prototypes.
- Execute the functional prototypes before knowledgeable end users to validate the proposed functionality in the context of target systems.
- Construct models of parallel machine architectures.
- Evaluate the performance of different mappings of functional prototypes onto parallel machines.

The system provides a specification environment for codesign of software and hardware architectures that incorporates a graphical user interface, several editors, an object-oriented database, a reuse facility, and an interactive simulator. These tools support construction of specifications using hierarchical dataflow graphs, models of parallel and distributed machine architectures, object-oriented data models, and mappings of software components to hardware components. Figure 1 gives an overview of the PProto methodology based on these prototyping concepts.

PProto is built on top of the UNIX™ operating system using C++. Database management services are provided by the *object manager*, which is built on top of a commercial object-oriented database. The *user interface manager* (UIM) supplies graphical and textual interface services to most system tools. This manager is built on top of the X Windows Server, which provides low-level, device-independent network graphics services.

Figure 2 shows a high-level dataflow diagram of PProto, including the editing, object management, and simulation subsystems of PProto. The editors generate one or more different kinds of

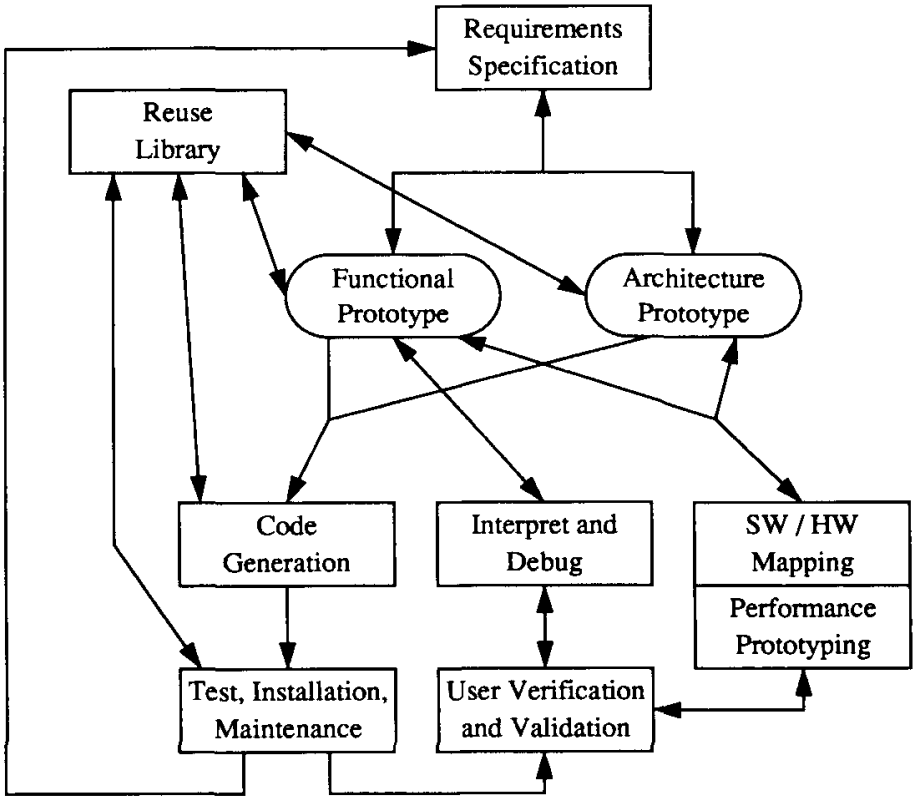


Figure 1. PProto Methodology Flow Diagram

objects that are saved in the database through services provided by an object manager. System objects include the following:

- **SSDL Objects.** Dataflow connectivity, behaviors, and software/hardware mappings
- **Visual Objects.** Graphical topologies and geometries
- **Schema Objects.** Object-oriented data models
- **Architecture Objects.** Connectivity and parameters for hardware primitives
- **Library Objects.** Hierarchical reuse libraries.

