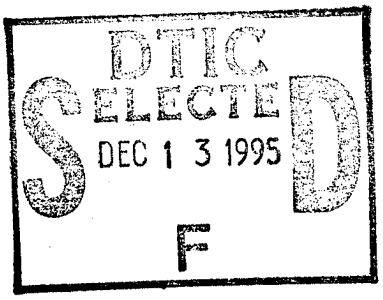
AL/HR-TP-1995-0018

RESEARCH, DEVELOPMENT, TRAINING, AND EVALUATION (RDT&E) SUPPORT: OPERABILITY MODEL ARCHITECTURE



Stephen E. Deutsch
Nichael L. Cramer
Carl E. Feehrer

Bolt, Beranek, and Newman, Inc.
10 Moulton Street
Cambridge, Massachusetts 02138

HUMAN RESOURCES DIRECTORATE
LOGISTICS RESEARCH DIVISION
2698 G Street
Wright-Patterson Air Force Base, Ohio 45433-7604

June 1995

Final Technical Paper for Period March 1993 to December 1994

Approved for public release; distribution is unlimited

ARMSTRONG LABORATORY
AIR FORCE MATERIEL COMMAND
WRIGHT-PATTERSON AIR FORCE BASE, OHIO 45433-7604

NOTICES

When Government drawings, specifications, or other data are used for any purpose other than in connection with a definitely Government-related procurement, the United States Government incurs no responsibility or any obligation whatsoever. The fact that the Government may have formulated or in any way supplied the said drawings, specifications, or other data, is not to be regarded by implication, or otherwise in any manner construed, as licensing the holder, or any other person or corporation, or as conveying any rights or permission to manufacture, use, or sell any patented invention that may in any way be related thereto.

The Public Affairs Office has reviewed this paper and it is releasable to the National Technical Information Service, where it will be available to the general public, including foreign nationals.

This paper has been reviewed and is approved for publication.

MICHAEL J. YOUNG
Contract Monitor

BERTRAM W. CREAM, Chief
Logistics Research Division

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 1995	Final - March 1993 to December 1994

4. TITLE AND SUBTITLE
Research, Development, Training and Evaluation (RDT&E) Support: Operability Model Architecture

5. FUNDING NUMBERS
C - F33615-91-D-0009
PE - 62205F
PR - 1710
TA - 00
WU - 60

6. AUTHOR(S)
Stephen E. Deutsch
Nichael L. Cramer
Carl E. Feehrer

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Bolt, Beranek, and Newman, Inc.
10 Moulton Street
Cambridge, Massachusetts 02138

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Armstrong Laboratory
Human Resources Directorate
Logistics Research Division
2698 G Street
Wright-Patterson AFB, OH 45433-7604

10. SPONSORING / MONITORING AGENCY REPORT NUMBER
AL/HR-TP-1995-0018

11. SUPPLEMENTARY NOTES
Armstrong Laboratory Technical Monitor: Michael J. Young, AL/HRGA, DSN 785-8229

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

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

The Operator Model Architecture (OMAR) project undertook the definition, implementation, test, and validation of an integrated suite of software tools to support the construction of Human Performance Process (HPP) models and the instantiation of the psychological framework previously developed by Bolt, Beranek, and Newman, Inc. (BBN). The project was undertaken in three phases. The objective of Phase 1 was to prepare a Software Functional Specification for the toolbox architecture. The Specification drew on analyses of requirements both to build and to support the testing of computational cognitive models, and on the anticipated requirement to operate within the OASYS simulation environment. During Phase 2, the software design and tools specified in Phase 1 were developed using a rapid prototying approach. Tests demonstrating the functionality of OMAR were conducted near the end of the phase, and an OMAR User's/Programmer's manual was written. In Phase 3, elements of the BBN psychological framework previously developed were instantiated with the aid of the OMAR tools developed in Phase 2, and an Air Traffic Control scenario identified during the earlier effort was used to test the resulting model.

14. SUBJECT TERMS		15. NUMBER OF PAGES
graphical editing and browsing human performance modeling object-oriented simulation	procedural languages	83 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	SAR

CONTENTS

FIGURES

PREFACE

This document presents the rationale for and specification of the integrated suite of software tools that form the basis of a test-bed for Human Performance Process (HPP) model research at the Armstrong Laboratory. It also contains a perspective on the initial use of those tools that was gained during the closing phase of the project, when the software was employed to build, exercise, and test a model.

The reported work, conducted in three phases between March 1993 and December 1994, supplements and extends the results of an earlier delivery order (Delivery Order No. 1: Computational Cognitive Models) administered under the same Air Force contract (No. F33615-91-D-0009) and conducted a year earlier. The latter effort formulated an HPP model based on theoretical and empirical findings in psychology and cognitive science that could be used to support operability investigations of new systems.

The authors are grateful to the contract monitor, Mr. Michael Young of Armstrong Laboratory's Human Resources Directorate (AL/HRG), for his constructive criticism and enthusiastic support of the work. They also extend their thanks to the following members of the Bolt Beranek and Newman Inc. (BBN) OMAR team: Dr. Marilyn J. Adams, Glenn A. Abrett, Dr. Eva Hudlicka, and Jeff Palmucci. Their contributions to the earlier Interim Report (AL/HR-TP-1993-0027) are contained in this report.

RESEARCH, DEVELOPMENT, TRAINING, AND EVALUATION (RDT&E) SUPPORT OPERABILITY MODEL ARCHITECTURE

INTRODUCTION

Goals of the OMAR Project

The goals of the Operator Model Architecture (OMAR) project were the definition, implementation, test, and validation of an integrated suite of software tools to support the construction of Human Performance Process (HPP) models and the instantiation of the cognitive framework developed by Bolt, Beranek, and Newman, Inc. (BBN), in Delivery Order #1 (Deutsch *et al.*, 1993a). To the extent that these goals have been met and future models can be successfully formulated, the resulting system will have made a major contribution to a program that began with a demonstration of the feasibility of combining human operator models and real human operators within a single system simulation (cf. the Automation Impacts Research Test-bed [AIRT] [Corker & Cramer, 1989a; 1989b; 1991]) and continues with the development of the Operability Assessment System (OASYS), an integrated system that will provide system designers with a simulation environment in which operability analyses can be efficiently conducted using both real and modeled operators.

Organization of the OMAR Project

The OMAR project was divided into the following three phases.

Phase 1: Software Functional Specification (4 months): The objective of Phase 1 was to prepare a Software Functional Specification for the toolbox architecture. The Specification drew on analyses of requirements both to build and to support the testing of computational cognitive models, and on the anticipated requirement to operate within the OASYS simulation environment.

Phase 2: Software Development (11 months): During Phase 2, the software design and tools specified in Phase 1 were developed using a rapid prototyping approach. Tests demonstrating the functionality of OMAR were conducted near the end of the phase, and a preliminary version of an OMAR User/Programmer Manual was written.

Phase 3: Demonstration (6 months): In Phase 3, elements of the BBN cognitive framework developed under Delivery Order #1 were instantiated with the aid of the OMAR tools developed in Phase 2, and an Air Traffic Control (ATC) scenario identified during the earlier effort was used to test the resulting model.

The experience gained from these activities will be employed during the first quarter of 1995 to train Armstrong Laboratory personnel in the use of the system.

Organization of the Report

This document presents a comprehensive picture of the work conducted during the three phases. To aid the reader in obtaining an in-depth understanding of the objectives, methodology, and products, this report borrows liberally from an interim report, written at the end of Phase 1. The interim report describes in detail the OMAR Operating Concept, a set of user and system

requirements based upon that concept, and a functional specification for software designed to meet the requirements.

This report is divided into six major sections. The first of these discusses the Concept of Operations for an OMAR system of the future, describing a set of tasks as it might be undertaken by a hypothetical user of tools. The user activities described in the Concept are summarized at the conclusion of the discussion, setting the stage for a review of requirements and specifications.

The second section presents a review of tool-building requirements and specifications for two target architectures: the BBN psychological framework developed in Delivery Order #1 (Deutsch *et al.*, 1993a) and a holon-based architecture developed by Michael Young of the Human Resources Group of Armstrong Laboratory (1993a; 1993b). Each of these is described briefly, and approaches to meeting their special requirements are summarized.

The third section begins with a comprehensive review of software requirements and specifications for software tools associated with the operating concept and target architectures, and concludes with the presentation of perspectives on the functionality required at the interface between OMAR and OASYS, and the interaction of the two systems. Specific areas addressed in this section include:

- two perspectives from which the system architecture proposed for OMAR can be viewed,
- knowledge representation languages, compilers, and simulator,
- direct support for HPP model development,
- knowledge acquisition environment,
- OMAR simulation environment, and
- special concerns related to graphic support for editors, browsers, HPP model performance tracking, and simulation animation.

The fourth section presents detailed descriptions of the major software components and programming conventions developed in the system during Phase 2 and utilized during Phase 3 for model instruction. This section also describes the steps taken to ensure the functional integrity, utility, and understandability of these components and conventions by programmers and modelers. As an aid to understanding the software design decisions, development cycles, and testing methods employed during the generation of code, attempts have been made in this presentation to maintain the basic chronology of events by summarizing the content of activity logs maintained throughout the project.

The fifth section continues with a presentation of Phase 3 development and test of an HPP model. As in the preceding section, it relies on activity logs to give a sense of the development cycles.

The final section presents a brief view of follow-on efforts that, in the designers' judgment, could be conducted usefully to extend OMAR's current capabilities, thereby ensuring its utility both as a stand-alone HPP modeling environment and as a resource during OASYS system simulations.

OMAR'S CONCEPT OF OPERATIONS

Perspective

OMAR is expected to serve two different user communities that are interested in modeling and simulating human behavior. One of these, the "OMAR community," is concerned primarily with formulating and testing HPP models and/or with evolving satisfactory simulations of such models on a computer. The second, the "OASYS community," is chiefly concerned with exploiting validated models of human behavior in order to assess the operability and general adequacy of display and control concepts proposed for human use.

Although they differ in their particulars, the basic requirements for supporting OMAR and OASYS users in the accomplishment of their tasks are quite similar. Both, for example, must enable their users to review the contents of data bases relating to prior modeling and simulation efforts and, where appropriate, to "reuse" elements of earlier scenarios, models, and workstation designs in new or revised formulations. Both must provide means for defining performance measures to be used with upcoming model or simulation runs, and both must aid their users in selecting among alternative displays of simulation "progress" and "output."

This section has two primary purposes. The first is to present the Concept of Operations for a fully developed OMAR environment for human performance modeling. Although the system that has been produced under this delivery order will support user activities in each category of interaction identified in the scenario, optimal design and implementation of the entire set of user aids implied here was beyond the scope of the project. This was due not only to limitations in the resources available, but also to the fact that it is only by observing the efforts of a range of users actually engaged in the use of OMAR tools that ideal designs can be evolved. Such "evolutions" fall, by nature, into the category of "product improvements."

The second purpose is to identify specific user functions that, given the Concept, must be addressed in the design of the OMAR system. The delineation of these functions provides the springboard for detailed specifications of software and hardware functionality that are discussed in the remainder of the document.

The reader is encouraged to refer to the OASYS System Segment Specification (SSS): Baseline with Software Requirements Specification/Initial Requirements Specification (SRS/IRS) (BBN, 1993a) and the OASYS Concept of Operations: Draft (BBN, 1993b) for complementary views of OASYS requirements and specifications.

The OMAR System

BBN's concept of OMAR is that of a collection of coherent software systems that support the development, test, and exercise of HPP models by psychologists, cognitive scientists, artificial intelligence (AI) researchers, and, in general, any persons actively involved in modeling human behavior. While similar to OASYS in its goal of providing a "complete" environment in which a user can construct and evaluate competing "designs," OMAR differs from OASYS by virtue of its emphasis on the use of specialized software tools to formulate engineering models aimed at the "human," rather than the "computer," component of human/computer systems.

One key advancement in OMAR will be the provision of a means for devising HPP models of the human operator that contain receptor, cognitive, and effector modules. These models will be represented by organized collections of smaller models grouped around a rich backbone of simulation tools and task organization heuristics. Researchers interested in devising new HPP models will be able to use the OMAR tool suite to design modules, usually by modifying already

existing modules and, where appropriate, plugging them directly into the remaining modules. This "whole person" simulation will be grounded in psychological reality to such an extent that it will be able to provide researchers with credible data on the psychological soundness of the HPP model they have built.

The basic set of utilities contained in the Concept and the interconnections of these within and outside OMAR are presented in Figures 1 and 2 and discussed in detail later in this report. Essentially, the set consists of:

- the OMAR HPP model-development tool-box,
- one or more OMAR-developed HPP models,
- an OMAR or OASYS model of the target system with which the HPP model is to interact,
- an object-oriented interface between the HPP model and the OMAR target system model or OASYS network protocol,
- a simulator in which to play out the events of a scenario,
- system functions to manage simulation execution, and
- analysis tools to evaluate HPP model performance.

Note that in Figure 1 the outside world is the "target" system, while in Figure 2 the "OASYS Network Protocol" is the interface to the outside world. Moreover, the "Simulator" in the former figure extends to the target system, while in the latter, it is contained entirely within the HPP Development System. These differences between the figures are meant to underscore the intention that OMAR (1) will be able to perform on the basis of its own, internally devised representations of a target-system or (2) will, after future refinement of the OMAR-OASYS interface, will be able to communicate with the OASYS simulator when more complete target-system representations are desirable for OMAR model development or for real-time OMAR model outputs to an OASYS simulation.

Concept of Operations

One of many possible scenarios describing the interactions of a hypothetical user with the OMAR system is presented below. The scenario begins with a search of the OMAR data base for models appropriate to the user's current needs. It then proceeds through the revision of an existing model, scenario and (simulated) operator task structure, and the specification of run-time displays and performance measures. Finally, the scenario ends with a run, analysis, and second revision of the model. The activities undertaken by the user across this set of tasks are summarized at the end of the scenario. The requirements and specifications discussed later in the document are addressed to this set of tasks.

Although the concept does not include a review of the steps that might be taken to set up and monitor exchanges between OMAR and OASYS during a joint simulation exercise, a list of activities associated with this OMAR function is included in the summary.

It is important to recognize that, depending upon the precise objectives of a real user, the actual number of "events," their order, and their specific character would be expected to vary from the picture presented here. Indeed, the capabilities of the system envisioned in this document support a wide variety of routes during the formulation, exercise, and test of HPP models.

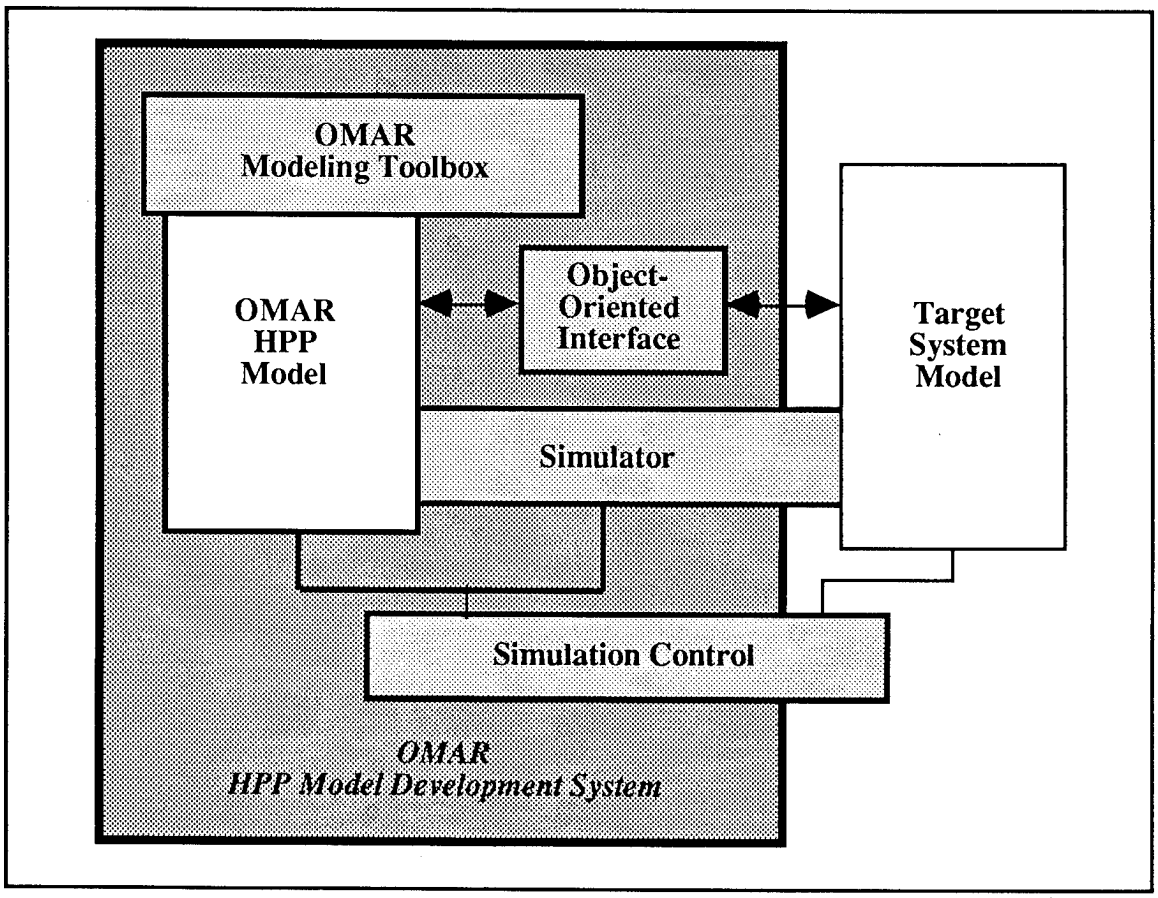


Figure 1
OMAR as a Human Performance Process Model Development Test-Bed

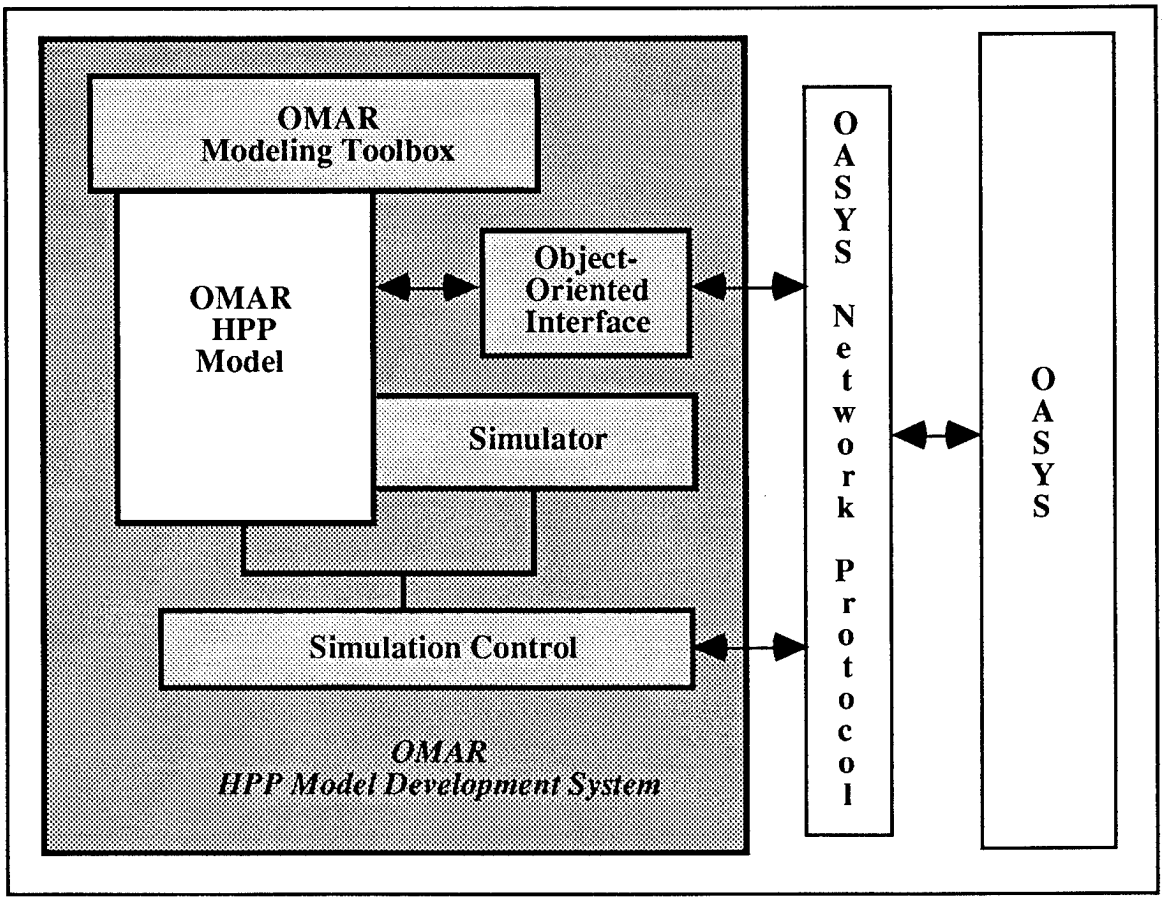


Figure 2

OMAR as a Human Performance Process Model in the Operability Assessment System Environment

User Objectives

The objectives of the hypothetical user include:

- constructing a model that emulates the behavior and performance of an expert air traffic controller engaged in monitoring an airspace, identifying aircraft in potential conflict, and issuing appropriate collision avoidance commands,
- modifying a previously used OMAR research scenario or, if necessary, formulating a new one that can be employed to test the model,
- running the model/scenario, observing performance, and adjusting as necessary, and
- arranging for errors to occur in the (modeled) expert's processing of information in order to identify their effect on task performance.

Task 1: Review Existing Models

The user logs onto the system, checks modeling-related electronic bulletin boards for up-to-date news on new model developments, then begins a review of recent efforts to create a model of the monitoring and decision-making performance of an air traffic controller. The user recalls that there are at least two already-formulated models in the data base and believes that, with suitable modification, one or the other of these may provide a satisfactory basis for a new model. Additionally, others in the user group may have recently contributed models of which this user is unaware. The user resolves to examine the directory of models before determining whether additional relevant models are contained in the larger OMAR data base.

The user begins by viewing the contents of the directory, noting the (expected) existence of the two models directed at controller performance and that of a third which was exercised some time ago in connection with an OASYS simulation of a Weapons Assignment Officer (WAO). Additional entries appear on the list, but notations on the file suggest that the scenario contexts to which they are addressed make them inappropriate for current purposes.

The user decides to make a quick determination of whether the OMAR User Group has made any recent and relevant contributions but is unable to remember precisely how to run an intersecting search for candidates. A review of the contents of the "Help" file yields the answer, and the user, given a set of model-type and scenario descriptors, conducts the intended search. No candidates are found.

Task 2: Examine Selected Model(s) and Reformulate

The user begins a review of the more recent of the two models in the directory. Acquiring a set of summary displays depicting the modeled inputs, processing, and outputs, the user quickly concludes that three aspects of the formulation must be changed if it is to satisfactorily test the current theory and be adequate as a vehicle for simulation. These three aspects are the following.

1. The level at which sensory input to the cognitive portion of the model is specified must be enriched such that elemental attributes of the visual stimulus (e.g., line orientation, texture, and color), rather than intact display icons, constitute the input.

2. New branch values and connections among information-processing nodes in the cognitive portion of the model must be supplied. The user views this aspect of the activity as iterative because, depending upon the performance of the model after these changes are made, new nodes, connections, and weightings may need to be defined.