PProto tools that generate and manipulate data include:

- **Graphic Editor.** Graphical editing of dataflow graphs
- **Behavior Editor.** Textual editing of behaviors
- **Schema Editor.** Menu-based editing of data objects

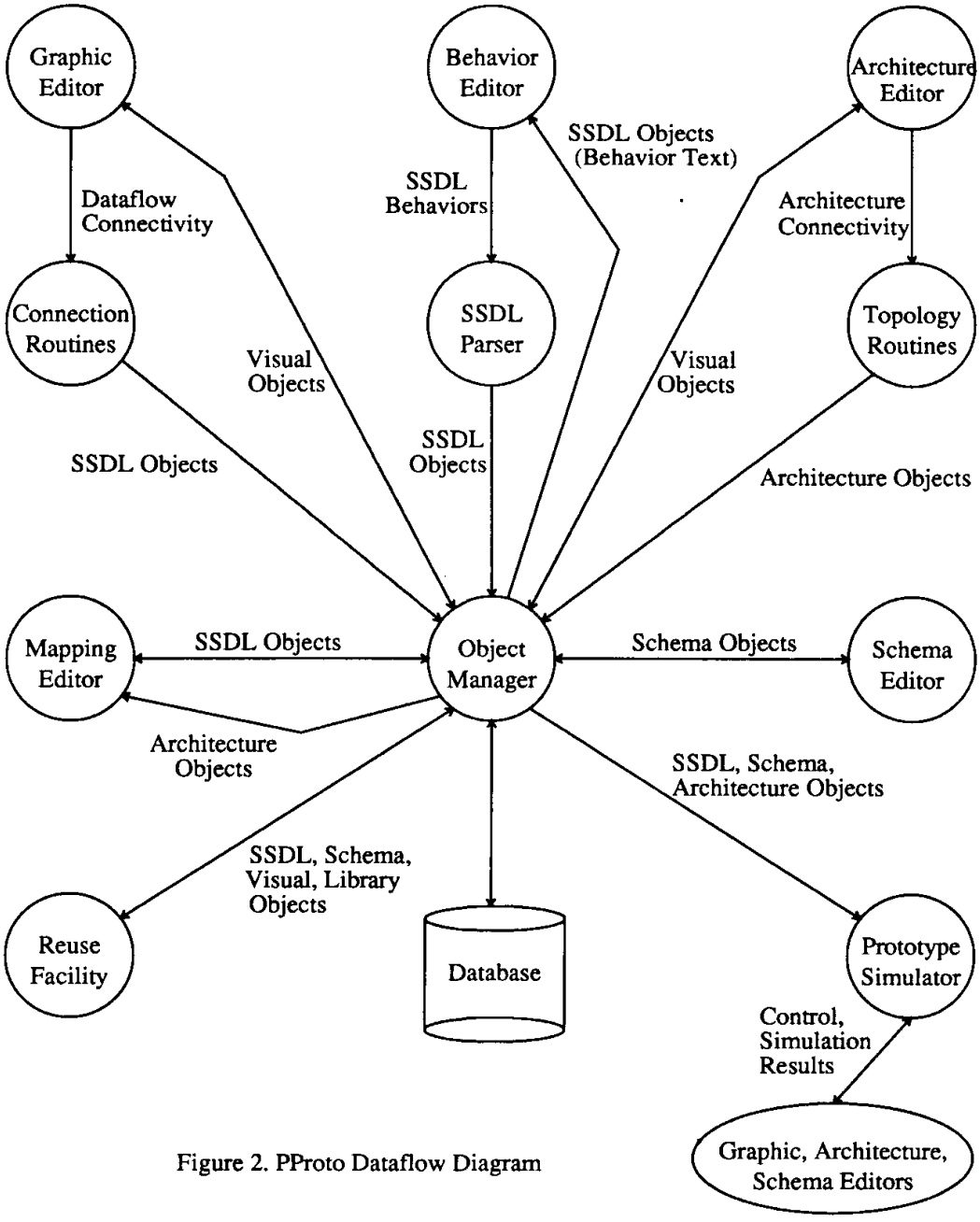


Figure 2. PProto Dataflow Diagram

- **Architecture Editor.** Graphical editing of hardware component topologies
- **Mapping Editor.** Software/hardware component mappings
- **Prototype Simulator.** Interpretation, animation, and debugging of a prototype
- **Reuse Facility.** Access to previously saved library components

3.1 Software Specification

A PProto system specification is a hierarchical dataflow graph consisting of

- **Nodes.** Concurrent processes, functions, operators
- **Data Stores.** Shared memory to store state information
- **Connections.** Communication channels between nodes and data stores
- **Ports.** Interfaces between nodes and connections

Conceptually, a node contains a transformation that is applied to input data whenever the node is activated. This transformation is depicted with either a *behavior* or with a dataflow graph refinement. The behavior of each leaf node of a graph is described with a simple structured programming language whose statements are interpreted during simulation. From behaviors bodies it is also possible to call functions in other languages (such as C). Although nodes are stateless, it is possible to preserve state across node activations by storing data in data stores. Thus, PProto supports functional prototypes incorporating both *message-passing* and *shared-data communication.*