3. The cognitive module must be configured to produce at least one instance of a specific type of error that an expert controller might make during the performance of skill-based activities. The user resolves to accomplish this step after successfully formulating an error-free model.

Attacking Aspect 2 first, the user must now review the structure of the control task that the model must "perform." The user calls up displays of the task as formally described in the ATC doctrine and as modeled during the earlier effort. The displays present a variety of textual and graphic information. Some present a comprehensive view of the (mission) goals of the task, inputs to and outputs from related tasks, criteria to be met by "successful" performance, and so forth. Others present information relating to the current formulation of the model, including its input, processing, and output parameters, as well as values chosen for initialization during earlier runs. Particularly useful at this level is a group of displays that graphically portrays the connections and weightings among process nodes in both sensor/effector and cognitive portions of the model and, in addition, depict the associations among these elements, the structure of the task, and the (modeled) goal hierarchy of the (human) operator. The views of these items can be zoomed in and out to reveal more or less detail, and if desired, the displays can be used as points of entry to underlying program code by pointing and clicking on specific areas of interest.

Instead of pursuing the point and click option, the user, who is familiar with the organization of the program and the programming language used, decides to use a shortcut and types a command that immediately results in a display of the desired code. Next, the user creates two new nodes and defines their "success" conditions. The connections of these nodes to the existing network of nodes are then created and transition weightings to be associated with branches from the nodes are assigned. Returning to the graphic display of connections, the user verifies that the nodes, connections, and weightings have been correctly specified and that the formulation of the "cognitive" portion of the model now satisfies requirements.

One major task remains to be accomplished before a test run of the model can be made: the sensory (visual) module that provides input to the model must be reformulated. Since no module of the type desired is contained in the OMAR data base, one must be created.

The user moves immediately to the code level and begins to develop the model. The attributes associated with visual stimuli of interest and objects defined by collections of these attributes are first specified. A set of procedures is then written that specifies the criteria by which attributes will be evaluated by the "eye" and transmitted to the cognitive module. Finally, a set of procedures that will cause the eye to exhibit dynamics associated with limited sensitivity and scanning behavior is coded.

The user then unlinks the procedures implementing the earlier version of the sensory module from the cognitive module, saves them in a file for later use, and links the new code. As a final check on the integrity of the link, the user returns to graphic displays of the completed model and zooms in on the region(s) of the formulation where visual inputs are accepted into the cognitive module, and processing begins.

Task 3: Review and Revise Scenario

The user now moves to selection and definition of a scenario to test the performance of the reformulated human performance model, beginning with a review of the scenarios used in earlier validation studies. The user examines the contents of the scenario file and selects a summary display of the ATC scenario most recently used in connection with the model.

The summary presents a set of ATC events; the initial locations, characteristics, and behaviors of five aircraft; and the "world" in which the events take place, including location, geometry, and rules governing the flow of traffic entering, leaving, and already within controlled

sectors. Graphic displays included in the scenario summary present plan and profile views of the airspace and initial positions of aircraft associated with the last run of the HPP model. If desired, the user can point and click on features of these displays to gain access to the underlying program code and/or parameter values associated with agents in the simulation.

After a brief review, the user decides that the earlier scenario is satisfactory as a test vehicle if the number of aircraft in Sector [S] is increased by one and an airport is added in the northeast corner. To make these changes, the user selects the item "aircraft" from a pull-down menu, positions the display cursor at the desired location on the plan view of the display, then clicks the mouse button to introduce an aircraft icon at that position. The user completes the specification by choosing the default settings on a pop-up panel that requests the desired heading, altitude, speed, and maneuvers of the aircraft just created. The need for an airport is met in a similar way, except that parameters to be specified relate to number, length, and orientation of runway(s). Since these parameters play no role in this test of the model, the user chooses the default values offered on the pop-up panel.

Task 4: Define Performance Measures/Output Displays

The goals of the user up to this point have been to enhance the model by developing a more complete specification of the operation of the visual receptor and of its input to the cognitive module; to create additional detail among simulated procedures, plans, and goal nodes; and finally, to specify a simulated task within which the success of these efforts can be gauged. It is now appropriate to identify aids to the visualization of model operation during a simulation run and to analyze performance after the run has been completed.

The user begins by selecting a set of graphic displays that will be useful for run-time monitoring from a menu. The display of ATC operator goals, plans, and procedures and the scenario of Task 3 are the first chosen because they provide the most highly aggregated summary of underlying connections within and among model and task elements. In its "run-time" mode, these displays will present the nodes and connections associated with the structure of the model, scenario, and task elements in various colors and highlights, indicating their active/inactive/pending status as the simulation proceeds.

An important aspect of the performance of the HPP model is the workload it "experiences" as it accomplishes various ATC tasks. The user is particularly interested in those situations where the simulated operator, by virtue of the complexity and immediacy of potential conflicts, may be compelled to accomplish a variety of monitoring, planning, and communication tasks at the same time; that is, to "multitask." Accordingly, the user selects and supplies parameter values for three distinctly different types of displays. One window on this display depicts the instantaneous and time-averaged workloads associated with receptor, cognitive, and effector modules. A companion window presents the relative priorities of active goals from the simulated operator's point of view.

The second display presents iconic representations of various agents in the simulation (e.g., the ATC operator, the aircraft flight crew members) connected by lines that change color as the agents interact with each other, showing the direction of the flow of communications. The user can click on any "active" line in this window to view the content of the communication.

The third display presents two different views of the progress of the simulation. On one of these (the "event" line), an entry appears each time a simulation event(s) defined by the user begins or ends. The second is a rendition of the classic time line. Supplemental windows on this display fill when multitasking occurs in the HPP model and when tasks already started are suspended and/or resumed, providing the user with an understanding of the circumstances and content of these events.

Because this will be the first of a set of pilot runs of the revised model, the user decides to specify only a small number of descriptive statistics to be displayed at the end of the run. The user anticipates that additional and perhaps more useful analyses will be required but defers specification until after the performance has been viewed in real time. These can be conducted with respect to audit trail data that will be accumulated during the run.

With the aid of OMAR's on-line "Help" and menu system, the user constructs an analysis "package" providing the following information: (1) maximum, minimum, and average operator workloads for each two-minute interval of the simulation; (2) ratio of procedures begun to procedures completed within each five-minute interval; (3) ratio of procedures suspended to procedures resumed during each five-minute interval; (4) average waiting time to satisfaction of each goal; and (5) frequency with which each goal was addressed during the run. As a final step, the user now saves the entire simulation setup in a file in the directory for later use.

Task 5: Initialize and Run Model

The user is nearly ready to run the model. However, the system first requires information relating to the desired length of the run. (The length of the run can be addressed in terms of elapsed [clock] time, in terms of initiation or completion of specific task or HPP-modeled goals, or in terms of criteria for terminating the run as a result of simulation-induced errors.) Moreover, the user wants to review the initial values of all model, task, and scenario parameters to ensure they meet requirements.

After supplying the required simulation time and error information, the user acquires a display containing windows which present, in tabular form, the sets of values currently assigned to parameters. Pointing and clicking on any of the values results in a graphic display of the region(s) of the model, task, or scenario where the parameter of interest is located.

Working back and forth between the tabular and graphic displays, the user reviews the startup specifications and modifies default values and some earlier nondefault choices as desired. OMAR actively assists in this process in two ways.

- For parameters that have been revalued, OMAR presents both prior and current values assigned, supporting some "dither" in the user's decision process as the review is conducted and providing a means for focusing efforts during later reviews aimed at sensitivity testing.
- During review of HPP model parameters, OMAR highlights, on both tabular and graphic displays, those process, plan, and goal nodes associated directly with revisions made during this modeling session.

The review is completed quickly, and the user initiates the simulation. The run-time displays selected earlier and now animated are monitored closely. Graphic presentations are occasionally zoomed in and out as the user's needs to obtain both microscopic and macroscopic views of modeled processes vary.

Frequently, the simulation is thrown into the "single-step" mode and combined with zooming to allow close examination of unfolding events in the procedure layer of the model and their effect on task accomplishment. Occasionally, the user identifies an event or sequence of events that might be of interest during postanalysis. By pressing a key, the user marks the simulation audit trail, thus enabling a quick return to this region of the run and to this configuration of displayed information at the end of the exercise. The model run continues until the criterion specified for termination (in this case, a period of time) has been satisfied.

Task 6: Induce Model-Based Errors and Rerun Simulation

The model now runs successfully, providing a good approximation of the error-free performance of a skilled ATC operator. To *satisfy* certain theoretical interests and to make the model more generally useful to the system-design community, the user now wishes to induce in the simulation errors of the sort that skilled operators occasionally make.

Two specific error types are of interest to the modeler. One is a simple blunder: unintended triggering of an effector. The second is more complex: application of an incorrect decision rule in a situation that, except for a subtle but critical difference, is similar to one in which the rule could be correctly applied.

The user reacquires the set of displays employed during the earlier revision of the model. Since the user holds no theoretical position regarding the genesis of blunders and current interest centers only on forcing them to occur now and then, attention is focused on the modeled boundary between the cognitive and effector modules. Here, and in a manner similar to that used in network models of an earlier, pre-OMAR generation, the user simply modifies the weights on certain efferent branches in accordance with the desired frequency of errorful behavior.

The rule-based error is far more interesting to the user and presents a much richer set of opportunities for incorporation into the cognitive layer of the model. Display windows associated with marked regions of the audit trail compiled during the earlier run are carefully scanned for regions where redefinitions of nodal "success" criteria could plausibly be introduced. Three candidates are quickly found.

With the aid of OMAR search routines, the user examines the entire audit trail to determine the frequency with which each candidate situation occurred. Only one is found to have occurred with a frequency sufficiently low to qualify as the desired "unlikely event." Using the earlier combination of graphic and tabular displays, the user modifies the appropriate nodes, connections, and weightings such that misapplication of the rule will occur in future runs when this unique intersection of conditions is encountered.

The user now re-initializes the simulation and initiates a run, monitoring the displays and performance measures as before. The rule violation does not occur in this instance, but, as expected, the blunder does. A quick review of the event-line display indicates that the conditions under which the violation might have been observed did not occur during the run. The user resolves not to run the simulation again at this time and, instead, saves the simulation setup for later use.

Summary of OMAR User Activities

OMAR Stand-Alone Mode

OMAR has two modes: Stand-Alone and Communication. Within each mode are several areas under which the user may perform numerous tasks. The following list summarizes user activities within these various areas/modes.

Manage/Housekeep

- review OMAR system conventions, macros, speedkeys, Help files, and so forth
- specify indexing features, model descriptors, or keywords to aid in reacquiring model
- save a new version of an existing model and/or save a new model in directory <username>

- enable/disable access by other users to directory <username>
- delete a model from directory <username>
- send/receive electronic mail

Review Model System Definitions

- browse the contents of own directory of prior models
- conduct search of system definitions for relevant models

Generate a Model

- build a new HPP
- edit an existing HPP
- assemble a (new) HPP by combining elements of existing HPPs
- initialize model parameters

Generate a Scenario/Task/Task Structure

- build a new scenario
- edit an existing scenario
- define new operator tasks and/or build a new task structure
- edit existing operator tasks and/or edit an existing task structure
- define/select errors/error modes to be exhibited by model

Prepare a Run/Post-Run Analysis

- specify the set(s) of data to be collected by the system
- define/select performance measures for run-time and/or later-time analyses (i.e., specify how data to be collected is to be mapped into desired measures of performance)
- define/select a set of displays for model performance monitoring
- define/select type(s) of post-run data analysis to be performed
- define/select data to be included in analysis

Run/Test a Model

- initiate a run
- select, zoom in/out displays of model performance
- pause/single-step/resume a run

Analyze Outcome of Run

- initiate data analysis
- compare results of the current run with those of an (earlier) run that used the same HPP model
- compare results of current run with those of an (earlier) run that used a different HPP model

Additional Activities in OMAR-OASYS Communication Mode

Although activities undertaken by OMAR to set up and monitor communication between OMAR and OASYS are not described in the preceding section, the following are likely candidates:

- set up interface for exchange of messages/data,
- initiate message/data exchange (OMAR-oasis handshake),
- monitor integrity of communications links, detect/rectify errors,
- import oasis target system description,
- accept requests from OASYS for OMAR model data,
- monitor OMAR model outputs to OASYS,
- initiate requests to OASYS for information, and
- manage OMAR simulation execution in accordance with OASYS protocols.

TOOL-BUILDING REQUIREMENTS FOR HUMAN PERFORMANCE PROCESS ARCHITECTURES

OMAR is a simulation environment in which to develop HPP models. The immediate users of the environment are expected to be computer scientists, but the environment is intended to support the collaborative efforts of psychologists, cognitive scientists, and other specialists in the development of the models. For the present, the environment is purely symbolic, but this does not preclude later integration of nonsymbolic (e.g., subsymbolic connectionist) components. In large part, OMAR is an integrated collection of computational tools concerned with representation. Simulation technology is important as well, with requirements driven particularly by the need to model the human multitask performance.

The OMAR design effort has focused, in part, on the requirement to support the development of two particular HPP models: an implementation of the psychological framework developed in Research, Development, Training, and Evaluation (RDT&E) Delivery Order #1 (Deutsch *et al.*, 1993a) and Michael Young's computational-theory-level holon architecture (Young, 1992; 1993a; 1993b). The intent of this section is to make explicit the links between the modeling requirements of these particular models and the computational tools to be included in the OMAR environment.

Relating the RDT&E Psychological Framework to Tool Requirements

Overview of the Psychological Framework

With deference to both the OASYS and OMAR user communities, the focus in the work reported here was on developing a computational framework that could adequately reflect the central or cognitive components of an HPP model. To the extent possible, the goal was to construct the implementation and associated tools in such a way that they readily support the foreseeable range of extensions, variations, and competing psychological conceptualizations. Nevertheless, the base or default structure of the present implementation is designed on the psychological framework that was presented in Deutsch *et al.* (1993b). Toward this end, the major components of the model include:

- a layer corresponding to the Scanner and Responder Modules, containing input and output ports that correspond to relevant perceptual-motor modalities,

- a layer corresponding to the Memory Module that contains users' direct, experientially based knowledge,
- a layer containing the open goal and plan structures of the Cognitive Module, and
- a layer overseeing the algorithms of the Cognitive Module that are responsible for generally evaluating system status and for arbitrating contention across goal and plan structures.

Given the framework developed in Deutsch *et al.* (1993b), the knowledge and processes of each layer are represented in cognitively coherent packages. Across layers, however, the knowledge and processes and their interactions differ in ways that are logically important to the model. Therefore, in describing the model, the terms "goals," "plans," and "procedures" are used noninterchangeably in this document to clarify distinctions. "Goals" is reserved for the knowledge structures that comprose the Cognitive Module. As in the larger literature, goals are hierarchically defined such that, from the bottom up, each goal structure specifies, at increasing levels of priority and abstractness, what the operator wishes to achieve, and the conditions that enable and criteria that define its adequate achievement.

"Plans" refer to the bottom-most level of a goal hierarchy. Formally, plans are like other subgoals in a hierarchy. However, they differ informationally from all superordinate goals in their hierarchy in that their triggering or satisfaction is necessarily defined by information or events that come from outside the hierarchy. Specifically, the triggering and fulfillment of plans are mediated by the Memory Module.

Unless the data required by a plan already reside in the operator's Memory Module, they must be acquired from the real world. The data acquisition, in turn, depends on the learned procedures of the Memory Module. The procedural knowledge in the Memory Module consists of experientially based action or event schema. In theory, the nodes of a procedural schema correspond to the declaratives (contentives or ostensibles) of the event sequence, while the relations that hold it together capture the spatial, temporal, and conceptual relations among its nodes.

The complete specification of a plan consists of designating the procedures for taking it from initiation to completion, along with the acceptable values that each of those procedures must return. Because the scope, connectivity, and contentive flexibility of such procedural knowledge are determined by personal experience, an action that is specified in terms of a single, extended, and well-specified procedural schema for an expert may instead be represented by a number of more limited procedural schemata for the novice. Within the present implementation, the minimum number of procedures for specifying a plan is two: one for executing the planned action and one for evaluating its results.

Scanner and Perceptual Modules: Perceptual-Motor Ports. To be capable of sensing and responding to its simulated environment, the model must possess virtual eyes, ears, hands, and a mouth. Significantly, whether from external trigger to recognition or from internal trigger to response, thoroughly modeling just one of these modalities would constitute a major effort, involving a broad and complex range of theoretical alternatives and implementation decisions. So difficult, in fact, are the challenges in modeling the knowledge and processes underlying any of these modalities that successfully doing so does not seem a feasible near-term prospect, except where the physical input or output possibilities of the modality have been pruned to a subset of interest. Yet, given the range of research and evaluation goals for this model, these subsets cannot be anticipated; they necessarily depend on questions about specific and to-be-determined person-machine systems or automation options.

Recognizing this, and with the goal of building a system that cognitively complements any such effort, the representation of each perceptual-motor modality is limited to "ports." The goal

is to design these ports so that more specific and extended models of peripheral processing can be hooked onto the system. Meanwhile, within the present implementation, subsymbolic processing is finessed to stub sensory input and motor output at cognitively indivisible levels of pattern recognition and response. In other words, the program will not be designed to see, hear, or feel but only to "know" the what, when, and where of its seeing, hearing, and feeling; similarly, the program will not simulate responses beyond stipulating their when, where, how, and what for.

The focus of the present implementation is on orchestrating the collective dynamic of the perceptual-motor modalities or, more specifically, on developing algorithms that distribute the simulated operator's attention and effort across input and output pressures and opportunities in cognitively plausible ways. In this context, it is significant that each perceptual-motor modality must be seen as a resource that is contended for at several levels of analysis. For example, a hand can be in only one place and position at any one time; furthermore, although most people have two hands, they generally cannot direct their two hands to perform two independent tasks of any complexity. Similarly, one's visual intake is limited at any given moment by where one's eyes are pointed and, despite the abundance of visual information in the field of view, one can focus physically and mentally on only a fraction of it. Although the ears differ superficially from the hands and eyes in the physical range of their sensitivity, what is "heard" nonetheless corresponds to a select and exclusive sample of the auditory stimulation in the environment.

As reflected by these examples, the nature of within-modality contention is not merely physical but cognitive as well. Moreover, such cognitive contention operates across as well as within perceptual-motor modalities. Everyday examples of these phenomena abound: one can talk and drive at the same time, but not if the traffic gets tricky; one can sing along to music or even harmonize, but it is very difficult to sing one song while hearing a totally different one unless the first is extremely familiar; one can enjoy a movie while brushing one's hair or folding the laundry, but not while solving probability problems; one can watch a movie while taking in its words and music, but one cannot absorb a movie or a musical piece while processing an unrelated stream of speech; one can type from copy, but one cannot type one message while reading another; one can read aloud, but one cannot read while talking about something else.

There may well be sources of intramodality contention that are not physical; the goal is to design the implementation such that it permits assessment of such possibilities. In its default structure, however, the implementation will support two types of contention for the perceptual-motor modalities: physical and cognitive. Physical contention is a within-modality phenomenon and occurs whenever two or more mutually incompatible events vie for the service of the modality. Cognitive contention, in contrast, may occur both within and between modalities.

In essence, cognitive contention is governed by the principles that psychological activities, including perceptual-motor activities, tend to be mutually inhibitive except (1) where both (all) are governed by a single plan or (2) where execution of the plans involved do not require evaluative activities for their realization (see Deutsch *et al.*, 1993b; Adams et al., 1991).

Importantly, sound execution of perceptual-motor tasks always requires evaluation of the information returned through the procedures to the plans (i.e., are the returned data acceptably similar to the expected data?). The time and effort invested in this evaluation will depend on both the nature of the data (e.g., whether the operator must notice how it evolves over time) and how the sought data is specified by the plan. In particular, it will increase to the extent that the sought data is either poorly specified (as in conditions of little familiarity or advance view) or very stringently specified (in terms of precision or number of relevant dimensions). (Alternatively, of course, for reasons external to that particular plan, the operator may shed or make short shrift of the evaluation process.) Verification and assessment of response effect are treated like any other perceptual act.

Finally, even though the model's perceptual information and motor programs are stubbed at the cognitive level, its performance cannot be evaluated unless it does something observable. The model's perceptual activities and responses will therefore be ghosted on the display monitor and recorded for analysis. For example, the simulated visual behaviors of the model are displayed by highlighting the scanned and fixated information on the facsimile of the operator's environment. In the same spirit, the display is designed to provide real-time traces and images of what the operator hears, feels, does, and says.

The Memory Module: Direct, Experientially Based Knowledge. The operator's experientially accrued knowledge of concepts, procedures, and events is represented within the Memory Module. As reviewed in Deutsch *et al.* (1993b), it is assumed that both recognition of impinging information or events as well as direct or indirect recall of old information depend on the activation of corresponding knowledge complexes within the Memory Module. Moreover, it is assumed that plans are translated to responses only by triggering some amenable procedure or set of procedures from this knowledge base.

In theory, the knowledge in this layer is the product of associative synthesis. That is, knowledge of the simplest concepts, events, and actions consists of sets of primitives, interconnected in ways that capture the relations between them. Knowledge of more complex concepts, events, and actions consists of interconnected sets of those sets, and so on, where the strength and nature of the connections between node (whether clusters or primitives) reconstruct the experienced relations between whatever those nodes represent.

With the recent interest in connectionism and neural nets, the art of generating such knowledge structures from primitives has become an active experimental science. Therefore, an attempt has been made to design the HPP framework so that such experimentation can be admitted within it. For the present implementation, however, the most primitive nodes in this Memory Module will correspond to situationally basic perceptual-motor engrams, such as the sound of a telephone bell or a warning alarm and the actions of pushing a button or speaking a message.

The rationale and justification for this level of analysis are as follow. First, research and theory indicate that sensed events are automatically propagated "upward" from sensory elements to their highest familiar levels of coherence. Thus, events at the level of telephone bells and button-pushes should be wholly familiar even to the most naive operator. Second, research indicates that people generally have reflective access only to terminal or, equally, highest levels of representation within the Memory Module. Since the present effort is focused on the cognitive dynamics of the situation, there is little incentive to decompose these familiar nodes. Third, these events are not coincidentally at the same level of abstraction at which the perceptual-motor ports are to be stubbed and their activities are to be ghosted.

Finally, by interconnecting these basic nodes into more- and less-complex knowledge representations, the descriptive adequacy of the knowledge-based theories about differences between expert and novice performance can be assessed. In a given situation, for example, the novice ATC might hear phone #4 ring, look to verify its location, pick up the receiver, receive a hand-off request from a neighboring ATC, think about the traffic display to assess the request, and so on. The expert, in contrast, may know with the ring of the phone not only where to reach but whom and what request to expect on the other end of the line and precisely what conditions to seek on the display in order to grant it. In terms of the Memory Module, the difference is that while the novice is working with a series of mnemonically disjoint sets of perceptual and procedural knowledge, the expert is working with only a couple more complex and extensive ones. In theory, this difference in the connectedness of the supporting knowledge should enable the expert both to receive and to respond to the information more quickly and comprehensively. The present implementation is designed in anticipation of researchers' interests in expanding,

contracting, and restructuring the procedural schemata so as to explore the ways in which their structure might influence performance.

The Cognitive Module: Goal and Plan Structures. The job of the Cognitive Module is to support the system's ability to interpret and act upon its environment in a way that is consistent with its collective needs and interests. Toward this end, it was proposed earlier that the Cognitive Module should include both the goal structures representing the system's needs and interests, and the algorithms governing the individual and collective activation of those goals. To manage the structure and dynamics of the Cognitive Module more cleanly and manipulably, the goal structures and their control algorithms should be separated into two distinct layers in the implementation. This first layer is to be composed of the goal structures.