PProto's visual language is called the System Specification and Design Language (SSDL). In the SSDL language, a clear distinction is made between the following portions of a design:

- **Dataflow.** Graphical model of a specification
- **Behavior.** Data transformations achieved by leaf nodes of a dataflow graph
- **Schema.** Object-oriented data definitions

SSDL contains many features that are useful in building systems containing concurrent cooperating processes, including several synchronous and asynchronous message-passing protocols, shared data stores, timing-related constructs, and data objects with dynamic attributes and inheritance. In addition, translation to other programming languages, such as Ada, is possible for the purpose of providing better performance and establishing a path towards construction of target applications.

3.2 Hardware Specification

PProto provides a generic mechanism for specifying several kinds of MIMD architectures. These include shared-memory multiprocessors, distributed-memory multiprocessors, and hybrid machines. Hardware modeling primitives include

- **Processors.** Execute node behaviors
- **Memories.** Contain data store values
- **Buses.** Transfer messages for communication between nodes

User-specified parameters are supplied for selection of hardware resource characteristics, including machine topology, processor execution speeds, memory-access times, and bus delays.

The system includes a simple mechanism for mapping software (logical) components to hardware (physical) components. A mapping consists of pairs of software components (nodes and data stores) and hardware components (processors and memories) taken from their respective definitions. Such a mapping results in an embedding of the dataflow graph of logical connections into the architecture graph of physical connections. The mapping facility also supports default *sequential* and *fully-parallel* mappings. More than one mapping per software specification is possible.

3.3 User Interface and Object Management

A set of integrated, highly interactive tools serves as the framework for building and modifying prototypes. A multiwindow graphical editor enables analysts to easily define and modify hierarchical dataflows. An SSDL text editor and parser facilitates editing of node behaviors. A menu-based editor is available for editing SSDL data object descriptions. Architectural features are described with a graphical architecture editor.

The PProto graphical editing and simulation tools are closely tied to a database object manager, allowing tools to share objects and screen images with consistent interpretations. A component reuse facility for specifications and designs incorporates hierarchical library management and browsing capabilities. This facility is closely integrated with system editors through the persistent object services available from the object manager.

3.4 Simulation

The simulation environment in PProto includes an *interpreter*, a *scheduler*, and an *architecture modeler* for prototype execution. These functions are controlled by a simulation kernel that implements a simulation cycle using a time-based event queue. The simulator uses the graphical interfaces of the graphic, architecture, and schema editors to display simulation results, although prototype editing is disabled during execution.

The interpreter executes all SSDL language constructs contained in behaviors and keeps track of processor execution times. The interpreter also generates events for message-passing and data-store accesses. A time-based scheduler uses software/hardware mappings to arbitrate among multiple nodes competing to execute on a single processor. The architecture modeler manages resource simulation, including message routing, serialization of memory and bus requests, and display of utilization statistics.

Dynamic mechanisms for assisting users in observing and debugging prototype execution include dataflow and architecture graph animation, tailorable instruments (e.g., queue-status), node breakpoints, object display statements, and dynamic simulation interruption. Most objects in the system, such as nodes, connections, ports, data stores, schemas, processors, memories, and buses, can be browsed during prototype simulation. Additional debugging assistance for parallel processing activities is available in the form of a deadlock detection facility.

4 Usage Scenario: Electronic Funds Transfer Example

Several of the parallel programming elements of the PProto methodology of Figure 1 are now illustrated using a simple example involving the architecture and design of a distributed electronic funds transfer (EFT) system. The EFT specification was developed from a set of requirements

adapted from the description of the system in [Staskauskas 1988]. A dataflow diagram of the system, developed using PProto, appears in Figure 3.

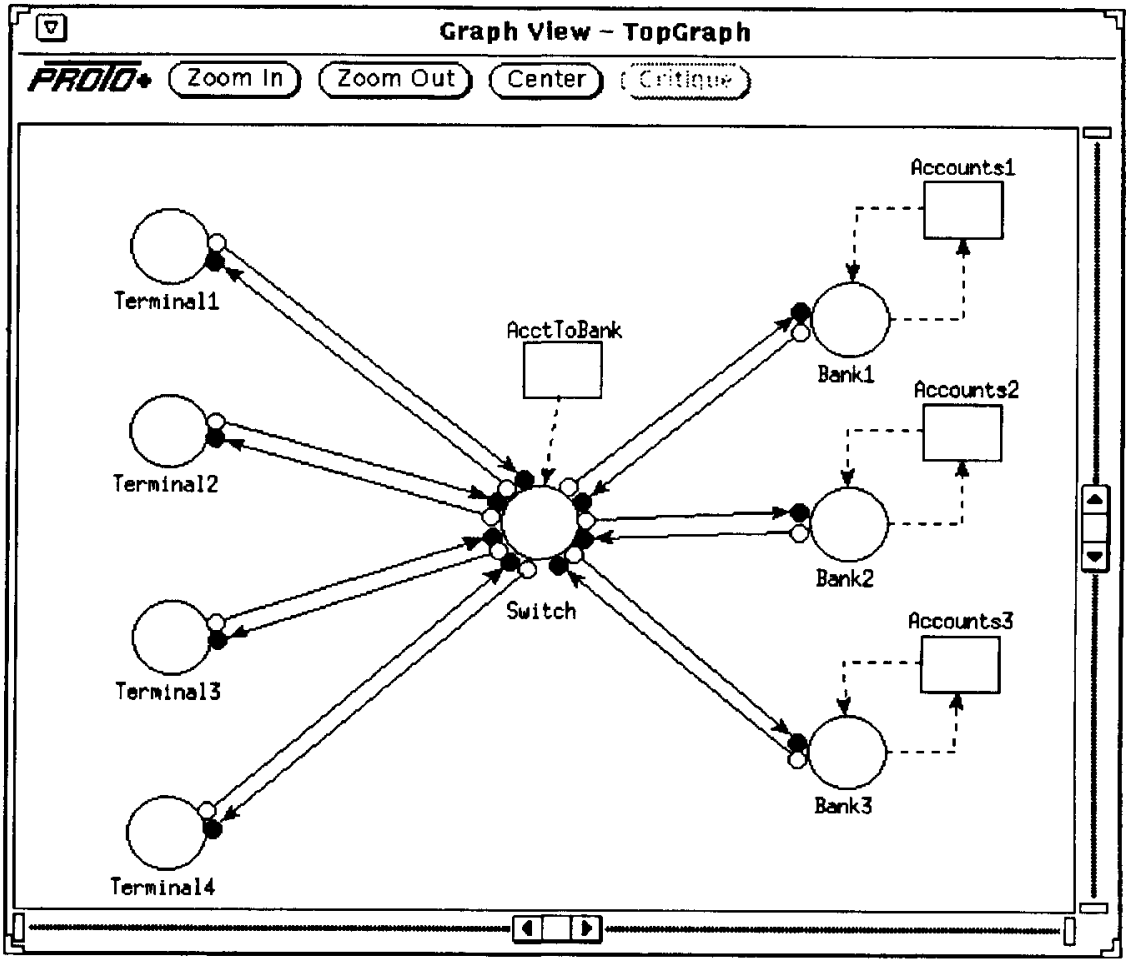


Figure 3. PProto Graphic Editor: EFT Example Dataflow

The EFT system processes *transactions* involving the automatic transfer of funds from *customer accounts* to *merchant accounts*. The following three kinds of nodes and two kinds of data stores are used in the system:

- **Terminal.** Originates new transactions and notifies customer whether transactions are accepted or rejected.
- **Bank.** Debits customer accounts and credits merchant accounts according to transaction requests.

- **Switch.** Routes transactions between terminals and banks according to customer and merchant account numbers and transaction status.
- **AcctToBank.** Array that maps customer and merchant account numbers to banks.
- **Accounts.** List of account numbers and balances contained at a bank.

This example is simple enough that node refinements are not necessary to capture the initial specification being considered. Although not illustrated here, future iterations of the design cycle could functional and performance simulation results to refine and decompose some of the functions (e.g., switch routing) and data structures of the system.

The design of Figure 3 constitutes an executable specification of the EFT system. The PProto simulator is used to verify the prototype's functionality. In addition to the simulation functions implemented by the SSDL interpreter and scheduler, the debugging, instrumentation, and animation facilities of the system can be used to observe architectural characteristics of the system (e.g., message communication, data store accesses) and verify functionality. Transaction data enters the system from the terminals by calling C functions that read data files. It is thus possible to easily exercise the system for different data sets.