Just as any person entertains a number of needs and interests at any one time, the Cognitive Module is designed to maintain a number of open or pending goal structures at once. In the ATC environment, for example, goals that might be simultaneously open include accepting hand-off of aircraft, maintaining separation for aircraft in a sector, inputting decisions into the computer, reorganizing the board to reflect current work space, receiving communications from pilots, assisting an apprentice, completing the current shift, general monitoring of the environment and maintenance of personal comfort, and any variety of goals unrelated to the immediate context. An open goal is closed when it no longer begs completion; either it is no longer of interest or it has attained closure through fulfillment.

In terms of the dynamics of the system, an important characteristic of open goals is that they influence the responsiveness of the system to activity in the Memory Module that is relevant to their triggering, pursuit, or satisfaction. In the present implementation, this influence is mediated by the interarticulation of the plans, or bottom-level goals, with relevant schemata in the Memory Module. The more precisely a goal's sought information is defined, the more focused (less diffuse) is the interarticulation between the Cognitive Module and the Memory Module.

This interface between the goal structures and the Memory Module inherently supports several key psychological phenomena. First, it provides a mechanism by which the operator can attend to events that are relevant to any open but deferred goal structure. Second, research indicates that although sensed events automatically gain representation in the Memory Module, only that small fraction of events that are "attended," however briefly, leave any enduring trace of their occurrence in memory. Through the implementation, it is assumed that "attended" events are those whose relevance is evaluated (even if evaluated and rejected) with respect to some open-goal structure. Third, this design builds a face-valid type of interpretive bias into the system; if the nature and implications of impinging events are examined at all, they are examined with respect to the system's reigning needs and interests.

The Cognitive Module: Evaluation and Contention Arbitration. As described in Deutsch *et al.* (1993b), it is assumed that the Cognitive Module can be directly engaged in the instantiation or pursuit of only one open-goal structure at a time. It is also assumed that each goal structure is defined (and therefore concerned) exactly and only with the satisfaction of its own goals. It follows that there must be some means of transcending the pressures of the separate goals in a way that promotes the performance and integrity of the system as a whole. The purpose of this layer is to allow representation and exploration of the sorts of control algorithms that may underlie this capability.

Toward this end, the present implementation incorporates the assumptions that open goals differ from one another in their level of activation, that the system can only support some maximum total activation across goals, and that the activation level of a given goal directly influences the strength with which it can prime associated structures in the Memory Module and (consequently?) the likelihood that it will be serviced. A corollary of these assumptions is that

the presence of one or more highly activated goals in the space will result in concomitant, if temporary, diminution of a person's sensitivity to the needs of other open but pending goals.

More specifically, the activation weighting for each goal will be a joint function of several factors, including its temporal urgency, the recency with which it has been attended, the total amount of effort already invested in its attainment, the extent to which the conditions on its attainment are already satisfied, the benefits of its successful attainment, and the costs of being mishandled, compromised, or forfeited.

Some, but not all, of these weightings can be realized within the separate goal structures themselves. For example, a reasonable way to manage the effort invested and its recency is to record each instance of the former and weight it with the latter. Although this algorithm is mindless of the relative importance of the goals to the operator's overall well-being, it does capture the strong characteristics of the Zeigarnik effect. In addition, this algorithm generally ensures that if an operator is intensely preoccupied with some goal, it is unlikely that goal will be displaced for very long if at all. After all, the recency and intensity of the operator's efforts will only augment the high level of activation that induced work on that goal in the first place.

Pressures related to the amount still done or not done toward the attainment of a goal can also be captured within the individual goal structures. First, the greater the progress toward goal attainment, the better specified the remainder of its plan is likely to be. In this way, partially attained goals will more strongly prime the event and action schema relevant to their completion. Second, the difference between an open goal and a dormant goal is essentially whether its bottom-most level is a plan that is defined entirely in the abstract or partially instantiated and concretized through articulation with the memory module. Yet, all a goal's planned actions or events may be seen as triggering conditions relative to its consummate or end condition. Thus, the greater the extent to which a plan has been completed, the higher its activation relative to the maximum afforded by the constellation of other factors.

Moreover, if the activation of a goal increases with the activation or tightening of the constraints that define its effective execution, increases in priority with the urgency or unforgivingness of its temporal constraints on attainment could equally well be mediated through properties of the individual goals. However, this creates an incomplete picture of the ways humans deal with temporal constraints. That is, in scheduling the pursuit of any goal, it is not merely the temporal deadlines of that goal that are considered but also the total time required to *satisfy* it, and the deadlines and total time required to *satisfy* other pending goals. Where not all of a set of such goals can be attained in the time available, temporal factors are reconsidered against the relative priorities and internal structures of the goals. Maybe one or more is abandoned, maybe they are reordered in a way that is discrepant with their relative priorities but that maximizes the extent or likelihood of their collective attainment, or maybe the constraints or criteria for the attainment of one or more goals in the set are relaxed or restructured so as to expedite their execution or reduce the conflict.

The point is that all these alternatives for adaptively reconciling temporal conflicts depend on the ability to consider the internal specifications of more than one goal structure at once. Also, because attention to one goal necessarily involves the deferral of others, adaptive evaluation and management of the costs of mishandling, compromising, or forfeiting goal opportunities must also involve the ability to consider the attributes and constraints of more than one goal at a time. Moreover, neither the benefits of succeeding nor the costs of failing on any pending goal can be properly evaluated and traded off, except by considering the means and ends of the contending set against competing needs and interests. The resolution of such conflicts will sometimes involve melding or restructuring the pertinent goals.

In view of this, the distribution of attention and effort will be mediated across open goal structures by means of a self-management process that is a component of each goal. This self-management process considers neighboring open-goal structures in its evaluation in modulating the distribution of activation across open goals. The process is capable of informally assessing the appropriateness or completeness of data to the plans by which they are primed or to which they are returned. By the same token, the self-management process is capable of the evaluative perspective required for searching through the Memory Module for plan-relevant prior knowledge or for restructuring or pruning open-goal structures so as to influence contention in ways that are adaptive. This job depends critically on the capacity to evaluate the constraints and instantiations of the open goals with respect to the collective needs of neighboring interests, an essential aspect of the self in its endeavors.

The Psychological Framework and OMAR Tool Requirements

The psychological framework developed in RDT&E Delivery Order #1 (Deutsch *et al.*, 1993a) established the theoretical groundwork for the implementation of an HPP model. This framework is mapped into a computational form that is implemented in OMAR. In large part, it is the design of the computational implementation that dictates the form the tools should take.

This section focuses on the aspects of the computational implementation of the framework that influence tool selection. Modeling human performance is an ambitious undertaking, requiring an integrated set of representation languages. A frame language, a procedure language, and a rule language form the basic building blocks for the modeling efforts to be undertaken using OMAR. The frame language provides a few representational extensions beyond the Common LISP Object System (CLOS) (Steele, 1990), the object-oriented substrate on which all system components are built. The procedure language implements procedures as frame language objects, and the rule language operates over frame language objects and procedures as objects. Extensions to the procedure language provide a basis for modeling proactive human behaviors and the multitasking capabilities of human operators.

Attributes of the Human Performance Process Model and OMAR Tools

Particular aspects of human performance identified in the theoretical framework need to be represented in the HPP models developed using OMAR. Those that particularly impact the selection of tools to be included in the OMAR tool suite are discussed here.

Proactive Behaviors

The psychological framework requires a view of goals as the basis for proactive human behaviors. A goal (Figure 3) expresses the conditions necessary for its achievement and includes a goal-tree with subgoals directed toward the achievement of the goal. The leaves of the goal-tree are plans, distinguished by their invocation of and response to one or more procedures-agent actions. In the current implementation of Sproket semantics, a goal is defined solely by the conditions necessary for its achievement. It is realized by a plan made up of subgoals and actions directed towards its achievement conditions. A goal expresses what is to be accomplished, a plan outlines the steps to achieve the goal, and the actions, or procedures, direct the behaviors of the agent.

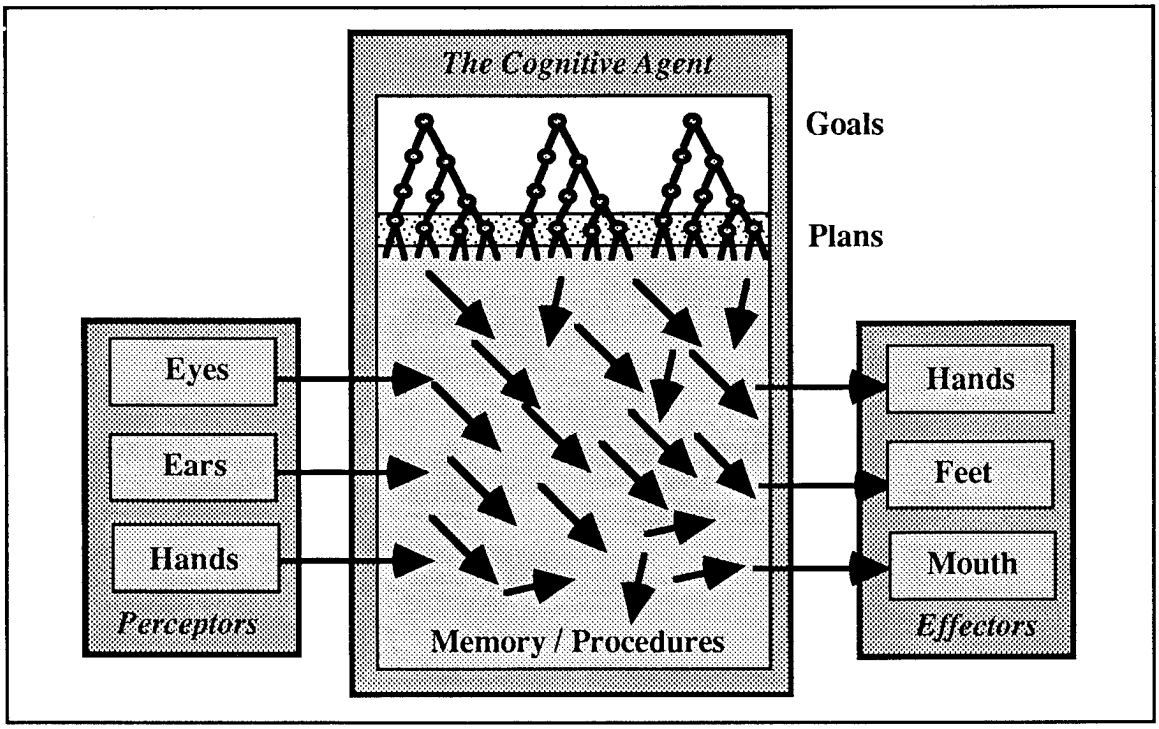


Figure 3
Psychological Framework

The procedural language, Simulation Core (SCORE) (Deutsch & Palmucci, 1992), provides a representation of actions as defined here. It has been enhanced to support one form of the semantics of goals and plans from an earlier Sproket (Abrett *et al.*, 1990; Deutsch & Palmucci, 1992) simulation language. The Sproket representation of goal-directed behavior, as outlined above, is close to that required by the psychological framework, and few changes were required to *satisfy* OMAR requirements. The distinctions among goals, plans, and procedures are now more clearly defined than in Sproket, and the instantiation of a plan more closely represents a psychological view of a task.

Multitasking and Automaticity

The modeling of multiple-task behaviors required a formalism to represent the interactions between tasks contending to execute. The psychological framework suggested basing this competition on the accrual of activation credits vis-`a-vis other tasks contending to execute. In a resource-based model, contention exists among tasks competing for a portion of the resource. Similarly, in a model explicitly separating the conscious and nonconscious portions of a task, there will be competition for attention among the conscious portions of contending tasks. That is, as tasks are developed, they naturally form contention classes — classes within which the competition to execute will need to be modeled.

The unique feature identified here is the classification of procedures in relation to the way they compete with other procedures to execute. Procedures may compete with other procedures within their class or from different classes. The Simple Frame Language (SFL) (Abrett *et al.*, 1990) used to define the basic objects of the OMAR environment is employed as the basis for classifying SCORE procedures. SCORE procedures are SFL objects. Given the classification of goals, it has been necessary to provide a basis for developing the protocols for implementing the explicit contention among the goals and procedures within or between classes. Generic functions are available for managing the decision to suspend or resume a goal or procedure and for the

actual suspension and resumption of individual goals and procedures. Furthermore, provisions were required to enable a suspended task to dictate its own behavior during the period for which it was suspended. As the activation levels of tasks are adjusted over time, these primitive functions provide the basis for a task asserting its right to execute vis-`a-vis other tasks in its contention class or between it and other task classes with which it contends. A loss in activation level for an active task opens a window of opportunity for competing tasks, while a gain in activation level by a nonexecuting task enhances its opportunity to execute. The means to establish restart points for resuming a task are also provided. Particular forms of contention-to-execute are developed and represented in appropriate procedure objects as needed to support the implementation of the psychological framework.

Task Contention and Emergent Behaviors

Tasks impacting the activation levels of other tasks are another dimension in which tasks will contend. This contention can take the form of inhibition as well as activation. That is, a running task may have either a positive or negative impact on other running tasks and tasks contending to run. The generic functions identified above and the ability to classify procedures are the structures needed to build the particular task contention protocols required for the implementation of the psychological framework.

As an example, eyes may be viewed as a resource that tasks contend for: either a task can direct the scan of the eyes or it cannot — there is no middle ground. When a task accrues activation credits in sufficient quantity to direct the eyes to a particular screen icon or to scan a panel, it will pour out inhibitors, suppressing the previously eye-using and currently competing eye-needing tasks. The accrual of activation credits for a task may be driven by events in the environment (e.g., a bright flashing signal) or by a cognitive decision that promotes the activation of a particular eye-needing task. In either case, the newly active eye-using task will inhibit other eye-needing tasks. Agent behaviors emerge from the dynamics of the activation levels among competing tasks.

Reentrant Map Semantics

The psychological framework is realized, in part, by a network of procedures. Emerging agent behaviors are the result of data-driven events and goal-directed actions. Task activation, with inputs arriving from multiple sources, have suggested a data flow architecture. This approach finds neuroscience support in Gerald Edelman's (1987) development of the Theory of Neuronal Group Selection (TNGS). In the BBN/OMAR model, procedures correspond roughly to reentrant map elements that are driven by multiple inputs and generate outputs for multiple consumers. The data flow in the network imposes sequentiality along the paths of its flow, while also enabling parallel execution among other nodes.

In the data-flow architecture, all arguments need not come from a single source; that is, one procedure may provide some of the arguments to a subsequent procedure, but additional arguments from another procedure may be required before the subsequent data-flow node may execute. Breaking with a traditional data-flow architecture means that not all the arguments need necessarily be present for a procedure to execute. Some data flow ideas will be used in the development, but the psychological framework does not enforce a pure data flow architecture.

A typical SCORE procedure is much like a LISP function in that it may return values to its calling procedure, but this event simply does not happen in data-flow networks. Instead, a procedure in the data-flow network is likely to use the SCORE *loop-forever* macro, sending out its results as events or further subroutine calls when it nears the completion of an iteration and returning to the top of its *loop-forever* to await its next activation. A prototype of the data-flow management of the delivery and arrival of procedure arguments using basic SCORE functionality and data-flow-like macros has been implemented to code these SCORE extensions more directly.

As this and other issues were addressed, the implementation evolves into reentrant-map semantics that support the implementation of the psychological framework.

Parallel Execution

The parallel execution of procedures can be handled nicely, both at the language level and in the simulator. At the simulator level, parallel execution of procedures is one of the two primary requirements driving simulator design on a serial machine; the parallel execution of tasks must be accomplished through simulation. (The other requirement is the need to simulate actions that take place over an extended period of time, referred to as actions-with-duration.)

At the language level, parallel execution may be expressed at two levels of abstraction. At the SCORE level, the primitives *race* and *join* manage the execution of parallel sub-procedures. Parallel procedures operating under *race* all complete when the first one completes, allowing execution of subsequent procedures. For *join*, each subprocedure must complete before subsequent procedures execute. The semantics developed for goals include similar capabilities for managing their serial and parallel execution as outlined in the psychological framework. In the and/or tree of subgoals for a goal, parallel execution is implicit at each level. Temporal precedence is controlled explicitly by dictating sequential subgoal execution where desired.

Rule-Based Behaviors

While it is likely that expert knowledge formulated as rules will not lead to expert performance (Holyoak, 1991), there are performance levels at which human behaviors are rule-based (Rasmussen, 1983; Dreyfus & Dreyfus, 1986). To model these levels of performance, a rule-based system is required as part of the OMAR tool set. Since objects in the OMAR environment are based on SFL concept definitions, the rule language is able to reference SFL-defined objects. Furthermore, since the reasoning that is to be modeled will frequently involve decisions on actions to be taken, the rule system is similarly required to reference SCORE-based goal, plan, and procedure definitions.

Skill Levels in Human Performance

Human skill level has a significant impact on measured performance. HPP models developed in the OMAR environment must be capable of exhibiting performance associated with selected skill levels. Across a range of skill levels, there will be qualitative as well as quantitative changes in the actions that an agent will take to complete a particular task. In modeling these actions as procedures, the quantitative changes will reflect the efficiency with which the tasks are carried out. To meet this requirement, the procedure language provides for the use of a function to compute the execution time for procedures. Improved skill levels are also reflected in improved multitasking performance — adjustments to accommodate these have been developed as part of the conflict-resolution strategies that implement multitask performance.

The transition from thoughtful deliberate action to automatic behavior is one of the more notable qualitative changes in performance with improved skill levels. The performance of a given task at these very different levels of performance will be represented by distinct sets of procedures, frequently with very different structures. In Delivery Order #1 of RDT&E (Deutsch *et al.*, 1993a), a prototype for matching agent task selection to agent skill level was prototyped. This prototype has been employed in the current project used as the basis for extensions to the required procedural language.

The Holon Architecture and OMAR Tool Requirements

Researchers have brought, and can be expected to continue to bring, a wide variety of modeling methodologies to bear on the problem of developing HPP models. One OMAR goal is to establish the capability to accommodate a reasonable segment of these approaches. As a first step toward accommodating additional modeling approaches and methodologies, the OMAR modeling environment is intended to support holon concepts as outlined by Young (1992; 1993a; 1993b). Young proposes the use of holon concepts as a basis for a computational-theory-level architecture for HPP model research. Holons (Koestler, 1976) and the hierarchies they inhabit are expected to be reasonably supported by the generic tools that are to be developed for OMAR; however, some special considerations are necessary. This section provides a brief overview of OMAR support for a holon-based HPP architecture and notes some of the special considerations that have been addressed.

Holons as Agents

In the OMAR environment, agents are just objects that are specializations of a basic agent object provided by the SCORE simulator. Two immediate approaches to mapping human agents onto SCORE agents are available within OMAR. In the first approach, a human agent is modeled as a single SCORE agent; in the second, the human agent can be modeled as a composite of SCORE agents. The autonomous nature of holons suggests the latter approach with each holon modeled as a SCORE agent — a mapping that SCORE easily accommodates. A holon, as a SCORE agent, can be assigned multiple goals and procedures to express its self-assertive tendencies (Koestler, 1976; Young, 1992). As outlined in Young (1992), a holon can consist of a hierarchy of holons. Here, the holon might be a SCORE agent with a slot pointing to the hierarchy composing its substructure of additional holons as agents.

The alternate approach is also quite viable: a single SCORE agent would be used to model the human operator and the organization of procedures assigned to the operator would be mapped onto the holon structure. This approach is a little less straightforward and may not have quite the flexibility of the former approach, but it does suggest the range of options open to the developers of the holon architecture using the OMAR environment. The discussion that follows will explore only the mapping of individual holons onto SCORE agents.

Hierarchies of Holons

Holons form a hierarchy (Young, 1992). In defining a holon as an agent in OMAR, parent and child slots can be included to express a holon's place in the hierarchy. The particular structure of the hierarchy is completely in the hands of the developers of the holon architecture. Since agent definitions are based on SFL concepts, clusters of holons may have hierarchies specialized for their particular organizational requirements.

Message Passing as the Communications Protocol for Holons

Most considerations regarding holons that have been noted so far can be addressed in a variety of ways using the generic capabilities of the OMAR representation languages. The communications protocol for holons can also be addressed with generic capabilities, but a few minor extensions will be considered to address particular holon message-passing protocols.

Developers will be able to express complex holon behaviors as goals and procedures in OMAR. The invocation of goals and procedures takes place through a message-passing protocol; procedures are invoked, execute, return a value, and report on the success or failure of the invoked behavior. Developers may also define holon behaviors by writing CLOS generic functions for holons as agents. The special parent-child relations of holons in a hierarchy may motivate extensions to the message-passing protocol. As a particular case, it might be useful for

a holon to be able to send a message to each of its children. This ability, and the few similar message broadcasts within a holon hierarchy, should be easily accommodated. A certain amount of caution is required on the part of the developer here because side effects can make results dependent on the particular order of message invocation.

The extensions suggested for passing messages within a holon hierarchy relieve the developer of explicitly naming the target holons. However, holons may pass messages across a hierarchy (Young, 1992). In the past, "acquaintance" lists have been used for agents as a means to track remote objects within a SCORE simulation. They take advantage of the capability to name objects and locate them by name. The acquaintance list starts out as a list of the names of other objects, such as holons, and the actual object references are cached as they are located.

Graphical Presentation of Holon Hierarchies

OMAR has the capability, through the Concept Editor, to present and allow the graphical editing of SFL concept definitions. The traditional graphical view of the concept hierarchy follows the *is-a* links of abstractions and generalizations that are built into SFL. Alternate user-defined networks, such as *part-of* and *has-parts*, can be graphed similarly.

OMAR SOFTWARE SPECIFICATIONS

This section specifies the integrated suite of software tools that were developed for the OMAR programming environment so that HPP models could be built. The effort described would not have been possible without the existence of a substrate of components developed during previous BBN efforts. Where such an initial substrate existed that satisfied, or nearly satisfied, a requirement, that substrate and any modifications made to it to ensure adequate fits to requirements will be described.

The following subsections discuss: (1) the system architecture and major software components that make up the OMAR HPP model development test-bed, (2) details on the individual software modules, (3) the knowledge-acquisition environment, (4) the simulation run-time environment, and, finally, (5) the software support layer for user interface development.

The OMAR Architecture

OMAR consists of a carefully chosen and well-organized collection of integrated software packages. As indicated in Figure 4, the centerpiece of a modeling effort is an HPP model. A target system model is created and scenarios are generated to drive an investigator's experiments. The "psychological reality" of the models is made possible by the provision of a select set of representation languages and a knowledge-editing environment that includes graphical editors, textual editors, and graphical browsers. Compilers are available to generate executable code for the simulator. The interactive simulator includes on-line analysis and animation capabilities that future HPP model developers may tailor to their requirements. Finally, data collected by the simulation recorder is available for on-line or post-run analysis.

The next two subsections provide an overview of the OMAR software system from two distinct architectural viewpoints: (1) as a functional decomposition and (2) as a layered simulation system.

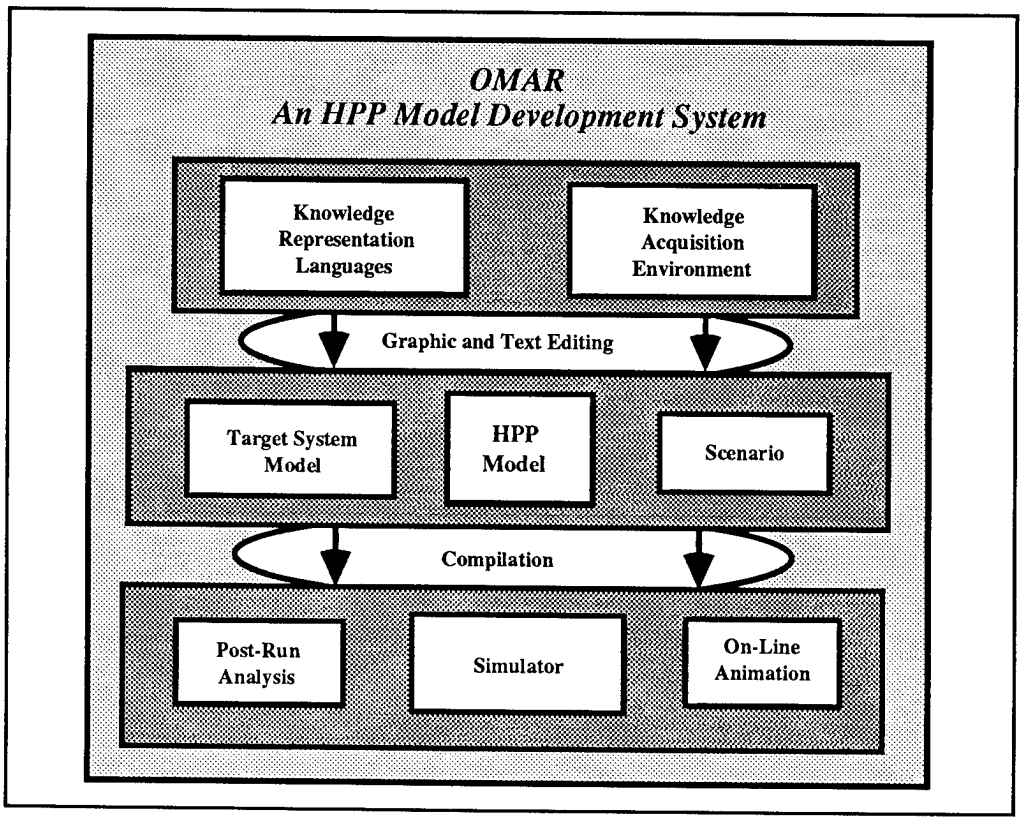


Figure 4
OMAR as a Human Performance Process Model Development System

Functional Decomposition Overview

OMAR software tools have been designed to address five distinct functional requirements (Figure 5): knowledge representation, knowledge acquisition, definitional forms, compilers, and simulator. Each of these is described below.

Knowledge Representation. The knowledge-representation tool is a set of languages whose semantics enable developers to compose OMAR HPP models and their behaviors, and target systems and their behaviors. There are three representation languages, each of which meets a particular OMAR requirement:

- SFL (Abrett *et al.*, 1990),
- SCORE language (Deutsch and Palmucci, 1992), a behavior representation language for an agent's goals, plans, the instantiation of plans as tasks (i.e. networks of procedures), and the forms that the contention between tasks will take, and
- McRule, a rule-based language.

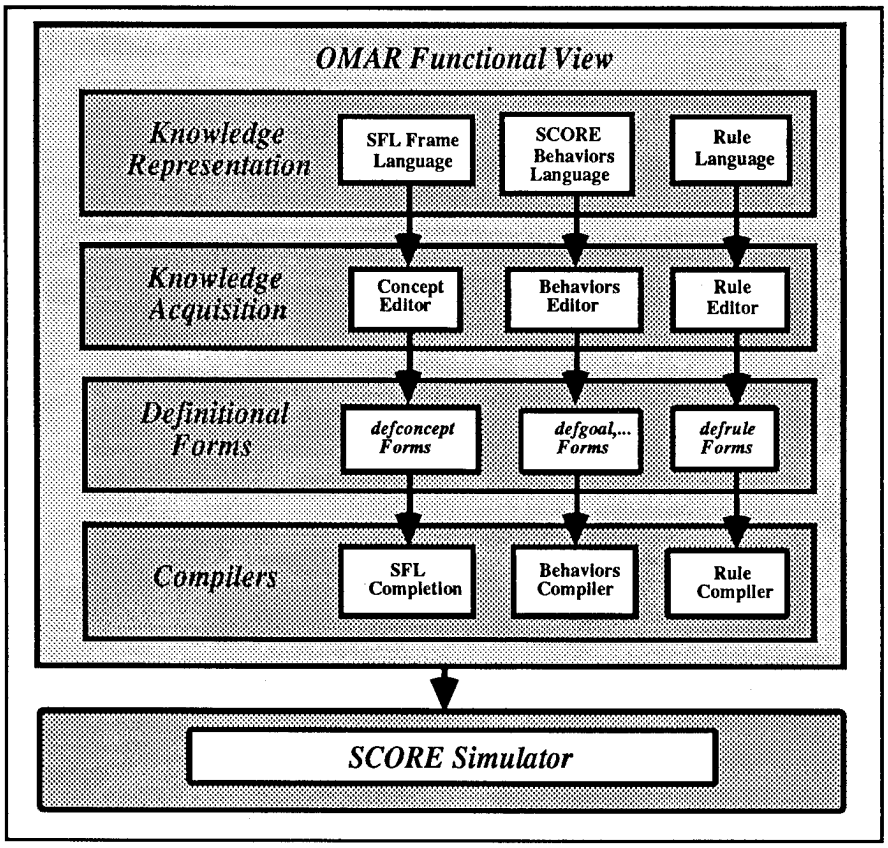


Figure 5
Functional Decomposition of OMAR

Knowledge Acquisition. The knowledge-acquisition tool is a graphical and textual editing environment whose purpose is to assist and simplify the process of creating the forms that define objects and their behaviors in the syntax required by the representation languages.

- A graphical editor, the Concept Editor, for the SFL is used to create *defconcept* and *defrole* forms. The Concept Editor provides users with a straightforward way to define object classes and revise their definitions as necessary.
- A textual editor is used to create SCORE forms and a graphical browser, the Procedure Browser, is available to examine individual procedures and the networks of procedure-calling sequences and signal interactions.
- A textual editor is used to create McRule forms.

Definitional Forms. Definitional forms are OMAR language forms produced by developers using the knowledge-acquisition environment to define HPP models and their behaviors, target system entities and their behaviors, and the scenario for exploring agent behaviors.

Compilers. Compilers are used to process individual language *def-* forms and to create run-time objects and their behaviors. The compilers enable developers to locate syntactic errors in their code and to generate efficient run-time code.

- The SFL define function turns *defconcept* forms into concept objects.

- The SCORE compiler turns *defgoal* and *defproc* forms into executable procedures and identifies the class objects to which they belong.
- The compiler for the rule language turns *defrule* statements into executable and efficient LISP code.

Simulator. The simulation is an event-based simulation that plays out the behaviors of HPP models in their interaction with target systems. The essential components of the OMAR simulator are:

- SCORE simulator that executes HPP agent goals, plans and their instantiation as tasks and their procedures, and the forms that the contention between tasks will take,
- simulation event processing, and
- simulation animation and trace tools.

OMAR as a Layered Simulation System

The layered simulation system view (Figure 6) of OMAR is a view orthogonal to the functional decomposition view. From this viewpoint, OMAR can be thought of as being built from basic computer science software components that provide a series of layers of fundamental simulation functionality. These layers are the Core Simulation Layer, Perceptor/Effector Model Layer, Plan/Task Layer, and Goal-Directed Agent Layer.

The Core Simulation Layer contains the basic building blocks out of which the cognitive modeling layer and the actual simulation application can be constructed. The more psychologically real cognitive layers sit above the Core Simulation Layer and rely on it for efficient execution. The components at this level are:

- SFL,
- SFL Completion Algorithm (concept compiler),
- SCORE procedure representation language and its compiler,
- Rule language and its compiler,
- knowledge-acquisition environment (editors) for frames, rules, and SCORE procedures, and
- SCORE simulation and rule execution engines.

The Perceptor/Effector Model Layer is built on the Core Simulation Layer. As its name implies, it is the place-holder for the preceptor and effector components of HPP models. The degree to which these models are elaborated is dictated by the purposes of individual HPP modeling efforts. For current purposes, this layer consists of a series of ports to and from the cognitive processor, including eyes, ears, hands, feet, and a mouth. The models are edited textually and a browser is available to examine their networks of procedures.

The Plan/Task Layer is built on top of both the Core Simulation Layer and the Perceptor/Effector Layer. Tasks are psychologically real instances of plans that human agents execute. This layer consists of:

- Plan/Task Representation Language components of SCORE semantics to express resource usage and time constraints,
- a set of software components from which to develop models of tasks and resource contention,

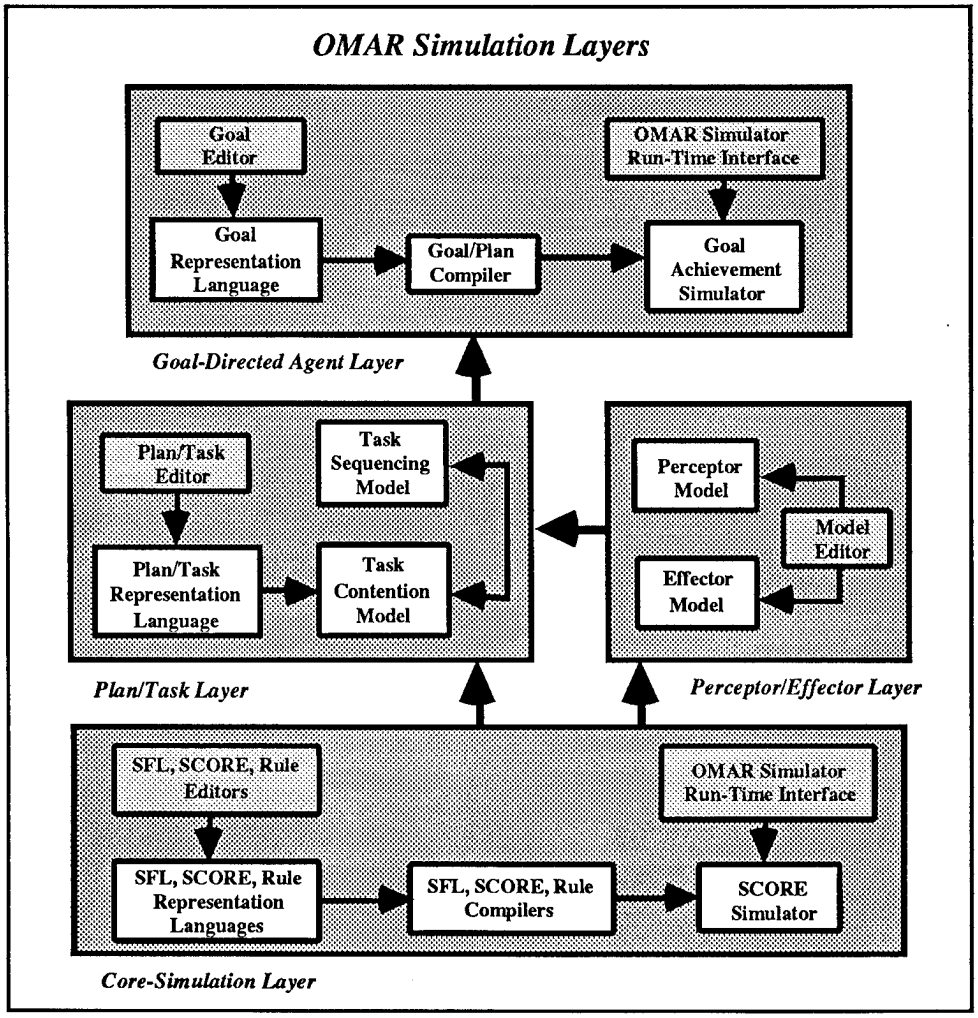


Figure 6
Layered-System View of OMAR

- compiler components that will produce plan objects that the simulation will execute, and
- textual editing of plans and tasks with a graphical browser available to examine the resulting procedure networks.

The Goal-Directed Agent Layer is built on top of the Plan/Task Layer. Goals represent the proactive component of HPP agent behaviors in OMAR. This layer facilitates the representation of an agent's goals and the simulation of the attempted achievement of these goals. This layer consists of:

- a basic simulation agent from which HPP agents may be developed,
- Goal Representation Language components of SCORE semantics that will express agent goals,
- textual editing of goals with a graphical browser available to examine the resulting goal and subgoal networks,
- compiler components that will produce goal objects that the simulation will execute,

- simulator components that will manage the achievement of agent goals, task contention, and task execution, and
- a run-time simulation interface to manage OMAR simulation execution.

Details of the Software Components of OMAR

The following discussion interweaves the functional decomposition and the layered system views.

Core Simulation Layer

This section addresses SFL, procedure representation components of the SCORE language, the rule language, their compilers, and the SCORE simulation engine (Figure 7). Editors for these languages are discussed later in this document.

Simple Frame Language and Common LISP Object System. A fundamental requirement for OMAR is an object-oriented programming substrate. CLOS (Steele, 1990) is an obvious choice because of its combination of robustness, power, efficiency, and ubiquity in the LISP community. Commercial CLOS systems are available for all hardware platforms that are considered for OMAR.

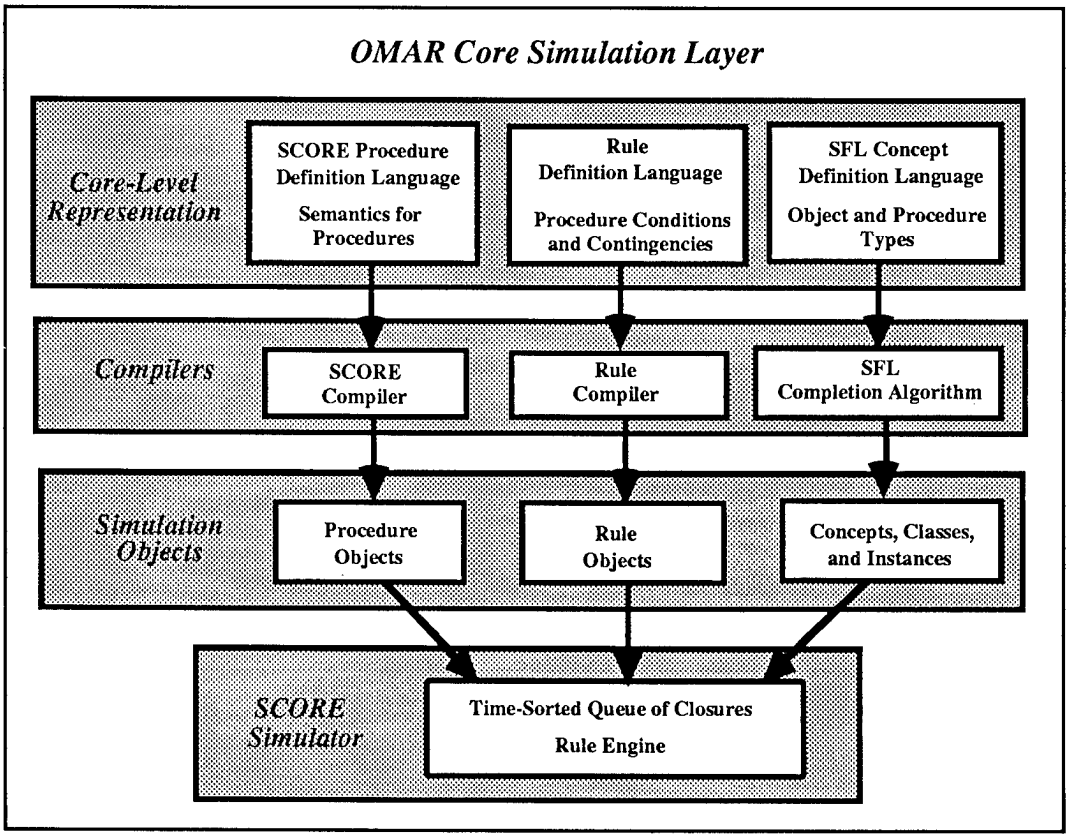


Figure 7
OMAR Core Simulation Layer

CLOS was designed as a core object-oriented programming language. HPP modeling in OMAR requires a frame language for knowledge representation. Traditional AI spawned frame languages, such as BBN's Knowledge Language (KL-ONE) (Brachman & Schmolze, 1985), and a level of definitional clarity to the object-oriented paradigm that is essential for modeling human behaviors. Unfortunately, traditional frame languages, while powerful for defining entity types, are inefficient at managing the run-time world composed of instances of those types. SFL is a child of KL-ONE via another BBN system called Knowledge Representation Editing and Modeling Environment (KREME) (Abrett & Burstein, 1987). SFL adds the following to CLOS slot semantics: value restrictions, number restrictions, first class relations, and semantically well-defined slot defaults. It incorporates a frame completion algorithm that deals with the complex inheritance protocol for value restrictions and defaults, and turns *defconcept* forms into concept objects. SFL uses CLOS to instantiate concept objects and instances of concept definitions. It directly connects, with no loss in efficiency, to CLOS's highly optimized run-time environment (e.g., object instantiation and method combination).

SCORE Semantics and the SCORE Compiler. Common LISP is an inherently serial language designed to operate on serial machines. LISP semantics define the strict order in which each step of a function is to execute. This limitation of serial execution complicates the design of LISP-based simulations that attempt to model several activities running in parallel — programmers must include their own code and data structures to execute and control concurrent activities. This has several disadvantages:

- it increases development time because concurrency control is a nontrivial task,
- it makes the simulation less robust because an error in the parallel semantics for a simulation can cause frustrating and transient bugs in the simulation, and
- it makes the code hard to read because LISP is not designed to run in parallel and has no language structures with which to represent concurrency.

The SCORE language is an extension to Common LISP that removes the disadvantages of an ad hoc approach to parallel simulation. SCORE directly addresses the complexity of simulating concurrency in large-scale LISP simulations. Its procedures have robust parallel semantics and provide an intuitive syntax in which to express parallel execution. These procedures are compiled into LISP code that runs very efficiently in the SCORE simulator.

While SCORE's principal purpose is to manage concurrency in a LISP-based simulation, it must also interface well with other subsystems in the LISP environment. To meet this requirement, SCORE is designed to be symmetric in the sense that it is easy to embed LISP code in SCORE, as well as to embed SCORE code in LISP. Any simulation code that is accessible from a LISP function will be equally accessible from a SCORE procedure. Similarly, a programmer can place SCORE code wherever LISP code can reside, such as in the action clause of a LISP-based rule system or in a method of a CLOS object. Most knowledge-modeling systems that are designed to work in the LISP environment also work, without change, in SCORE.

SCORE is a compiled language. The SCORE compiler generates LISP code that is, in turn, compiled by the underlying LISP compiler. The primary advantage of this is performance. SCORE procedures compile into very efficient LISP code. In fact, it is not likely that a programmer, implementing the same parallel semantics by hand, could consistently match the performance of the SCORE compiler.

A secondary advantage of a compiled language is compile-time error checking. Many programming errors can be caught as soon as the compiler processes the code rather than later,

when the code is actually executed. Bugs are easier to catch, and development time will be reduced.

Just as SCORE is designed to be extended externally by other systems in the LISP environment, it can be extended internally through the definition of new language primitives. The SCORE compiler consists of a top-level parsing component that calls subcomponents for each language primitive that it encounters. To define a new language primitive in SCORE, a programmer simply needs to define the relatively small portion of code that translates the new SCORE primitive into the equivalent LISP code. The SCORE system provides a macro, *defexpander*, to hook this new code into the compiler.

The current semantics of the SCORE procedure language forms a very robust and extensible simulation language. By taking advantage of the extensibility of SCORE, goals, procedure contention, and procedure suspension and resumption have been implemented as additions to the basic SCORE language.

Rule-Based Language. The initial rule-based language to be used for OMAR is a forward-chaining language. Rules are assembled into rule packets, forming a basic OMAR simulation object type. The rule packets are invoked as generic functions with arguments. Variables in rule clauses are capable of referencing SFL-defined objects; in particular, they are able to reference SCORE procedures.

This last point is particularly important: tasks that model deliberative agent processes will frequently be called upon to reason about what is to be done next, that is, to reason about procedures. SCORE representations of goals and plans can be expected to frequently invoke rule sets to declaratively represent this decision-making process. A rule compiler, closely related to the SCORE compiler, operates on rule sets, generating run-time objects and efficient executable code.

The procedures that invoke rule sets define the context in which the rule sets operate, a burden that must usually be covered by additional *if-* clauses in the rules. Hence, each rule packet may be expected to be made up of a few rules that focus on domain issues. In spite of these anticipated simplifications, the tracing of rule firings remains important. The earlier SCORE event mechanism used to record and support the display of goal and procedure execution has been extended to supply this important feature for the rule system.

SCORE as the Core Layer Simulator. The SCORE simulator is a discrete event simulation engine. At its heart is a time-sorted queue of future events. As procedures execute, they place events on the future-event queue. The events themselves are executable objects in the form of closures. A simple way to think about closures is as functions whose arguments are bound at the time the closure is made but which can be executed later. The simulator pulls the top event off the queue and executes it. That execution often causes additional events to be placed in the queue. The use of closures in a time-sorted queue enables individual threads of execution to occur asynchronously and simulates the parallel execution of agent tasks.

SCORE also implements asynchronous event-monitoring. Procedures can associate an executable closure with a specific condition. The simulator maintains a hash-table of executable closures (things to do) and their triggering conditions. Other procedures can "signal an event," causing all closures (the things to do that are associated with the event) to begin execution. This construction provides a very natural mode of communication and interaction among sets of concurrently executing procedures.

The SCORE simulator is the basic discrete event simulation engine for OMAR. Attention has been devoted to improving its run-time debugging environment and to improving its

efficiency, but little more has been needed to use it as the generic Core Simulation Layer for OMAR.

Core Simulation Layer Summary. Summarizing the above, the OMAR Core Simulation Layer includes a parallel procedure language whose semantics covers multiple agents executing multiple asynchronous tasks. The tasks can communicate with one another using an asynchronous event paradigm. A forward-chaining rule language is available from the procedure language and is used to represent contingencies and preconditions declaratively. Accompanying the procedure language, as well as being a foundation for it, is the SFL, with its well-established semantics for representing frames. Companion compilers for the rules, procedures, and frames produce efficient executable code. The engine that carries out the simulation is simple and efficient. Contention between procedures and the suspension and resumption of procedures are addressed in a simulation layer that has been built on this substrate.

The Cognitive Level

For the purpose of developing HPP models, the most important simulation layer is the Cognitive Layer. This is the OMAR Simulation Layer that contains the collection of tools that is to be used in attempts to mirror, in a more psychologically real manner than can be achieved in the Core Simulation Layer alone, the cognitive processes of HPP agents. The Cognitive Layer is made up of the Goal-Directed Agent and Plan/Task Layers of Figure 6. This simulation layer includes languages to describe agents, their goals and plans, and the tasks they undertake to achieve their goals. The OMAR Goal Achievement Simulator layer manages goal-directed behaviors as well as task sequencing, task contention, and task interleaving.

Agents. In OMAR, agents are entities capable of executing goals, plans, and tasks with their instantiations as procedures. In the most straightforward use of the OMAR simulation environment, an HPP model consists of a single agent. However, there is nothing to preclude assembling an HPP model from a collection of agents. In the discussion that follows, an HPP model is described as being composed of a single agent. This is intended to simplify the text, but should not be interpreted as a bias toward this approach to HPP model construction. In building HPP models, developers can be expected to parameterize agents, particularly along dimensions representing performance skill levels.