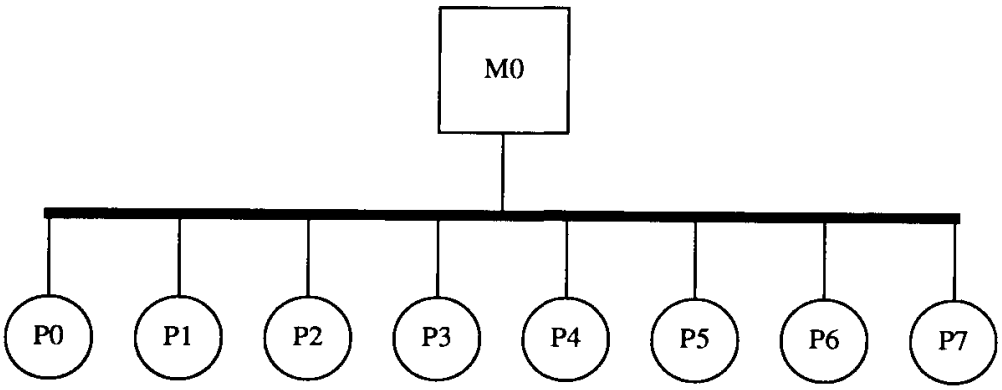
Particularly useful is the ability to automatically assign each node to a different processor using a fully-parallel mapping. The model obtained by this approach is an ideal distributed EFT system in which communication and memory access costs are eliminated. Not only does this allow verifying the functional capabilities of the system, but simulation results can be used to suggest target architectures and software/hardware mappings that are best matched to the EFT architecture. Also, such prototyping helps identify serial bottlenecks that can be reduced through further functional changes.

Performance simulation continues with mapping of the EFT software specification to one or more hardware architecture definitions. Several mappings and architectures are suggested in Figures 4-6. By observing resource utilization and performance characteristics of the combined software/hardware models, analysts can identify advantages and disadvantages of different system architectures in early phases of the specification and design cycle.

5 Conclusions

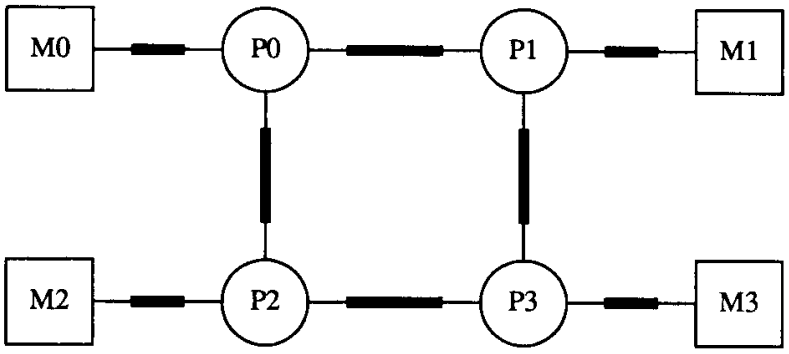
Parallel Proto (PProto) is a computer-aided software engineering (CASE) environment that aims to overcome difficulties associated with building parallel programs by means of rapid prototyping techniques. Using functional and performance modeling of dataflow specifications, PProto assists in codesign and analysis of high-level software and hardware architectures. The system supports mechanisms for specifying scheduling, concurrency, data dependencies, synchronization, and performance characteristics of multiple processing threads.

PProto provides a sophisticated specification and design environment that incorporates a graphical user interface, several editors, an object-oriented database, a reuse facility, and an interactive simulator. The simulator contains components for instrumenting, animating, debugging, and displaying results for functional and performance models. These tools support construction of specifications using hierarchical dataflow graphs, models of parallel and distributed machine architectures, object-oriented data models, and mappings of software components to hardware components.



(Terminal1, P0)
(Terminal2, P1)
(Terminal3, P2)
(Terminal4, P3)
(Switch, P4)
(Bank1, P5)
(Bank2, P6)
(Bank3, P7)
(AcctToBank, M0)
(Accounts1, M0)
(Accounts2, M0)
(Accounts3, M0)

Figure 4. EFT Mapping to Shared-Memory Machine



(Terminal1, P0)
(Terminal2, P1)
(Terminal3, P2)
(Terminal4, P3)
(Switch, P0)
(Bank1, P1)
(Bank2, P2)
(Bank3, P3)
(AcctToBank, M0)
(Accounts1, M1)
(Accounts2, M2)
(Accounts3, M3)