Goals, Plans, Tasks, and Procedures. The proactive component of HPP model behaviors is represented by goals assigned to an agent. A goal contains an explicit statement of what the agent has set out to do — the criteria for its successful completion. The particular circumstances under which the goal may become active is also stated for the goal, and a LISP form is established that is used to dynamically compute the priority, or activation level, to be associated with the goal in its bid to execute. If a modeler wishes to have several goals with the same success criteria, it is necessary that a method for selecting among them be devised.

The parallel or serial execution of the subgoals that form the plan for a goal (Figure 8) is described using the *race/join* semantics of SCORE. As outlined in the discussion of the psychological framework, the subgoals at the leaves of the goal tree are referred to as plans, marking the transition between thinking and acting. Task execution involves the activation of a plan and the procedures that it encapsulates — the agent's attempt to achieve a goal. The Goal and Plan/Task Representation Languages define the extensions to the SCORE Procedure Language layer that are used to express goals, plans, and tasks.

The Cognitive Level Simulation is driven by an agent's efforts to achieve its goals. The agent's attempt to achieve a goal generates one or more tasks which, in turn, activate SCORE procedures that execute in the Core Simulation Layer. Tasks, as the instantiation of a plan, activate a network of procedures, each member of which includes a wrapper whose purpose is to

deal cleanly and modularly with contention for resources and the interleaving of sequences of activities.

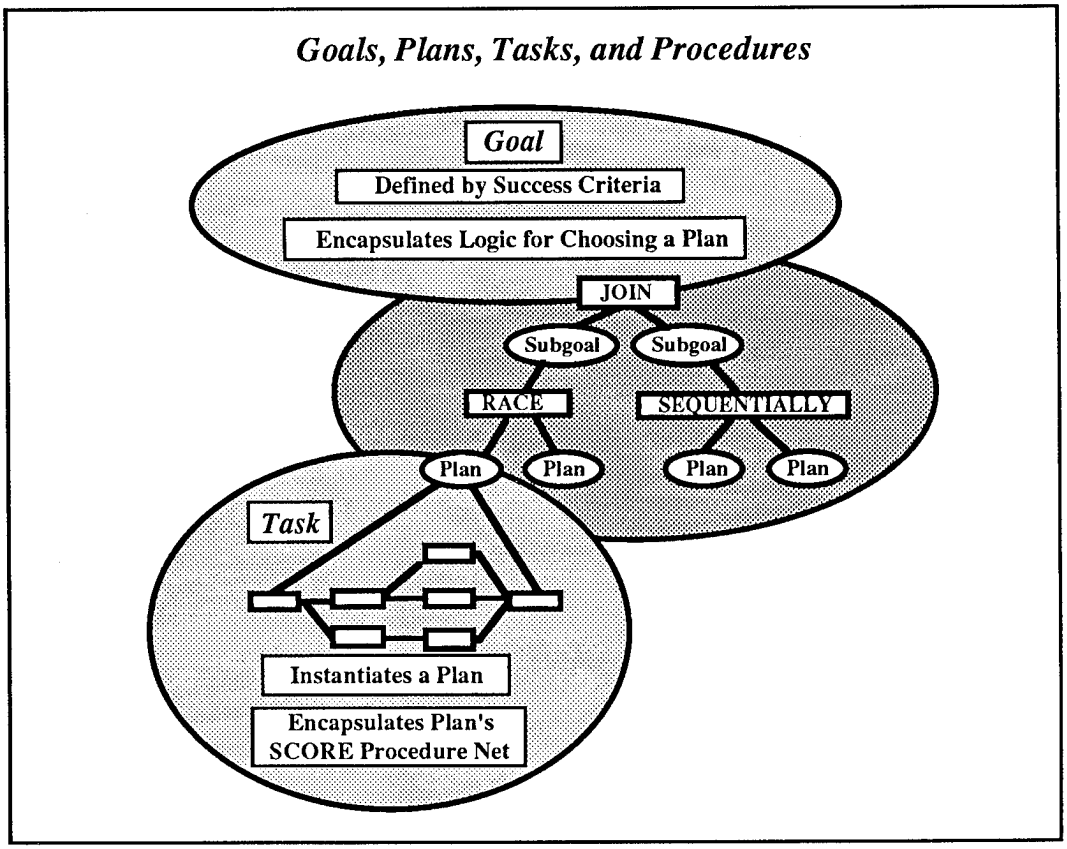


Figure 8
Goals, Plans, Tasks, and Procedures in OMAR

The Task Language is used to describe the resources that the procedures of the task require and the temporal constraints among the task's procedures. When compiled, a task encapsulates an executable SCORE procedure network and is ready to run in the Goal Achievement Simulator as a discrete activity in the service of accomplishing a specific agent goal. When tasks execute in the Goal Achievement Simulator, they are primarily concerned with resource management and time-constraint sequencing. The SCORE procedures encapsulated by a task execute in the Core Simulation Layer after the Goal Achievement Simulator has ascertained that the task has met its resource and time management constraints.

The Goal Achievement Simulator. The purpose of the Goal Achievement Simulator is to implement the agent's pursuit of its goals. When a goal is activated, code specialized by goal type and using information stored with the specific goal object selects a plan to attempt to achieve the goal. As described above, the goal consists of one or more subgoals with plans at the nodes. Temporal precedence is also expressed among the subgoals and plans.

The achievement of the goal's success criteria determines the success or failure of the goal. The completion of all tasks driven by a goal-tree does not necessarily imply that the goal has been achieved. Conversely, a goal can be achieved even though some of its tasks are still in

process. The Goal Achievement Simulator monitors the world for conditions under which a goal might be achieved.

The Goal Achievement Simulator attempts the execution of the subgoals and plans that make up the tree for the goal as dictated by the temporal precedence relations. The instantiation of plans as tasks contains information about the resources (e.g., eyes, ears, hands, cognitive processing) required by the task and about sequencing the task's procedures in time. When the Goal Achievement Simulator determines that the procedure of a task is qualified to execute, it invokes that SCORE procedure in the Core Simulation Layer.

Task Contention. Multitask behavior is grounded in a framework of tasks that contend for execution along several dimensions. Many tasks need the services of a particular resource. "Eyes" provide a good example. Each of the visually based tasks may be defined to belong to the SFL class of "eye-needing" tasks. As another example, a task that requires an agent to speak may conflict with a task requiring the agent to listen. In these cases, there is contention between classes of tasks. OMAR contains basic tools that allow for mediation among these contending classes to be modeled. When there is competition between tasks, the appropriate contention model determines which task should run and which should wait. Examples of issues that contention models can be made to take into account are:

- the relative dynamic priorities of competing tasks,
- the class membership of the competing tasks,
- how close the task is to completion, and
- the cost of interrupting an ongoing task (How much inertia does it have? What is the cost of restarting the task?).

Deliberative Task Scheduling. OMAR agents must also be capable of thoughtfully interleaving a modest number of tasks in a realistic manner. This is of particular interest in modeling an agent's management and coordination of work to be performed by several crew members (Orasanu, 1990). Task contention heuristics alone are not sufficient to provide this capability. Task contention alone provides a reactive model of agent behavior but does not provide the deliberative facility by which an agent can organize his or her future actions. An approach to meeting this need, which has been used in related work (Abrett, 1991) and could be adapted for OMAR, is described here.

Humans interleaving multiple tasks over time often rely on some form of loose agenda for a rough idea of the sequence of tasks they will be doing in the future. This agenda of future activities can be used as an information resource to help guide decisions about what to do next.

In addition to a contention model for the resources that tasks use, the Goal Achievement Simulator might contain a representation of thoughtful task scheduling. Among other things, this scheduling module would act as a "knowledge resource" for an agent so that the agent can know what it plans to do in the future and can use this knowledge as the basis for informed decision-making on what it wants to do in the present.

The implementation would work as follows. Certain kinds of tasks that are concerned with making choices about future activities exist. Running those tasks results in the creation of a task schedule item that is placed on an informal, fine-grained schedule of future agent activities. The heuristics that determine how future task items are placed on the scheduler are partitioned by task type and address issues of task length and agent location.

The schedule is informal. It does not determine whether or when a task will run; rather, it acts as a knowledge resource, providing other tasks with a view of what the agent will be doing in the future. The deliberative scheduling module presents one approach to providing an HPP

agent with the ability to thoughtfully interleave a multiplicity of tasks in an intelligent and human-like manner.

Knowledge Acquisition Environments

Overview

The OMAR knowledge-acquisition environment includes textual editors, graphic browsers, and graphic editors. Graphic editors and browsers are available where they can be particularly effective for operating on large networks of objects. Browsers have been chosen over graphic editors in cases where (1) developers can profit from a network view but network editing is not essential or (2) network editing requirements are beyond the scope of the present effort. The Concept Editor for creating HPP agents and target system objects is a graphic editor. A graphic browser is available for the networks of HPP model goals, plans, tasks, and procedures, but the editing of these objects is performed textually.

The Concept Editor

In the OMAR knowledge-acquisition environment, the SFL has a graphic editor to assist developers in managing the large inventories of object definitions that may be required for HPP model creation. The capability to graphically edit objects speeds the process of definition, provides a unique view of the large-scale structure of the knowledge base, and facilitates significant restructuring of object knowledge bases that could not be accomplished in a text-editing environment.

The primary function of the Concept Editor is to support the definition of concepts and roles, the building blocks of object definitions. New concepts can be created by selecting the New Concept menu item and entering the requisite information. Concepts are then grounded as CLOS class definitions. Roles, the basis for concept slot definitions, are similarly created. The basic network presentation for concept hierarchies is maintained among the abstraction/specialization links — the "is-a" hierarchy. The real power of the Concept Editor lies in its support for managing the large taxonomies of Concept Editor concept definitions. The Concept Editor contains editing tools that allow modelers to rapidly rearrange networks of concepts by graphically rearranging the abstraction/specialization links among them.

The network of Concepts in a given application can often be very large; thus showing the entire tree of Concepts can become unwieldly at times. As a result, several tools have been used to aid the developer in "navigating" the tree. The most important of these tools is the notion of a "Focus" node. When the graph is focused on a Concept node, a portion of the tree that contains the focus-node, the parents and additional ancestors (abstractions), and the children and further progeny (specializations) of this focus concept is shown. Concept graphs can also be easily panned and zoomed.

In addition to the graph being focused on a node, information specific to a concept can also be displayed (this "edited" concept need not be the same as the concept on which the graph is focused). In particular, a table of slots for the edited concept is displayed in a separate pane. Tabular displays present all the slots of a concept (including slots inherited from ancestor concepts) or just those locally defined. For each concept slot, the Concept Editor table entry presents the role-name (slot-name), the source of the slot definition, the value-restriction and number-restriction for the slot, the default value, and the textual description for the slot.

The Concept Editor maintains a stack of concepts that have been focus concepts during the editing session. When a concept is modified, this fact is noted in the stack entry for the concept. This stack is visible at all times, showing the modification status of the concepts in it.

Furthermore, concepts appearing in it are available to be selected to be either the focus concept (in the graph) or the currently edited concept.

The Concept Editor provides several means for selecting a new focus concept. Concept names appear in several forms on the editing screen and each instance may be used to affect the selection of the concept for either focusing or editing. Concept names appear at the nodes of the concept graph (as value restrictions on concept slots) and in a push-down stack of concepts already edited. In addition, a global menu item, Edit Concept, is available. This item will prompt for the name of a concept to edit when the menu item is selected.

Global operations are available for reading in knowledge bases of concept definitions from multiple files, integrating new knowledge bases with knowledge bases that have previously been read in during the session, and saving knowledge bases to files. Facilities exist to partition large collections of concept definitions into manageable and independent knowledge bases.

Roles are the binary relations that are the basis for slot definitions. The graphical and textual operations available for roles closely follow those for concepts. The Concept Editor provides the capability to create roles, edit roles, and graphically review and edit the role hierarchy. Tables which list the concepts that define slots for a relation and show the inverse relations (concepts that use the focus concept as a value restriction in a relation) for any concept are available.

Taken together, the facilities of the Concept Editor compose a rich and powerful environment for the creation and editing of large networks of Concept Editor concepts and roles.

Editing and Browsing Networks of Goals, Plans, Tasks, and Procedures

Creating and modifying the definitions of individual OMAR goals, plans, tasks, and procedures are accomplished with the aid of a textual editor. Following good programming practice, these modules can always be expected to be significantly less than a page in length. However, there is another dimension along which the complexity is expected to be far greater — the network structure formed by the totality of an agent's goals, plans, tasks, and procedures. (NOTE: This section has been made more readable by defining the term "procedures" to include "goals, plans, tasks, and procedures" wherever the context can be expected to make the actual reference apparent.) The ability to graphically browse this network structure at selected levels of abstraction and from differing perspectives provides considerable assistance during the creation of HPP models.

In the current system, HPP model developers will necessarily have to employ both graphical browsers and textual editors during creation of their models. The formulation of a graphical editor (based on either a "drag-and-drop" or "structure-editing approach) by which to create and modify the procedures and, in particular, by which to graphically manipulate their network connections would be a highly desirable extension of the basic capability.

The Procedure Browser enables HPP model developers to view large networks of procedures from a variety of perspectives and provides network navigation facilities. Procedures are connected to one another in two distinct ways: as procedure and subprocedure and as signaling procedure and signaled procedures. The browser provides views of a network from each of these perspectives and an easy-to-use, consistent mechanism for switching among the various views.

As in Concept Editor, the Procedure Browser has a single focus procedure. The standard view of the focus procedure shows the procedures that call the focus procedure and all the procedures that are called by the focus procedure (i.e., the procedure-subprocedure network around the focus procedure). An alternative view shows the connections between the focus

procedure and the procedures it interacts with via the asynchronous signaling mechanism. "Antecedents" of the focus procedure in this viewpoint are those procedures that signal the focus procedure. "Descendants" are those procedures that are signaled by the focus procedure.

In addition to the network around the focus procedure, the Procedure Browser makes available a graph of the focus procedure itself (i.e., a graphical presentation of the textual form of the procedure). This graph resembles a flow chart and makes obvious the logic that the procedure embodies. There is immediate access from the graphical presentation of procedure networks to the textual code to facilitate modifications to procedures.

As in Concept Editor, there are several ways to select a new focus procedure. Procedure names appear in several forms on the screen and each may be used to effect the selection of the procedure as the focus of the Procedure Browser. Procedure names appear at the nodes of the various network graphs as well as in the structure graph of the current procedure. A "history stack" of previously viewed procedures, as well as a complete listing of all available procedures is displayed. Any representation of any procedure can be clicked on to bring up that procedure in the current (or any other view). In particular, clicking the right mouse button over a representation of a procedure brings up a menu of all available views.

The graphical presentations of goals, plans, and tasks are specialized to reflect their particular syntax and semantics. The success and initial conditions are presented for goals, as well as the structure of the graph and the temporal precedence relations of the nodes of the graph. For plans and tasks, graphs that indicate the contention that exists among the procedures that they encapsulate are presented.

The graphical browser for procedures is an important first step in the development of the future graphical editing environment for HPP representation languages. The Concept Editor serves as a model for this future effort. Within the present OMAR effort, the Concept Editor and Procedure Browser are expected to play an important role in the collaboration among experts from several domains working to develop HPP models. The graphical presentation of the actual code will provide insight into the structure of the developing model at any of several levels of abstraction.

The OMAR Run-Time Environment

The OMAR run-time environment contains the set of software tools that allows investigators to execute the goals, plans, and procedures of an OMAR HPP model; view the interaction of the model with the target system in the ongoing simulation; and obtain and evaluate experiment results.

Modularity in the software design for OMAR is a particularly important aspect of the run-time environment. As a software development environment for producing HPP models, OMAR is a complete simulation environment (Figure 1). It provides an integrated collection of software tools for creating HPP models and the target system model that the HPP models must interact with, scenario development capabilities, a simulator and run-time environment in which to exercise and evaluate an HPP model, and a post-run analysis capability to further support model evaluation. Beyond the term of the immediate project, OMAR is expected to support the development of HPP models for use in the OASYS simulation environment (Figure 2). The focus of this section is on the specifications of the run-time environment for OMAR. Several of the functions needed for OMAR as a stand-alone research tool for HPP model development will be subsumed by the OASYS system. The modularity in design and implementation of the particular run-time features described here is intended to facilitate the operation of OMAR in an independent stand-alone mode, while laying the groundwork for the operation of OMAR-developed HPP models in the OASYS environment at a later time.

Scenarios

Scenario setup requires the selection and initialization of the scenario agents and, the target world system, and the establishment of a set of events tailoring the scenario to the investigator's particular goals for the experiment. The SCORE simulation language provides a *defscenario* form to meet these needs. The *defscenario* form, much like the *defproc* form, provides the programmer with the capability to select and initialize HPP agents for the simulation run, select and initialize target world systems, and establish target world procedures to meet the needs of the particular scenario.

The Simulation Window

The simulation window provides for control of the selection and execution of a scenario, space for a trace of scenario events, and animation of HPP agent actions and those of the target system. In the ATC domain, prime candidates for the trace pane are the dialogues among ATCs and between ATCs and aircraft in their sectors. A menu that will allow the selection of the scenario to be run or the rerunning of the current scenario, the running and pausing of the simulation, and the stepping of the simulation will be available. Stepping of the simulation is a useful debugging tool for HPP model development. Control of the OMAR simulation has been developed in such a way that it can be invoked via a simulation window menu or from a second driver, such as the OASYS simulator.

Events at Run-Time

The signaling of events marks significant HPP agent actions (e.g., the beginning or end of a procedure, or the completion of a goal). The events are specialized and used, for example, to note communications among ATCs and between ATCs and aircraft in the airspace. Print methods that form the basis for presenting trace material in the simulation window have been written for events. To maximize the usefulness of the trace material, the simulation run-time menu makes it possible to control the content of the on-line trace by enabling the selection of event types to be presented in the trace and the selection of the agents to be traced. Event recording will be the basis for post-run analysis. Again, the particular event types to be recorded for later analysis will be available for selection through a menu at any point during a simulation run.

Human Performance Process Model Interaction with the Target System

The interface between the HPP agent and the target system model is object-oriented (Figure 1): data structures as objects with generic functions are available as needed. A SCORE layer between the HPP model and the object-oriented interface provides HPP-model access to the target system through SFL objects using SCORE procedures. In the future, when an OMAR HPP model is expected to run in the OASYS simulation environment, the OASYS side of the OMAR object-oriented interface will be adapted to meet the requirements of the OASYS Network Protocol (Figure 2). The HPP model will have the same object-oriented view of the target system in both the stand-alone OMAR simulation environment and the integrated OASYS simulation environment.

Timeline Inspector

Timelines are traditional means for presenting a task analysis. The OMAR simulator has a timeline as one form of presentation for HPP agent actions. Several extensions to the usual timeline presentation make this display particularly valuable as an analysis tool. The most significant enhancement over capabilities contained in the initial BBN substrate is provision of means for following the causal chains leading to agent events both forward and backward in

time. Additional HPP agents playing a part within the presented causal chain can be added to the timeline as their roles are uncovered.

The timeline display is available for post-run analysis or at any point in a simulation. During a simulation run, it is available as an option from the display associated with any simulation event in the simulation run-time window. For example, in an ATC simulation, the ATC radio messages to an aircraft might be presented as text on the screen as part of the animation of the communication appearing in the simulation window. Once the message has been selected using the mouse, the timeline display for the ATC would be presented. When the user selects the event detailing the radio message and follows the causal thread forward in time, the timeline will be extended to include the aircraft crew members that heard the radio message.

The User Interface Support Layer

The OMAR user interface includes graphical editors and browsers, timeline displays of HPP agent activities, presentations of the multitasking performance of HPP agents, and animation of scenarios at run-time. To support the orderly implementation of these "applications," a User Interface Support Layer has been developed (Figure 9).

This support layer is also intended to ease the implementation of a consistent look and feel across all components of the OMAR user interface. In the current implementation, the support layer consists of a user interface toolbox for the developers of the OMAR system; eventually, essential parts of it will form a component of OMAR available to the developers of HPP models.

Components of the User Interface Support Layer

The OMAR User Interface Support Layer is tailored to support the development of the major components of the OMAR software development effort (Figure 9). Those major components are:

- the graphical Concept Editor for creating and modifying concept and role definitions,
- the Procedure Browser to provide programmers and nonprogrammers with insight into an HPP model's goals, plans, tasks, and procedures,
- table-editing facilities to permit the editing of individual table instances (e.g., slots on Concept objects),
- displays providing insight to the current state of the HPP model (e.g., timeline displays, simple animation of the HPP agent's behaviors, traces of the history of the model's state, graphs of agent performance measurements, etc.); and
- animation-based highlighting of HPP agent and scenario events at run-time.

The Grapher Interface. The grapher has the capability to present graphs with user interface operations available on the nodes and links. A variety of standard shapes will be available for graph nodes. SFL *is-a* hierarchies are acyclic, while procedure graphs include SCORE *loop* functions. The grapher meets the needs of each of these applications.

Of particular concern is the ability to "open up" a node in a graph, then display and modify the revealed substructure. Consider a display that shows a goal hierarchy at a very high level of abstraction. The user is able to examine a subgoal in a recursive manner, effectively walking down the network and opening subgoals while maintaining a sense of position in the network. Basic functions support the implementation of these applications.

Graphs can also become very complicated, requiring tools for clustering or collapsing in ways that are meaningful to the user. Functions are provided so that displays of subgraphs may be

opened and closed. It is also important that the user be able to navigate large graphs and return to a previous editing or browsing state conveniently. To facilitate this, tools for "push-down lists" of previously edited or visited objects are provided.

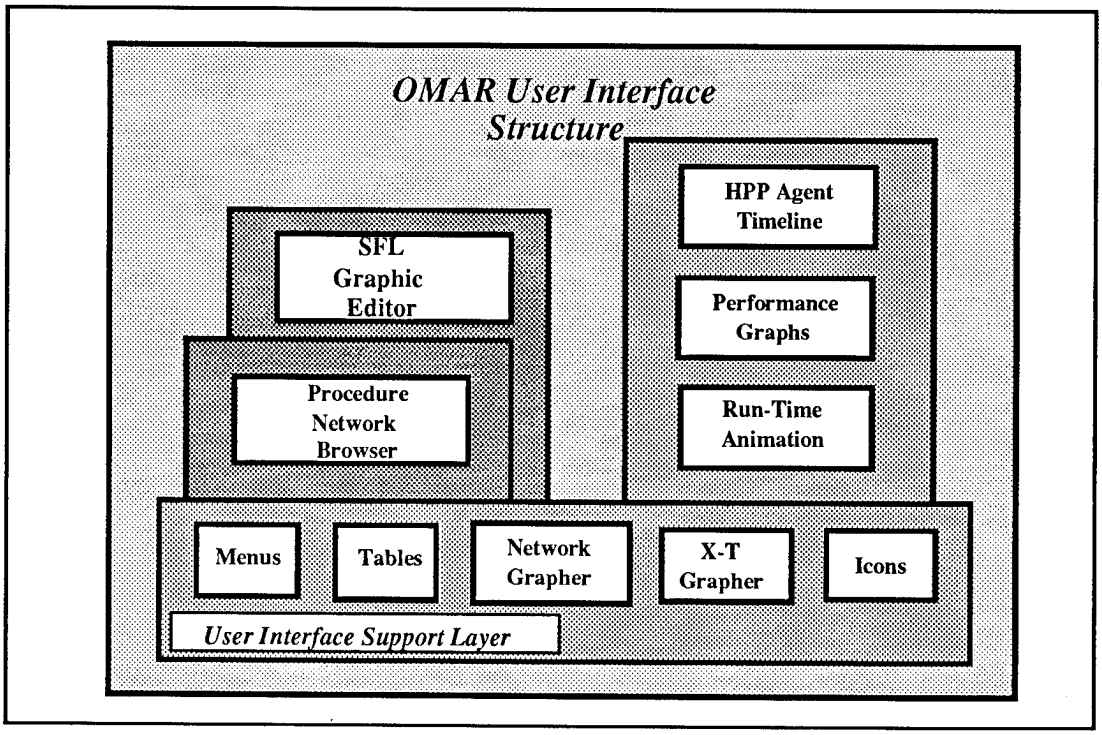


Figure 9
OMAR User Interface Support Layer

Tables. Tables are needed, particularly for the entry and editing of the slots of SFL concepts. Within the table for an SFL concept, each row contains a slot for the concept with column entries for slot name, value- and number-restrictions, default value, and slot description. Table-editing is performed with the aid of a mouse.

X-T Graphs. One mode of presenting HPP agent performance is by plotting parameter values as they vary over time (e.g., an X-T plot). Several straightforward variations on the basic theme of X-T plots are supported.

Menus and Icons. The Common LISP Interface System (CLIM) meets many of the requirements in this area. However, several features linking screen presentations to actions that control the simulation or impact the behaviors of HPP agents are desirable. In part, these features are directed towards encouraging a consistent look and feel in OMAR development (e.g., the left mouse button effects object selection in the Concept Editor and in the timeline display).

User Interface Design Considerations

The design of the User Interface Support Layer has been motivated, in part, by the generic problem of managing the presentation of large amounts of information and two related design considerations. First, displays should be constructed such that groups of entities can be collected to form higher-level constructs that are more readily presented. The layout, nature, and functionality of each display should make it easy to understand what has been clustered together to form the higher-level abstraction. The tools that support the clustering of objects creating

these abstractions enable users to move easily back and forth between levels of abstraction so that they may maintain their orientation.

The second design consideration is closely related to the use of abstraction to manage the presentation of large amounts of information: "We cannot display everything all the time." It is essential that information selected for display be minimized in a useful way. Leaving aside the issue of a necessarily limited amount of screen space, poorly designed, dense displays simply overwhelm the user. It becomes difficult to extract the required information from a cluttered interface.

The following principles have guided the design of individual displays and the overall approach to the OMAR user interface.

The amount of textual output on the screen shall be kept to a minimum and removed completely when feasible. The following approaches have been taken to minimization of text in displays.

- Highly "iconized" displays are used (i.e., information is presented in a graphically oriented manner utilizing color and shape). In particular, with a properly designed icon, a useful representation for a given object that requires substantially less screen space than a comparable textual description or label can typically be made.
- While it is not feasible to remove all textual descriptions from a display, it is typically possible to remove much of the textual description from a highly detailed display and place it in an associated "description" window. The detailed description of an object in the display can then be obtained by moving the mouse over the object.

The notion of additional information "on demand" can be extended in several ways.

- A number of auxiliary displays can be associated with a main display, each of which provides more detailed information about the main display (e.g., by extending the "mouse description" outlined above). Just as one level of description appears when the mouse is moved over an object, so a deeper, more extensive description will appear as the result of a mouse-click on the object.
- The "mode" of a display can provide customized presentations (e.g., alternating between textually labeled objects and text-free, highly iconized objects).
- The most current state information is generally adequate. Old or outdated information will be obtainable by the user, but any space taken up by displays, primarily textual displays, will generally be restricted to presenting the current state of the system. For example, long, scrolling, textual "Output History" displays, aside from being computationally expensive, are almost always a waste of valuable screen space. (There are a few specific situations in which such displays are actually useful for accomplishing a specific task — for example, to debug code. However, in the current implementation, such displays are available "on demand" rather than as the default display.)

Because of the complexity of the HPP modeling process, the many forms of the associated data, and the varied needs of HPP model developers, the interface tools include support for presenting a number of alternative visual perspectives into the data. Many objects are made accessible from more than one viewpoint, displaying different aspects according to the context of the display. For example, SCORE procedures are available for review in graphic and textual form, and an SFL concept may appear as a node in the large graph of a concept hierarchy or in full detail as the focus concept in the editor. The User Interface Support Layer provides for the creation of and movement between multiple views, while informing the user of the relationships among the various perspectives.

AN OMAR PERSPECTIVE ON THE OASYS-OMAR INTERFACE

The primary concern of this project has been the specification and realization of a tool suite aid in the implementation and test of HPP models (OMAR as represented in Figure 1). A longer-range perspective suggests it will be desirable to use one or more of the HPP models, developed using OMAR, as agents in an OASYS simulation (OMAR as represented in Figure 2). An immediate concern is the form the interface between OMAR and OASYS will take. This section provides an informal OMAR perspective on what that interface might be.

Currently, OMAR uses its own target system model, an object-oriented interface to the target system, and a simulation control module to handle the capabilities identified here. These modules were developed while attending to the OASYS interface design as it evolved. The final step of connecting OMAR simulation control and the OMAR object-oriented target system interface to the OASYS network protocol will be completed at such time as OASYS and OMAR are to actually run together.

The Network Layer and an Object-Oriented OMAR

The OASYS simulation environment has the potential to become quite large in the sense that it may operate across more than one machine. Hence, it is not unlikely that an OMAR HPP model will run on a machine connected to OASYS over a network. Discussions with OASYS designers have indicated that they envision a network protocol based on passing messages composed of text strings. On the OMAR side, there is a strong commitment to object-oriented design and implementation; thus, it will be necessary, at a future time, to complete OMAR as shown in Figure 2. An OASYS target system model will replace the OMAR target system model, and the OMAR object-oriented target system interface and simulation control module will communicate with OASYS through the OASYS text-based network protocol.

The remainder of this section outlines an initial OMAR view of the functionality that the network protocol ought to provide. Briefly, it will provide object data structures reflecting the organization of the target system as seen by the OMAR agent, object data structures reflecting the state of the target system updated as required by the actions of the OMAR agent, and notifications of selected events to which the OMAR agent may be expected to react. Generic functions will be necessary to gain access to OASYS data necessary to initialize an OMAR agent for a simulation run, to update and manage the OMAR view of target system data objects, to effect OMAR agent-directed actions on the target system, and to coordinate the synchronization of the OMAR and OASYS simulators.

Initializing an OMAR Agent for an OASYS-OMAR Simulation Run

From an OMAR perspective, initialization tasks fall into three categories: (1) obtaining the structure of the target system with which the OMAR agent will interact, (2) obtaining scenario-specific state data that the OMAR agent might be expected to know about at the beginning of a simulation run, and (3) informing OASYS of event types to which the OMAR agent might be expected to respond.

Target system structure refers to the functional and physical organization of the operator work place (e.g., that there is a panel with three switches, a meter, an alarm light, and details relating to each operational part). As envisioned by the OASYS development team, this work place can be restructured between simulation runs, hence OMAR will need access to the work place layout at initialization time. A novice OMAR agent may not "know" the exact layout of the work place, but this is a separate concern.

An OMAR agent about to take part in an OASYS scenario might be expected to know quite a bit about the situation at the start of the simulation run, particularly if the agent is an experienced operator. Much of the "know-how" of the OMAR agent comes prepackaged in the goals that the agent is prepared to attend to and in the procedures that the agent is capable of executing. The situation-specific data required by some of these goals and procedures will be needed so that these procedures can be "in process" at the start of the simulation run.

Run-Time Functionality

Run-time interface issues fall into two categories: (1) system functions, primarily coordinating the execution of the OMAR simulation with the OASYS simulation, and (2) simulation domain functions concerned with how the OMAR agent finds out about target system state and events, and how the OMAR agent effects target system actions.

Coordinating OASYS and OMAR Simulator Execution

The OASYS and OMAR simulators will be required to operate in fast-time and real-time modes. The singular real-time issue is whether each simulator will be able to keep up with the wall clock. If each can keep up with the wall clock as expected, simulator design problems will be significantly reduced. Coordinating the execution of the simulators in real-time will be revisited periodically as better estimates of the run-time performance of the simulators becomes available. The discussion here is restricted to the fast-time simulation case.

Parallel-distributed simulation is a field rivaling HPP modeling in its complexity; hence, it would be best to maintain the focus on HPP modeling and not attempt to address both fields. It is recommended that a particular local solution be sought for synchronizing the simulators; for example, running OMAR agents and the OASYS simulator on a single machine should be considered. The recommendation was not arrived at lightly. The SCORE simulator was implemented by a former member of the Actors (Agha, 1986) research effort in parallel computation, and there is considerable interest in parallel computation among those contributing to the design of OMAR. However, while SCORE resembles Actors in many of its features, SCORE was a several-person-month rather than several-person-year effort, largely because it was designed to operate on a single machine.

One relatively simple scheme by which the OASYS simulator and, perhaps, several OMAR simulators might coordinate their actions may be described as follows. An OMAR simulator would be invoked by OASYS with a future time to which it might run. The OMAR simulator would then run to the indicated time or until such time as it took an action that impacted the OASYS target world. OMAR queries about the target system state could be returned immediately by OASYS and, hence, not impact the basic synchronization scheme. Upon returning control to the OASYS simulator, the OMAR simulator, as an event-based simulation, would inform OASYS of its next scheduled-event queue-item time — the time at which it would next need to run.

OASYS then knows when each OMAR simulator next needs to run and runs forward to that time, with the following exception: OMAR simulators will have registered event types about which they must be apprised. The occurrence of one of these event instances requires that the OMAR simulators waiting on the event be notified of their occurrence.

One problem in this mode is that each simulator must be quite conservative in examining its pending event queue to determine when it must request execution again. Pending actions can be inspected, but it will not always be possible to determine whether a pending queue item needs the services of another simulator or is a purely internal matter on which execution may proceed. A more serious problem is that there may be little parallel processing advantage realized in spreading the simulators across more than one machine. Empirical evidence is needed here; the

fast-time performance of the simulators may well reduce this to a nonissue, or this issue may need to be revisited in greater depth.

Interacting with the OASYS-Modeled Target System

The OMAR agent must be able to scan the state of the target system that makes up the OMAR agent's work place, and the agent will need the opportunity to respond to work-place events (e.g., an auditory or visual alarm). The first involves the ability to query OASYS for the state of a particular object (e.g., a switch or meter) or composite object (e.g., a panel with several switches and lights). The query of a composite object state will be used by OMAR to support the visual scan of a panel of switches or a computer screen. These queries address the needs of the proactive sensory actions of the OMAR agent. The OMAR agent must also react to events in the target system. As suggested earlier, the event types to which the agent will be expected to react will be communicated to OASYS at run initialization time. At run-time, notification of the event occurrences must be provided by OASYS. Whether the OMAR agent actually perceives and responds to the event will be a function of the multitasking capabilities of the agent and the workload in the immediate situation as mediated by the HPP model.

The OMAR agent must also be able to effect objects in its work place. These are the simple acts of setting and resetting switches, adjusting levers, and, more particularly in the ATC environment, speaking on the telephone or party-line radio and rearranging flight strips. The extent to which these actions might be animated in the OASYS simulation environment (e.g., presenting a hand moving across a panel to set a switch or the scanning pattern of the OMAR agent's eyes) is an open question.

In the future, the planned OASYS text-based network layer must be tailored to conform to the object-oriented development that is planned for OMAR. On the OMAR side of the network, interface objects representing the target system state will be created and updated, and generic functions will be developed to provide the capabilities outlined here. OMAR will include an object-oriented interface to the local target system model that will be used for OMAR development, and the code to link this object-oriented interface to the OASYS text-based network layer will be completed as required by plans to run OASYS.

PHASE 2: SOFTWARE TOOL DEVELOPMENT

Introduction

The software specification for the OMAR HPP Modeling Environment is described in the previous section. The implementation of the specification that took place during the 11 months of Phase 2 is described in this section. For purposes of discussion, software development undertaken during Phase 2 to implement the software specifications can be conveniently partitioned into the following areas:

- representation language development,
- graphic editors and browsers,
- user interface support layer,
- simulator development,
- simulation run-time environment,
- system integration,
- user documentation,

- test and evaluation, and
- preparation for Phase 3.

The major objectives and the nature of the work effort associated with each of these areas are described below. Developments within each area, with the exception of test and evaluation, are discussed chronologically to aid the reader in understanding the choices made among competing design and programming approaches. (Note: Because of the nature of the development cycle, there were months in which there was no activity for a given area.)

Key Areas of Software Development in Phase 2

Representation Language Development Synopsis

The OMAR representation languages are SFL for simulation object definitions, SCORE language for behaviors definition, and McRule, the rule language. SFL is a mature and stable language that required changes that were mostly related to adapting it for use with the new graphic editor. SCORE was the subject of a critical review directed at meeting the requirements laid out in the Phase 1 software specification for implementing the psychological framework. Significant changes were made, particularly in the representation of goals and plans, in the ability to specify behaviors related to task interruptions, and in the tools for resolving conflicts between procedures. McRule had to be adapted to run with SCORE in the OMAR environment.

Month 1: A preliminary design review of the proposed representation languages was undertaken. SFL was found to be sufficient to meet the initial object representation requirements for OMAR. The review of the SCORE language identified task contention resolution as an area that needed to be carefully reviewed. Limited experience with this area indicated that the basic design was good, but that there were subtle bugs present in the implementation. A need for a more declarative representation of predicates was also identified. Design proposals for incorporating a rule language were initiated.

Month 2: The work on representation focused on the rule language to be used for OMAR. A CLOS version of the primary candidate for a rule language was found to be available. The availability of the CLOS version of the language made it possible to move the scheduled starting date for work on the rule language to a later date.

In SCORE, goals were expressed using a *goal* clause within a procedure. The *plan* for a goal was defined as an *and/or* tree of subprocedures with temporal precedence expressed by an additional form. The need for a *defgoal* form was recognized, and it was suggested that the and/or tree for plan representation was not consistent with the *race/join* semantics of *defproc*.

Month 3: The work on representation focused on the procedure language, SCORE, to be used for OMAR. SCORE is built on top of LISP, and the design goal was to provide functionality that someone already comfortable with LISP could quickly learn. A related goal was to make it possible to learn LISP and SCORE concurrently. With this in mind SCORE forms for *when* and *unless* were added to the language, complementing the existing *cond* and *if* forms.

Work continued to devise a *defgoal* form. Formerly, goals were expressed with a *goal* form within a procedure definition. The *plan* for a goal within a procedure was expressed as an *and/or* tree with a separate expression detailing the temporal precedence among subgoals. Using *race* and *join* in procedures and an *and/or* tree with goals burdened the user with learning a syntax and semantics for each. To correct this problem, plans were laid to use *race/join* semantics for goals too. Toward this end, a SCORE *satisfy* form was developed and *defgoal* was built around *race/join/satisfy*.

Month 4: The work on representation continued to focus on the procedure language, SCORE, to be used for OMAR. The development of two additional language constructs was completed: *defprimitive* and *defpredicate*. The construct *defprimitve* now encapsulated pure LISP code as a SCORE procedure. The type of the procedure was inspectable, but in using this construct, the programmer would mark the enclosed code as a self-contained "black box." This feature was initially developed for the earlier Sproket simulator and proved very useful. *Defpredicate* marked the enclosed LISP code as a predicate and the procedure returned the value rather than the procedure itself. The logical operators *and*, *or*, and *not* were added to provide for these straightforward operations on predicates. Providing the *defpredicate* form satisfied the need for a more declarative identification of predicates.

Several conflict resolution strategies, as well as tools for crafting strategies using the basic SCORE language, were planned for development. Coding was completed for a strategy used frequently in the past. Here, a suspended task kills its subprocedures when suspended and reconstructs the environment to restart them again when resumed. This contrasts with the strategy existing at that time, which provides that subprocedures be simply suspended and resumed in place when appropriate.

Section 7.2.3 of Deutsch *et al.* (1993a) describes the development of a coding prototype to support the symbolic modeling of Edelman's Reentrant Nets. The prototype code was developed using existing SCORE language features, and it was indicated that a SCORE macro would be developed to simplify the coding of such networks. Development has been completed on the SCORE form, *with-multiple-signals*, with the functionality as outlined in the report. Since individual values can arrive across a span of time, a time-out feature may also be specified for each item. Through the use of the new macro, the original code shown in Figure 10 has been reduced to the code shown in Figure 11.

With-multiple-signals makes use of the *with-signal* form. The signals and their related behaviors were recoded to store them in a hash table to improve the performance of this SCORE feature which was anticipated to be heavily used.

Month 5: Coding was completed for a strategy in which a suspended task kills its subprocedures when suspended and reconstructs the environment to restart them again when resumed. Testing of this code continued during the month.

A large ATC scenario was ported to run in OMAR. As part of this effort, the SCORE code was updated to reflect new features of the language developed for OMAR. This test case was then used to validate the execution of the new SCORE language features.

Month 6: SCORE language features aimed at devising conflict-resolution tools continued under development as the basis for implementing the conflict-resolution strategies, and testing of the new features was commenced. The SCORE form *suspend-handler* was available to specify the behavior of a procedure that had a subprocedure that was suspended. The new form *on-suspend* was added to the language to enable a developer to specify SCORE forms to be executed just before a procedure itself is actually suspended. *On-succeed* and *on-fail* were also added to provide similar functionality for procedures that completed, either successfully or unsuccessfully.

The rule language was the last major language component to be included in this round of OMAR development. At this point, the rule language was running under Allegro Common LISP, and work on the interface from SCORE to the rule language accelerated.

```
(defproc WAIT-FOR-AIRCRAFT-HANDOFF (en route-atc-procedure)
    ()
 (with-slots (agent) self
  (loop-forever
   (let ((handoff-model nil)
         (optional-arg1 nil)
         (optional-arg2 nil))
    (join
     (sequentially
      (join
       (with-signal ((receiving-new-aircraft-event ?who ?model)
                     :test (eql ?who agent))
                    (setf handoff-model ?model)))
       (signal-event `(data-flow ,self)))
      (race
       (join
        (with-signal ((optional-arg ?optional-arg1))
                     (setf optional-arg1 ?optional-arg1))
        (with-signal ((optional-arg ?optional-arg2))
                     (setf optional-arg2 ?optional-arg2)))
        (asynch-wait `(data-flow ,self))))
     (process-aircraft-handoff :priority 20 :handoff-model handoff-model)))))
```

Figure 10
Data Flow Before the *with-multiple-signals* Form

```
(defproc WAIT-FOR-AIRCRAFT-HANDOFF (enroute-atc-procedure)
   ()
   (with-slots (agent) self
   (with-multiple-signals
            (((receiving-new-aircraft-event ?who ?model)
                     :test (eql ?who agent))
              ((optional-arg1 ?optional-arg1) :optional-p t)
              ((optional-arg2 ?optional-arg2) :optional-p t))
       (process-aircraft-handoff :priority 20 :handoff-model ?model))))
```

Figure 11
Data Flow Code Using *with-multiple-signals* Form

Month 7: Two fairly complex SFL features for concepts and roles are *rename* and *delete*. The former makes it possible to rename a concept or role; the latter makes it possible to remove a concept or role definition. The code for these features was implemented and the features were made available from the Concept Editor.

Work on the SCORE task conflict-resolution strategies continued. Testing of the implementation of the existing strategies was used in constructing a model demonstrating that they were working correctly. This effort verified the working of a number of the new SCORE language features. Team members continued to collaborate on OMAR extensions to the then-current conflict-resolution strategies.

Work on building the interface from SCORE to the rule language continued. The intent was to provide a calling protocol for invoking a rule packet that was as much like calling a procedure as possible. The details of the rule language features were tested to verify their operation.

Month 8: The rule language was the last major language component to be included in this round of OMAR development. A simple test case was used as the basis for debugging several features of the language that were not working correctly. The rule language also had individual features, each of which could be invoked using different labels (e.g., four rule types, each of which had two names). In the interest of simplifying the language from the user's perspective and providing a more concise user document, the duplications were removed.

Work on the SCORE task conflict-resolution strategies continued. The development of a high-level strategy of failing the subprocedures of an interrupted procedure and restarting the procedure upon resumption was begun.

Month 9: A simple test case continued to be used to debug the major features of the rule language. As indicated above, the rule language also had a few features that could be separately invoked using different labels. The last of these duplications were removed.

SFL functions to support the deletion of concepts and roles from a concept hierarchy, and the capability to change the name of a concept or role, were completed. Plans to make these capabilities available from the Concept Editor were outlined.

Work on the conflict-resolution strategy was completed this month. The SCORE around-method *fail-subs-on-suspend* will fail all the subprocedures of a suspended procedure. The suspended procedure will restart at the beginning when it is resumed. Similar behavior was available using the SCORE form *atomically*, but this was believed to have serious bugs. *Atomically* was carefully tested; the problems identified were corrected.

Month 10: The last step in integrating the rule language in Phase 2 was to provide the capability to execute rule packets as SCORE procedures. A SCORE *defrule-packet* form was developed that is analogous to the McRule *defrule-packet* form. The form simultaneously built a SCORE procedure and a McRule *rule-packet*. The rule-packet could then be invoked as a SCORE procedure with keyword-style arguments.

A SCORE procedure returns the procedure object itself as its value. This is consistent with the Actors system on which SCORE is modeled. Earlier, the SCORE macro, *return*, provided a means to access a LISP-style returned value. The *return* macro was replaced with the reader-macro *!r* to maintain consistency with the other reader macros that operated on procedures (e.g., *!t* and *!m*).

The effort completed OMAR Phase 2 work on representation language development.

Graphic Editors and Browsers Synopsis

A major portion of the OMAR software effort was devoted to the development of the Concept Editor and the Procedure Browser for SCORE. A new approach to the Concept Editor was taken, using separate windows for the graph of the hierarchy of SFL concepts and the table of slots associated with a concept. Likewise, the mode of user interaction with the system was simplified and made more consistent through the use of "pull-down" menus and a "selection" paradigm

The editor once again proved to be a very valuable aid to software development. The Procedure Browser provided a new capability to view a network of procedures across procedure-subprocedure calls and through activations driven by the SCORE *signal-event* operation. The implementation of each of these graphic software development tools was based on the User Interface Support Layer, assuring a consistent system look and feel.

Month 1: As the first change to the original software development schedule, software developers expressed a strong desire to have the Concept Editor up and running as soon as practical. To initiate that effort, the old Symbolics-based (flavors and pre-dynamic windows) Concept Editor was reviewed. The functionality of the graphical editor was very good. The code was, unfortunately, totally obsolete. A closely related *browser* for CLOS classes was also reviewed. The code for that browser was evaluated as the basis for the new Concept Editor.

Month 3: Work began on the development of graphing tools for the Concept Editor. The design of the table-editor display for use in the Concept Editor was completed and implemented. The table editor for a concept now provided for the editing of the slots of a concept, including the value restriction, number restrictions, default value, and documentation for each slot. The mode of interaction of the table-editors was modified to bring it in line with a more familiar "spreadsheet" paradigm.

Month 5: The first release of the Concept Editor was completed. With the aid of this editor, a concept is selected as the *focus* concept and the graph appropriate for that concept is presented. Similarly the slots for an "edited" concept are presented in a table. (The "focus concept" and the "edited concept" need not be the same.) Items presented in the graph and in the table are mouse-sensitive, providing additional detailed information on the object presented in a separate "Documentation" pane. Inheritance among concepts may be modified directly in the graph. Slot values are added or modified by editing table entries for the concept. In the initial release of the editor, the concept graph and the slot table were included in a single window.