Figure 5. EFT Mapping to Distributed-Memory Machine

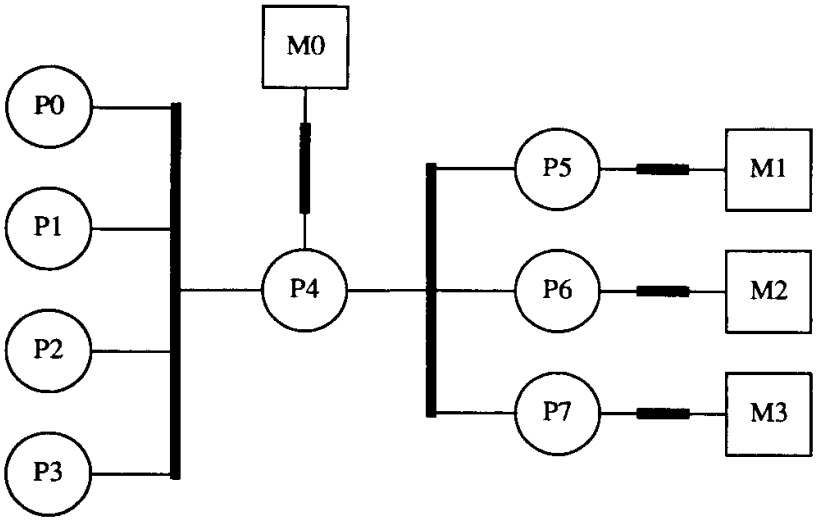


Figure 6. EFT Mapping to Hybrid Machine

6 Acknowledgments

PProto is being constructed under the sponsorship of the Rome Air Development Center (RADC), Command and Control Division (CO), C^2 Software Technology Branch (COEE), Contract No. F30602-89-C-0129.

7 References

R.D. Acosta, "System/Subsystem Specification for Parallel Proto – DRAFT," Technical Report No. ISSI-A89A00002-DRAFT, International Software Systems Inc., Austin, TX, August 1, 1990.

R.D. Acosta, "Simulation of Modeling Parallel Programs in PProto," to appear in *Proceedings of the1991 Summer Computer Simulation Conference*, July 22-24, 1991.

J.C. Browne, M. Axam, and S. Sobek, "CODE: A Unified Approach to Parallel Programming," *IEEE Software*, Vol. 6, No. 4, July 1989, pp. 10-18.

V.A. Guarna, D. Gannon, D. Jablonowsky, A.D. Malony, and Y. Gaur, "Faust: An Integrated Environment for Parallel Programming," *IEEE Software*, Vol. 6, No. 4, July 1989, pp. 20-26.

D. Hartman, "Functional Description for the C^3I Reusable Specification," Air Force Contract No. F30602-88-C-0029, Technical Report No. ISSI-C88A00002-DRAFT, International Software Systems Inc., Austin, TX, June 1989.

M. Konrad and D. Hartman, "Functional Description for Proto," Air Force Contract No. F30602-85-C-0124, Rome Air Development Center, Griffiss AFB, NY, January 1988.

T. Lehr, Z. Segall, D.F. Vrsalovic, E. Caplan, A.L. Chung, and C.E. Fineman, "Visualizing Performance Debugging," *Computer*, Vol. 22, No. 10, October 1989, pp. 38-51

B.P. Miller, M. Clark, J. Hollingsworth, S. Kierstead, S.-S. Lim, and T. Torzewski, "IPS-2: The Second Generation of a Parallel Program Measurement System," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 1, No. 2, April 1990, pp. 206-217.

D. Pease, A. Ghafoor, I. Ahmad, D.L. Andrews, K. Foudil-Bey, T.E. Karpinski, M. Mikki, and M. Zerrouki, "PAWS: A Performance Evaluation Tool for Parallel Computing Systems," *Computer*, Vol. 24, No. 1, January 1991, pp. 18-29.

E.C. Russell, "SIMSCRIPT II.5 and SIMANIMATION: A Tutorial," *Proceedings of the 1987 Winter Simulation Conference*, A. Thesen, H. Grant, W.D. Kelton, Eds., 1987, pp. 102-111.

SES, "SES/workbench Introductory Overview," Scientific and Engineering Software, Inc., Austin, TX, April 1989.

M.G. Staskauskas, "The Formal Specification and Design of a Distributed Electronic Funds-Transfer System," *IEEE Transactions on Computers*, Vol. 37, No. 12, December 1988, pp.1515-1528.

J.C. Yan, "Post-Game Analysis – A Heuristic Resource Management Framework for Concurrent Systems," Ph.D. Dissertation, Technical Report No. CSL-TR-88-374, Computer Systems Laboratory, Departments of Electrical Engineering and Computer Science, Stanford University, Stanford, CA, December 1988.