The programming team completed a detailed review of the initial release of the Concept Editor. The human-computer interface aspects of the editor were carefully reviewed to ensure consistency across OMAR software development tools.

Implementation of the Procedure Browser, an important portion of the OMAR HPP model development environment, was begun this month. There were two basic tasks to bring the OMAR procedure browser on-line. The first involved developing the code that "walks" over the SCORE procedures, generating the graph structure that is representative of the SCORE code. This task began during this time period. The second task, not yet begun, involved developing the CLIM presentation of the procedure graphs and browser commands for selecting and moving over the presentation.

Month 6: The file-reading and -writing interface for Concept files was completed, along with additional pop-up editing dialogues for concept and role items. These included the *make-*

class? and *other-mixins* features of concept definitions and the documentation fields for both concepts and roles.

The Concept Editor has two distinct components, a graph-based presentation of concept and role hierarchies, and a spread-sheet-like table for detailed information related to a particular concept or role. Editing functions are available in each format. At this point in the development, as well as for historical reasons, the graph-based presentation of concepts and roles and the table-based presentations existed in a single CLIM frame. To gain greater flexibility in editing Concept objects and to bring the system in line with the more familiar desk-top-like metaphors of screen management, programmers planned to split this frame into two individual frames, providing separate frames for graph-based and table-based portions of the editor. Work was initiated to create the separate frames.

The OMAR development team worked with the Concept Editor, reviewing its functionality and evaluating interface ease-of-use. Consistency across the OMAR software development tools was also reviewed in preparation for the next set of refinements to the Concept Editor.

The development of the procedure browser was initiated at this point. Activity was begun to make the procedure browser, PIASTRE (Piastre Is A STRucture Editor) take the form of a true structure editor. In this revision, it is intended that flow of control will be depicted in the form of a graph, while subsections of graphs will be enclosed in bounding rectangles that depict, for example, the binding environment of a *let* or *with-slots* form. Each feature of the SCORE language will be accompanied by one of a small number of graphical presentation forms.

Month 7: The Concept Editor was now being used extensively in the browser mode and was starting to be used in the editing mode. The review of Concept Editor functionality continued. Improvements were made in file selection for reading and writing concept files. A number of additional minor improvements were made in response to the initial use of the editor. The process of reviewing Concept Editor operation and evaluating suggestions for the next release continued.

Work on the OMAR Procedure Browser progressed very rapidly. The set of SCORE language features currently spanned by the SCORE Procedure Browser now covered most of the language.

Month 8: The Concept Editor has two distinct components: (1) a graph-based presentation of concept and role hierarchies, and (2) a spread-sheet-like table for detailed information related to a particular concept or role. These now appeared in separate windows, allowing each to be individually adjusted in size to meet a particular user's needs and interests. Initial testing of the separation of the windows was very favorable. The Concept Editor was being used daily to aid in the development of the OMAR programming environment itself and to support the development of test scenarios used in validating the execution of OMAR programming tools. The final evaluation of Concept Editor capabilities and usage was now complete.

A major function of the OMAR Procedure Browser is to facilitate communication between programmers and nonprogrammers at the level of examining working code. This was accomplished by making the code available for viewing in graphical form. The SCORE Browser was extended to cover almost all of the features of the SCORE language. This development made use of the SCORE language examples file as the source of SCORE code for presentation by the SCORE Procedure Browser.

Month 9: In addition to the two windows provided for the Concept Editor, a third Concept window was developed to address such requirements as concept file management, preferences related to the editor, statistics having to do with concept hierarchy, general system output, and access to previously viewed graph or table layouts. The major component changes identified in

the design review were incorporated in the Concept Editor. A final review of the editor menu configuration and editor commands was completed in conjunction with the documentation effort.

Month 10: Work on the Procedure Browser focused on the display modes. An initial display that presents an individual procedure in graphic form was made available. In addition, two modes by which to view the network of procedures were defined: (1) a traditional view of the calling sequences of procedures, and (2) a view showing the connections among procedures driven by the *signal-event* SCORE form. Given a procedure name, one may now obtain a view of all procedures that the procedure calls or a view of all procedures called by the procedure. Similarly, given a procedure name, one may obtain a view of all procedures waiting for the signals that the selected procedure generates, or a view of all procedures providing the signals for which the selected procedure waits. Development of these display modes was begun.

Month 11: The display modes for the Procedure Browser outlined above were completed. These new network views of procedures were expected to be of significant value in developing complex HPP models.

This effort completed the OMAR Phase 2 work on the Concept Editor and Procedure Browser for the SCORE language.

User Interface Support Layer Synopsis

Completed very early in the development cycle, the user interface support layer was essential to assuring a consistent look and feel across the OMAR Concept Editor, Procedure Browser, and run-time system components. As the base layer for each of these system software components, the user interface support layer was a major productivity aid in their development.

Month 1: A review of the functionality and coding of existing graphical browsers and editors was undertaken. The review covered the previous version of the Concept Editor, a browser for CLOS classes, and the graphical editor for an earlier procedural language.

Month 2: Menu styles and buttons were selected to be included in the user interface support layer. These menus and buttons were designed to support the familiar "pull-down" and selection-centered modes of user interaction. Coding of these display objects was initiated.

Month 3: Work continued on generating displays for various menus and buttons to be incorporated into the user interface support layer. The following were under development:

- shadowing ("three-dimensional") buttons and table-cells (with inverse-shadowing highlighting),
- synchronized multipane scrolling,
- autodescription of cells and buttons pointed at (by the mouse),
- centering/layout algorithms for custom-button menus (including automatic resize), and
- customizable pull-down menus.

Month 4: The user interface support layer was completed and then used in the development of the Concept Editor, the Procedure Browser, and the run-time windows. It became an essential building block in each of the graphical interfaces to OMAR and provided a means to ensure consistency in operation across the many tools that make up OMAR.

Simulator Development Synopsis

Prior to OMAR development, SCORE simulator development, driven by slightly different application requirements, had followed two divergent paths. The first step taken toward

resolving the differences was an evaluation of the two development threads. This led to the selection of the version that supported recording and the most recent language extensions. This simulator was used as the basis for developing a version to run on the Sun. The new version of the simulator for OMAR was then developed with a functional interface in preparation for use with the OMAR run-time environment.

The anticipated requirement to run an OMAR HPP model within the OASYS simulation environment was factored into the design of the OMAR simulator. Developers from the OMAR and OASYS groups met regularly to address this design requirement and to coordinate the development of their respective simulators.

Month 1: Each development thread was carefully evaluated; the more broadly based of them was selected as the basis for OMAR development. The primary advantages of the selected thread were the inclusion of the recorder (on-line data collection and post-run analysis tools) and language extensions (e.g., *with-signal* and *one-of*). The other thread provided significant work in the important area of task interruption and resumption, which was also incorporated in OMAR simulator development.

Month 2: The SCORE simulator was made operational for OMAR on the Sun. It formed the basis for the development of the OMAR simulator. Test cases were developed to review task interruption routines available in the other simulator development thread.

Month 3: In preparation for the development of the new run-time environment, the simulator was separated from the existing user interface and was made available through a simple set of function calls. The new function calls provided for the instantiation of a simulator; the selection and creation of a scenario; and the running, pausing, and stepping of the simulation run.

The simulator was used extensively to review task suspension routines.

Month 4: Members of the OMAR and OASYS development teams met for a series of preliminary reviews of OASYS-OMAR interface issues.

Month 5: System development reached the point where a large test case was needed. An en-route ATC scenario used in the third phase of RDT&E Delivery Order #1 (Deutsch *et al.*, 1993a) was selected. The scenario, based on earlier work done for National Aeronautical and Space Agency, was ported to run in OMAR, where it was used as a run-time test case for the simulator. As indicated above, the scenario was also for SCORE language development.

Discussions concerning the interface between OMAR and OASYS continued with the OASYS group in order to ensure compatibility among the uses of "signals" to be employed during an OMAR-OASYS simulation run.

Month 6: Further exercise of the recently ported en-route ATC scenario for simulator testing continued.

Month 8: At this point, the simulation code was considered to be solid. Further revisions were not anticipated except in connection with new or revised SCORE language features.

Month 9: The en-route ATC scenario and the example scenarios from the documentation were used for testing. The simulator code was considered to be adequate at this time and was expected to be impacted only by new or revised SCORE language features as they were developed.

Work on the OMAR simulator for Phase 2 was now complete.

Simulation Run-Time Environment Synopsis

One of the implementation objectives was to have a working test-bed up and running very early in the development cycle. To meet this need, the existing SCORE interface was updated to CLIM 2 and made available as a prototype for OMAR. With a working test-bed in place, it was then possible to step back and prepare for the more orderly development of the run-time environment. The run-time environment was based on the user interface support layer, with menu-based links to the software development tool windows.

Month 1: The interface to the earlier SCORE simulator was reviewed in preparation for developing the run-time interface for OMAR.

Month 2: A simple CLIM 2 run-time interface was developed for the simulator then running on the Sun. Consistent with the rapid prototyping approach, this interface provided access to OMAR scenarios, run-time control of the simulator, and trace facilities suitable for debugging. This interface and the simulator formed the core around which OMAR software development progressed.

Month 3: The CLIM 2 run-time interface for the simulator was made operational. This interface made use of the new function calls provided by the simulator for scenario selection and scenario execution. The basic run-time test-bed was now in place for OMAR language and scenario development.

Month 6: The OMAR development team began a review of the run-time requirements for OMAR as the final step before starting work on the run-time interface.

Month 7: The OASYS Mirage II Graphic User Interface (GUI) builder was selected for building windows for analysis and animation because its run-time capabilities enabled the panels to be displayed at any workstation on the network.

Month 8: By the end of the month, the OMAR development team had completed its initial review of the run-time requirements for OMAR. The team now completed a prototype for the run-time environment that was driven by a test scenario developed for the purpose. The run-time environment featured a Mirage panel developed using the OASYS Mirage GUI editor. The panel showed various effective states of an individual agent in the test scenario. The run-time interface allowed users to select a scenario, control its execution, and observe details of the internal state of a specific agent. The prototype run-time environment included a LISP window to be used primarily to support debugging during model development.

The underlying core Mirage code was shared between OMAR and OASYS with specific "gadget-types" available to each system. The specific GUI created for the OMAR run-time environment was a test application of Mirage and an important step in validating the Mirage GUI and user-oriented GUI-building concepts.

Month 9: A prototype interface was developed between the OMAR simulator and Mirage so that Mirage display panels could be used to present OMAR HPP model data and parameters. This was the last major run-time piece to be developed in Phase 2.

Month 10: The OASYS Mirage II interface builder was upgraded to Mirage III, and the OMAR interface was revised for use with the new release. Mirage display panels were used to present OMAR HPP model data and parameters.

At this point, the run-time environment included a CLIM window for selecting and managing the execution of a scenario and presenting basic scenario-generated data, a timeline display of SCORE events related to procedure execution, and the new capability to link Mirage panels to the OMAR simulation. Now, windows were revised to make use of the user interface

support layer tools. With this change in place, the run-time environment had a look and feel consistent with the graphic editors and browsers of the OMAR system.

System Integration Synopsis

OMAR consists of a number of major subsystems: a set of representation languages, graphical browsing and editing tools to support HPP model development, and a simulator with a run-time environment. Each of these, in turn, is composed of a series of subsystems. This level of complexity required special attention to system integration. It is important that the software subsystems work together to effectively support the development of HPP models. Furthermore, OMAR had to have a consistent look and feel at the model developer's level, and it had to provide smooth access across individual subsystems.

The major contributor to system integration supporting a consistent look and feel was the user interface support layer. Individual subsystems, sometimes adapted from existing code, were developed with prototype interfaces and quickly integrated with previously integrated subsystems. The development of each subsystem followed this process. Hence, it was possible to verify that major HPP model development components were working together long before user interface issues were addressed. An iterative approach to integration at the user-interface level then followed. This process led to the clustering of subsystem windows that OMAR now uses.

Month 8: As the first step in system integration, an OMAR system window was developed to provide access to the multiple Concept Editor windows and the SCORE Procedure Browser. The system window would later be extended to include the Mirage and LISP run-time windows. Movement between windows would then be available through the overview window and directly from window to window. The cluster of Concept Editor windows included an "overview" window to accommodate high-level Concept Editor features not appropriate to graph or table windows.

Month 9: Work on system integration continued. Movement among windows was enabled through the system window and directly from window to window, as outlined above.

Month 10: Using the user interface support layer, the system integration of the run-time windows was initiated to bring their appearance and operation into conformance with the Concept Editor and Procedure Browser. The user interface support layer, already in use for the editors and browsers was now being used in the run-time windows.

Month 11: The integration of the Concept Editor window cluster, the Procedure Browser, and the LISP and Mirage run-time windows with the top-level OMAR window was completed. Movement among windows was enabled through the OMAR system window and directly from window to window, as appropriate. Phase 2 System Integration of the run-time windows was now complete.

Phase 3 Month 5: The Phase 3 effort on the ATC scenario provided extensive experience in HPP model development and the use of OMAR. Experience with the cluster of three windows that make up the Concept Editor was very positive, hence this clustering was extended to the run-time environment. Simulation run-time control, scenario animation (i.e., Kate and the agent dialogue panes), and the trace window made up the new cluster. The top-level OMAR system window now includes icon-like buttons for the Concept Editor window cluster, the Procedure Browser, and the run-time window cluster.

User Documentation Synopsis

Despite efforts to reduce conceptual and programming complexity, it is likely that some OMAR programming conventions and tools are difficult for new users to learn and employ in the formulation of human performance models. This likelihood argued for a sustained, iterative effort that would achieve consistency in functionality and terminology, and an orderly progression in the difficulty levels of examples to be included for illustration. Accordingly, a person skilled in the preparation of user documentation was asked to join the project at the conclusion of Phase 1. This person remained a key member of the development team until the end of the project, working closely with the technical manager and programmers both as an advocate for the OMAR user and as a technical writer. The work conducted in this area substantially complemented efforts to test and evaluate software as it was produced.

Month 3: To initiate the documentation activity, a series of meetings were held with OMAR developers to provide the documentation expert with a detailed look at OMAR components and capabilities. These sessions also functioned as design reviews for the OMAR languages. The final form that *defgoal* and other functions were to take was determined by these discussions.

Preparatory documentation efforts focused on four areas:

- understand how the conflict-resolution protocol worked to mediate task contention,
- learn about the Rules Language, the last of the languages to be documented,
- make a checklist of all the SCORE Language forms to ensure completeness — most of the additions were commonly used LISP forms that were implemented as SCORE forms, and
- meet with the OMAR team to plan and discuss an approach to an "Examples" section.

Month 9: Developers supported the documentation effort and assisted in testing OMAR language features. The SCORE language feature tests were assembled in a file of SCORE test procedures and were also used to support test and evaluation. Work was initiated on three example scenarios of increasing levels of complexity which were designed to demonstrate the use of the OMAR languages. The scenarios involved party-line radio communications between an ATC and an aircraft. Development of a fourth example scenario was also initiated using a simple holon-based model for the ATC.

Month 10: The documentation efforts on the SCORE language features were continued. The file of SCORE test procedures was used as the basis for this effort. Additional test cases were developed as needed to verify language features. The four scenario examples to be provided with the documentation were completed.

Month 11: The draft of the OMAR User/Programmer Manual was completed on schedule. The documentation effort played a significant role in assuring the consistency of the overall look and feel of the OMAR system, design review of the OMAR languages, and system test and evaluation.

Test and Evaluation

Objectives

Two levels of testing were deemed essential for the OMAR system: detailed language tests and system-level tests. These test suites are depicted in the lower portion of Figure 12. The detailed language tests were designed to verify the operation of the features of each of the OMAR languages: SFL, as the definition language for simulation objects; SCORE, as the

definition language for agent behaviors; and the rule language. System-level tests were more broadly based. At the highest level, they were designed to assess the mobility provided for moving among the individual subsystems that make up OMAR. They were also necessary to ensure a consistent look and feel across the system components. The most important portion of the system-level testing was directed at verifying that the mix of software tools provided the essential flow necessary for efficient software development.

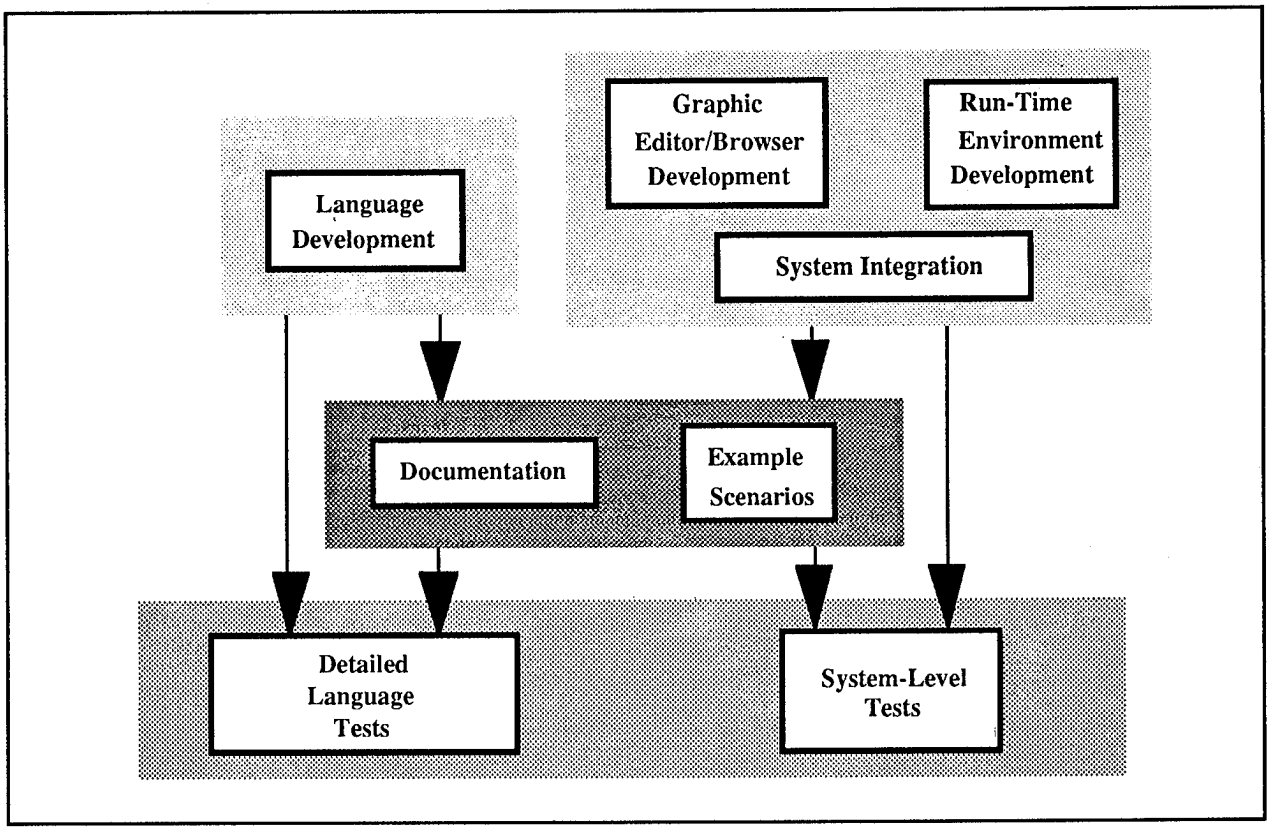


Figure 12
Test and Evaluation

Approach

Initial testing is frequently the responsibility of individual contributors to a software effort, with wider access to software component behaviors available only very late in the development cycle. Making changes is often expensive and frequently impossible because of end-of-project constraints. The scheduling of the documentation effort was used to attack this problem: it was initiated very early in the software design and development process. The documentation for the OMAR languages was underway while the design process for the languages was still in process. The documentation for system operation often drove the specification for system integration and system operation. System documentation was used to force consistency of look and feel across system components. It was not uncommon for developers to use drafts of documentation as system specifications. The middle row of Figure 12 is intended to depict the role played by the documentation effort in supporting the test and evaluation process. The sample scenarios used as examples in the documentation were used in the testing procedures.

Outcome

Early in the evaluation process, primary responsibility for testing resided with individual developers because, at that point, they were best equipped to verify individual component operation. The early start on the documentation effort also meant that developers were "peppered" with questions, driving a very thorough, detailed evaluation at a time when change was easy to accommodate. An early start on documentation uncovered numerous details in the languages where there were opportunities to provide greater consistency. Detailed language testing became a very thorough, collaborative effort.

The process for system-level testing was built on the experience gained in the detailed component-level testing. Documentation staff assisted in system-level testing as an integral part of the documentation process. The four graded scenarios developed as part of the documentation and the extended ATC scenario became the basis for the system-level testing. As shown on the right side of Figure 12, developers were once again supported by documentation staff in test and evaluation. Collaboration in the design process with the early involvement of the documentation staff enabled a broadly based and effective test and evaluation process.

Preparation for Phase 3

The focus of the Phase 3 effort was the HPP model of the ATC. Initial design review meetings were held with the objective of identifying issues that were specific to the ATC domain to be modeled.

The developers had their first meetings on software design for the ATC scenario. Work on the representation and display of the three-dimensional ATC workspace began, along with development of agents to be modeled: en-route ATCs, flight crews, and their aircraft.

PHASE 3: ATC HPP MODEL DEVELOPMENT AND SOFTWARE TOOL REFINEMENT

Introduction

Activities undertaken during the final phase of the project covered a broad range. Modules developed in Phase 2 were refined and tested, and details of the ATC handoff scenario and the (human) controller behaviors to be included in the simulation were decided upon and programmed. Early in Phase 3, the potential value of a real-time display of an HPP model's behavioral output was clearly identified. It was recognized as an essential aid for visualizing the complex behavioral sequences generated by an HPP model necessary for troubleshooting code. An effort commenced to develop a mannequin whose head, eyes, and hands responded to events in the simulation. Finally, Revision 1.0 of the OMAR documentation was completed, and plans were made for the training of AL/HRG personnel to be conducted during the first quarter of 1995.

ATC Human Performance Process Model Development

This section follows the format established in the previous section, partitioning development activities into a set of distinct areas and presenting chronologies of major design considerations within each of these areas. Four areas are considered:

- agent behaviors,
- agent performance evaluation,

- agent animation (Kate), and
- the three-dimensional ATC workplace and the airspace.

The principal agents for which HPP models were developed were the en-route ATCs and the flight crews of the aircraft. Each two-person flight crew included a captain and a first officer. The principal event of the scenario was the handoff of an aircraft from one ATC to the neighboring ATC.

Month 1: The verbal communication skills of the HPP model were the first areas to be addressed. Work was initiated on party-line radio, telephone, and in-person conversations, with reasonable handling of interruptions (e.g., an in-person flight deck conversation interrupted by a radio communication from an ATC).

Month 2: Work was initiated on the procedures for the handoff of an aircraft from one ATC to another. The complete set of procedures required: (1) a telephone call from the ATC controlling the aircraft to the ATC that was to receive the aircraft, (2) a radio message from the current ATC to the aircraft informing the flight crew of the handoff and the radio frequency for the new ATC, and (3) the contacting of the new ATC by the aircraft following the resetting of the radio frequency. Work was also initiated on several of the supporting procedures for hand and eye movements and coordination. Work on the more cognitive aspects of the handoff and the management of the airspace were just getting underway.

Kate was used for the first time for the animation of hand and eye motions in the three-dimensional workspace. Previously, evaluation of these actions was possible only via careful scrutiny of statements in the simulation trace.

Month 3: Work continued on the procedures for the handoff of an aircraft from one ATC to another. An area that also received attention was the set of expectations associated with the arrival of a new aircraft in the airspace. In real operations, the phone call from the neighboring ATC and the initial contact from the new aircraft are anticipated and checked off as they are processed. In a similar manner, the behavior of an aircraft following a descent directive is monitored. The management of expectations involve the interplay of the proactive and reactive tasks of the agent. The reactive tasks of responding to telephone calls and radio communications *satisfy* the proactive goals of managing new aircraft entering the airspace.

Month 5: Early in the ATC scenario, the Fort Worth controller directs DAL100 to "descend and maintain 33,000" and UAL10 to "maintain 35,000." The expectations for the actual altitude behaviors of the aircraft were added to the ATC's behaviors and integrated with a background scan pattern of the radar screen. The ATC now maintains tasks associated with managing each aircraft in the airspace. Associated with each of these monitoring tasks is a low priority task to scan the aircraft with a loosely specified frequency. Hence, there is now a default, "background-launched" scan as well as the higher-priority scan associated with expectations maintained for a subset of the aircraft. The expectation-based scan subsumes the background scan. Furthermore, an expectation-based scan of one aircraft will trigger expectation-based scans and background-based scans of related aircraft in the sector. This is a first step toward the grouping of aircraft that ATCs are known to use to manage their airspace.

Agent Performance Evaluation

Month 1, 2: The OMAR development team held a series of meetings to review options available for agent performance evaluation. For many experiments, specific performance evaluation routines were found to be necessary. The presentation of an agent's performance as a task timeline was selected as the most useful generic tool for examining agent performance.

Month 3: Task execution and task interruption were viewed as central to agent performance. The timeline display of the tasks executed by an agent was modeled after the available agent-event timeline. In this portrayal, a bar in the timeline represented the period for which a task is active. The timeline bar was annotated with the name, start time, end time, and priority of the task as well as the name of the task that initiated it. The nesting of the tasks in the timeline followed the nesting in the procedure-calling sequence.

Month 4: Through the use of the OMAR event-recording capability, a very complete record of agent performance could now be maintained during a simulation run. It is these data that now form the basis for displays such as the event display and the new task timeline display described above. These data can also be made available in a form that takes advantage of the numerous data analysis and data graphing packages that are generally available on the Macintosh and personal computers. The short-term goal is simply to demonstrate the viability of this approach. For this purpose, a sample data set was transferred from the Sun to the Macintosh, and an Excel display was created.

The agent timeline display was refined to include the presentation of task suspension and resumption. The layout of the display was changed to better accommodate the larger number of tasks that an agent typically has active.

Month 5: Task execution and task interruption are central to agent performance. The timeline display of agent-task execution is modeled after the current agent-event timeline. A bar in the timeline represents the period for which a task is active, and its annotation was now reduced to the name and priority of the task. The procedure type (e.g., goal or procedure), the start- and end-time, and the name of the procedure that initiated it now appear in a mouse-highlighting window as the mouse is moved over the timeline. The timeline was also enhanced to show all procedures that interrupt a given procedure. The mouse-highlighting window provides the name, priority, and time frame of the interrupting procedure.

Agent Animation (Kate)

Month 1: Development of "Kate," a simple three-dimensional stick-figure representation of the human body, began. When completed, OMAR users will be able to monitor the motor behaviors of an OMAR HPP agent as they are played out by Kate. The basic set of appendages will include fingers, hands, arms, legs, feet, and eyes. It is planned that each joint in Kate's body will become a coordinate system that can be addressed and controlled separately. The Kate system will become one specific application of the coordinate-tree system described below in the subsection.

Month 2: Kate was endowed with initial versions of hand and eye movements. At that time, the motions were preplanned between selected workplace objects and body positions. Kate's behavior was then tested using the ATC scenario.

Month 4: In the initial release of Kate, all body motions were instantaneous. Kate was now revised so that her movement of an effector (e.g., the left hand) between locations has an elapsed time. The duration for hand movements is computed using Fitts' Law (Adams, 1989), and the size of the time step used in the animation may be selected as appropriate.

Month 5: Up to this point, there had only been the capability to animate one of the agents of a scenario using Kate. The capability to have multiple Kates was now being added so that it would be possible to show, for instance, a captain and first officer on the flight deck. Object-and event-naming conventions were extended to accommodate the use of multiple Kates. This completed the effort initiated in the prior month to model multiple workplaces.

The Three-Dimensional ATC Workplace and the Airspace

Month 1: Development was initiated on the system for specifying relative three-dimensional positions of a set of scenario objects. The basis of this is a "world-tree" of coordinate systems. From a base, fiducial origin, a tree of transformable coordinate systems is defined. As an example, a workstation/operator pair can be defined with a central, common-coordinate-system basis established for the pair. Relative to this base coordinate system would be base coordinate systems for each workstation and the human operator. The base coordinate system for the human operator would, in turn, have base coordinate systems for the upper and lower bodies. The upper body would, in turn, branch into base coordinate systems for the head, the left shoulder, and the right shoulder. This sequence of subdivisions would continue until coordinate systems for the entire body were defined(e.g., for each of the eyes, hands, and feet).

As part of its definition, each coordinate system can be both offset from and rotated relative to its parents. Each coordinate system contains methods for translating individual (x,y,z) points back into the root coordinate system. Once this base transformation is established, it becomes possible to determine relationships among various components of the system, such as, relative distance between the hand and a button, angular orientation to which an eye must be turned in order to look at an icon on a radar screen, and the maximal angle through which an elbow must be extended in order to reach a given switch.

The work on Kate outlined above and the development of the ATC workplace were based on this three-dimensional representation system.

The basic objects to support agent communication behaviors were developed: party-line radios, telephones, and in-person conversations.

The aircraft and the airspace in which they operate were also made operational. Much of the capability for this was carried forward from the ATC scenario that had been used as a test scenario for Phase 2. The simulation airspace currently under development, "the Fort Worth-Houston area", included the Dallas-Fort Worth Airport, Very High Frequency Omnidirectional Range/Tactical Air Navigation (VORTAC) aides, reporting points, sector boundaries, and airways.

Month 2: Development continued on the system for specifying relative three-dimensional positions of a set of objects. An initial three-dimensional ATC workplace model was released. It included a radar screen, keyboard, telephone, and the Kate representation of the ATC. Within the three-dimensional workplace view, the aircraft icons on the radar screen were static objects.

Month 3: A set of macros was developed to specify the locations of objects in the three-dimensional workspace. These macros were refined to enable each object position to be specified relative to its parent object. Planar objects could be positioned in two-dimensional space, then translated and rotated into their three-dimensional positions in the workspace. Macros supporting workplace layout now became much easier to use.

A simplified version of operator motion was established. In the initial version, the operator has three movable "effectors": the left hand, the right hand, and the eyes. It is assumed that the motion of each effector is independent of and does not affect the others (e.g., reaching with the left hand does not cause the trunk to move, consequently altering the base position of the eyes).

Motion for the eyes is specified by requesting that the eyes look at a particular object in the workspace (e.g., the telephone). The necessary angular orientation of the eyes (and head) can be determined and the eyes (and head) turned to that orientation.

Because they are coupled to the motion of the arms, the motion of each hand is somewhat more complicated. In general, placing a hand at a given location involves determining the orientation of each of several angles for each of the affected joints (shoulder, elbow, and wrist). The determination of these angles is known as the "Inverse Kinematics Problem."

In its initial version, the system takes a very simplified approach to this problem. In short, the hand is moved by interpolating the joints of the arm through angle space. In more detail: first, a set of target-locations is defined for each hand. For each target-location, the complete set of orientation angles for each joint necessary to move that hand to that target-location is specified. When a request to move a hand to a new target-location is received, the current location of the affected orientation angles is noted. The motion of the hand is then accomplished in a number of steps. On each of those steps, each affected orientation angle is incremented an appropriate amount.

The capability to manage the display of multiple agents and multiple workplaces was based on extensions to the coordinate system hierarchy described above. A position-naming scheme was now put in place for multiple agents and workplaces (e.g., rather than the telephone, there was a telephone at each ATC workplace).

ATC Scenario

ATC Scenario Overview

There are two en-route ATCs in the scenario: one at Fort Worth and one at Houston. The Fort Worth ATC is the one shown as Kate in the display. The trace material that is printed also relates to the Fort Worth ATC. There are two aircraft, DAL100 and UAL10, in the Fort Worth ATC's airspace and a single aircraft, AAL1, in the neighboring Houston ATC's airspace. The main event of the scenario is the handoff of AAL1 by the Houston ATC to the Fort Worth ATC. There are telephone links between neighboring ATCs. Communication with the aircraft is via party-line radio.

Each aircraft has a two-person crew: a captain and a first officer. The captain maintains an in-person conversation with the first officer to manage operation of the aircraft. On each aircraft, the first officer is responsible for maintaining party-line radio communication with the ATC managing the airspace. The captain monitors the radio conversation.

There is a little background activity on the part of the Fort Worth ATC. When there is nothing else going on, the ATC's eyes are directed to the center of the radar screen and, because of a general inability to sit still for long periods of time, the ATC will move her dominant hand between her lap and the keyboard. The telephone that the ATC uses is located adjacent to the keyboard.

Events in the ATC Scenario

T=12.10 DAL100 Descend and Maintain FL330

The Fort Worth ATC directs DAL100 to descend to 33,000 feet and maintain that flight level. The ATC's eyes are focused on the icon for DAL100 on the radar screen. DAL100's first officer completes the transaction by acknowledging the directive. DAL100's captain sets the aircraft's Mode Control Panel altitude level to 33,000 to effect the altitude change.

The radio message from the Fort Worth ATC to DAL100 has actually interrupted a conversation on the flight deck, which the captain resumes when the first officer has completed the acknowledgment of the ATC radio communication.

T=52.90 UAL10 Maintain FL350

The Fort Worth ATC directs UAL10 to maintain 33,000 feet. The ATC's eyes are focused on the icon for UAL10 during the transaction. This is just a simpler version of the previous transaction.

T=80.10 Flashing Icon Initiates the Handoff of AAL1

The icon for AAL1 is the one on the lower left. As simulated, it has been there all the time; however, from the Fort Worth ATC's perspective, it has just appeared and is flashing, indicating that an aircraft is approaching her airspace. A number of activities follow from this event.

At this point, the Fort Worth ATC has two new expectations: she will receive a telephone call from the Houston ATC to set up the handoff, and she will subsequently receive a radio contact from the aircraft when it enters her airspace. These expectations are noted in the trace output.

In preparation for the handoff, the ATC now checks for conflicts between the new aircraft and aircraft already under her control. Her skill level is intermediate; hence she does a pairwise check. She looks at the new aircraft, moves her eyes to an aircraft under her control and then back to the new aircraft, repeating this pattern for each of the aircraft under her control. If she were an expert, she would look at the new aircraft and then simply cycle through each of the aircraft under her control, finally returning to the new aircraft. As an expert, she would dwell for less time on each aircraft.

T=128.20 Receive Handoff Telephone Call from the Houston ATC

The Fort Worth ATC receives the telephone call from the Houston ATC. She looks at and picks up the telephone, saying hello as her eyes return to the radar screen. The Houston ATC announces the handoff of AAL1 and the Fort Worth ATC looks at the icon for AAL1 on the screen. She accepts the handoff, completing the transaction and satisfying the expectation of the telephone call that she has maintained. She looks at the telephone as she hangs it up, and her eyes return to the radar screen.

T=167.70 Receive Radio Contact from AAL1

Previous to this event, AAL1 has received a radio directive from the Houston ATC announcing the handoff and telling it to contact the Fort Worth ATC on radio frequency 133.775. They have each set their radios to the new frequency.

AAL1's first officer now makes radio contact with the Fort Worth ATC, announcing the presence of AAL1 in the Fort Worth airspace. The Fort Worth ATC looks at the icon for the aircraft and acknowledges the contact. The expectation of the radio contact by AAL1 has now been satisfied, as noted in the trace. She returns her eyes to the center of the radar screen. The scenario is now compete.

FUTURE RESEARCH

In the OMAR project, a three-step approach to the HPP model-building process was pursued.

1. A psychological framework was developed that focused on the multitasking capabilities of human operators as the primary contributor to their successful performance on complex tasks.

2. Next, a computational framework was identified that was to form the basis for representing and executing these behaviors, and a suite of software development tools was created to facilitate model-building, model execution, and model debugging.

3. Finally, ATC scenario was developed that included HPP models of en-route ATCs and aircraft flight crews as the principal agents of the simulation.

This was a large undertaking that necessarily entailed a series of compromises. One source of suggestions for the future direction for this research is to revisit areas that it was simply not possible to address in adequate detail in the initial OMAR effort. A second source is the experience gained in developing the ATC scenario. The software development environment was very supportive, yet desirable new capabilities can be identified.

The recommendations identified here fall into two categories: those that are primarily enhancements to the HHP model development environment and those that are directed at enriching model behaviors for which it will be necessary to revisit and extend the psychological framework.

Psychological Framework Extensions

Teamwork, Attention, and Memory

The development of the ATC scenario was the first exercise in using OMAR to formulate an HPP model of operator performance. The active agents included not only the en-route controllers but also the two-person flight crews of each aircraft. One focus of the modeling effort was the verbal communication used by the ATCs to manage the airspace. This involved telephone conversations with neighboring controllers and party-line radio communication with aircraft in the sector. As described, these were application-level models in which ATCs were concerned with managing the airspace, and flight crews were concerned with the operation and safety of the aircraft.

Underlying the application level of the models was the psychological level of the models. At this level, the models were concerned with the multitasking capabilities that humans rely on to manage the complex mix of achieving their goals and responding to impinging world events. Multiple tasks may demand attention driven by auditory or visual inputs placing varying demands on short-term memory. In developing the ATC scenario, each of these areas was addressed to a reasonable degree that was consistent with the resources available. The scenario agents, using a mix of basic psychological-level and application-level capabilities, exhibited reasonable behaviors in addressing the tasks with which they were confronted.

An underlying theme in the management of the commercial airspace is teamwork. It is essential for overall control of the airspace by the ATCs and the management of each aircraft by its flight crew. Teamwork is similarly essential to any number of military operations, particularly command and control; its importance is such that teamwork is recommended for use as the application-level driver for future HPP model development. Within this framework,

agents must respond to external demands on their resources, but they may also use others as sources of assistance. Judith Orasanu (in press) has looked carefully at teamwork in flight-crew performance, providing a good entry point to this literature. Meeting these modeling demands will also require attention to the psychological level of the models — particularly attention and memory. The work of Michael Posner (Posner & Peterson, 1990) and Alan Allport (1989, 1993) are good starting points for looking at attention. Douglas Hintzman's (Hintzman & Harty, 1990) multiple-track memory model focuses on context and its role in short-term memory. His work is closely related to Gordon Logan's (1988, 1992) reference research on automaticity that was an important focus of the psychological framework described here. Extending the research effort in modeling at the psychological level is important in that it is generic and, hence, applicable to many modeling applications.

Human Error and Novice through Expert Performance

In the psychological framework and the ATC scenario, multitask behaviors were reviewed and carefully modeled. Issues of attention and, to a lesser extent, short-term memory were examined and explored in the scenario. On one hand, the human ability to simultaneously address multiple tasks makes the operation of complex equipment possible, while on the other hand, it is a source of human error. A more thorough study of these sources of human error is appropriate. In particular, the sources of error will vary with workload and will change as an operator's skills improve. An interesting recent paper by Riccio *et al.* (1994) examines "loss of memory for the characteristics of stimuli" in initiating inappropriate actions. This is an area that may be explored at the level of the psychological framework and grounded in extensions of the present ATC scenario.

Learning

In recent years, there has been a great deal of research in the areas of both human and machine learning. Learning has been studied from symbolic and connectionist perspectives and, more recently, with the aid of hybrid approaches. This is a very difficult subject area and one that has not been addressed in the development of OMAR. However, the OMAR architecture was designed from the beginning to be an open architecture and is, therefore, well suited as a medium in which to explore human learning.

The emergence of automaticity through experience is one area that might be explored. Initially, HPP model behaviors would be driven by OMAR procedures with decision making and object recognition represented with the aid of rule sets. This is the symbolic material that might be taught through classroom learning in preparing students for a real-world task. The response would be the result of a slow deliberative process. An artificial neural network (ANN) operating in parallel (Logan, 1988) with the procedures and rule sets would use the procedure-driven outcomes as training data. When the ANNs, representing automatic behaviors, recognize the stimuli, they would control a more rapid HPP response.

OMAR System Extensions

Enhanced Model Development Support

Sometimes rather simple software debugging tools can lead to better insight into software behavior and significant improvements in productivity. As an example, the rule-based language in use in OMAR maintains a trace of rules that have fired. Better access to this trace data would be helpful. The ability to re-compile a SCORE procedure and continue scenario execution from that point would be similarly helpful.

As the number of procedures becomes large, software behavior is more difficult to track. OMAR now includes support to track procedure conflict resolution and event-based interactions

between procedures; these areas will require additional attention as HPP model complexity increases. Scaling issues will also impact holon-based models, requiring similar support. Additional improvements specific to the Concept Editor and the Procedure Browser are identified below.

The Concept Editor

The initial version of the Concept Editor, available relatively early in Phase 2, went through a testing phase and was used extensively in the development of the ATC scenario. As it was used, it went through a number of revision cycles, then remained unchanged late in the project as more urgent requirements were met. There is a small backlog of changes, mostly minor, that are recommended. Representative of these additions are the tracking of changed SFL files, a reminder that altered file data has not been saved before exiting, and more immediate access to editing items now only available through the *edit internal* option of the Concept Editor Table Window.

The Procedure Browser

The initial Procedure Browser view of individual procedures was available for the first time midway through Phase 2. Very shortly thereafter, the network views of procedures came on-line and, within surprisingly few iterations, the Procedure Browser achieved its final form. As with the Concept Editor, there is a small backlog of changes that are recommended. Representative of these changes are a mouse action providing immediate access to the textual code for a procedure and a network view combining the current procedure/subprocedure and *signal-event/with-signal* network views.

Dynamic Task Priorities and Task Execution Times

Variations in workload are accommodated by human operators through adjustments in their multitask behaviors. As workload increases, some task will be deliberately shed, simply forgotten, or reduced in priority. The speed of execution of individual tasks can also be expected to vary with workload. Task duration is modeled using the SCORE *sleep* form, and task priority is the basis for contention between tasks bidding to execute. Tools to manage task execution time and to dynamically adjust task priorities are in place, but these should be reviewed. Areas suggested for attention are adjusting task duration during task execution, adjusting the priority of a suspended task upward, and adjusting the priority of an active task downward.

Agent Performance Analysis

Event recording is a basic OMAR capability that was designed to be easily extended to meet the needs of a particular experiment or scenario. In the ATC scenario developed during Phase 3, extensions included expectation events that recorded the occurrence of actions anticipated by an agent and communication events that tracked in-person, radio, and telephone conversations. Communication events were the basis for the ATC dialogue presented during the execution of the scenario. Additional effort in the area of event recording and the provision of data presentation tools would be very useful in agent performance analysis. In a Phase 3 test case, OMAR simulation data was transferred to a Macintosh, and Excel graphing capabilities were used for presentation. This path should be explored further as well as paths local to the Sun. Currently, the event histories are not recorded on disk. Recording simulation run data is not a simple problem, and it should be addressed.

Agent Visualization and the Three-Dimensional Workplace

OMAR has excellent trace capabilities that may be customized to meet application demands. However, it became apparent early in the development of the ATC scenario that the debugging of the en-route controller's developing capabilities was going to be extremely tedious if it used only the textual trace. The anthropometric model, Kate, was developed during Phase 3 to provide a visualization of the en-route controller's motor activities. Even in the very simple form which Kate currently takes, real-time insight into agent behavior that is not available in the trace output is provided. Since virtually all agents of interest for modeling exist in a three-dimensional world, the layout of a three-dimensional workplace was essential to provide a place in which Kate could "operate."

There are several options open for the further development of the workplace and anthropometric models. In the short term, a few limited extensions will make it easier to develop the three-dimensional model of the workplace. These extensions would focus on the macros used to define the relative positions of objects in the workplace. Slightly more extensive extensions to Kate will make the utility easier to use. These primarily involve path planning for her motor activities. Furthermore, as currently implemented, Kate functions primarily as a display device. It would also be possible to use the human-model framework to assign a Kate-based "skeleton" to each operator in the system. A longer-range view should include evaluations of the use of a more fully developed anthropometric model such as Jack (Badler *et al.*, 1993) and a computer-aided design (CAD) system as the basis for assembling the three-dimensional workplace.

The Mirage GUI-building system is currently used in the OMAR system to build simulations of real-world workstations and panels. However, Mirage could also be used as a basis for pop-up displays which the analyst could attach to specific system values for "on the fly" display of real-time variables at run-time.

Standard Operating Procedure Editor

One important area of HPP model application is operability experiments. These experiments are designed to assure that system objectives are met and that operating procedures accomplish these objectives. A typical goal of the experiments is to evaluate an allocation of tasks among operators and between operators and equipment. Designing standard operating procedures (SOPs) is an important part of the process. A graphical editor to support the development of SOPs would be an important productivity aid in this process.

To be successful, the development of an operating procedure editor must address two problems that have been difficult to overcome in the past. First, graphical procedure editors have generally not been well-received by their users, and very few graphical programming languages now exist. Secondly, an HPP model that has multitasking capabilities and a set of operator primitives defined at the level of the SOPs must be available. The basis for building the graphical editor here was founded on the experience gained in developing the OMAR Procedure Browser and editing capabilities of the Concept Editor. Experience in developing the ATC scenario suggests that a generic set of appropriate primitives can be developed in an HPP model. Particular applications will require domain-specific SOP primitives.

REFERENCES

Abrett, G. (1991). Concurrent goals in an elaborate simulated world. *Proceedings of the Conference on AI, Simulation, and Planning in High Autonomy Systems*. Cocoa Beach, FL: IEEE Computer Society Press.

Abrett, G., & Burstein, M.H. (1987). The KREME knowledge editing environment. *International Journal of Man-Machine Studies, 27,* 103-126.

Abrett, G., Deutsch, S.E., & Downes-Martin, S. (1990). Two AI languages for simulation. *Transactions of the Society for Computer Simulation Special Issue on Artificial Intelligence and Simulation, 7*(3), 229-250.

Adams, J.A. (1989). *Human factors engineering*. New York: Macmillan.

Adams, M.J., Tenney, Y.J., & Pew, R.W. (1991). *State-of-the-Art-Report: Strategic Workload and the Cognitive Management of Advanced Multitask Systems*. Wright Patterson AFB, OH: Crew System Ergonomics Information Analysis Center (CSERIAC) Publication Series.

Agha, G.A. (1986). *Actors: A model of concurrent computation in distributed systems.* Cambridge, MA: MIT Press.

Allport, A. (1993) Attention and control: Have we been asking the wrong questions? A critical review of twenty-five years. In D.E. Meyer and S. Kornblum (Eds.), *Attention and Performance XIV* (pp. 183-218). Cambridge, MA: MIT Press.

Allport, A. (1989) Visual attention. In M. I. Posner (Ed.), *Foundations of Cognitive Science* (pp. 631-682). Cambridge, MA: MIT Press.

Badler, N.I., Phillips, C., & Webber, B.L. (1993). *Simulating humans: Computer graphics, animation, and control.* Oxford: Oxford University Press.

Bolt, Beranek, and Newman, Inc. (BBN). (1993a). *OASYS System Segment Specification (SSS): Baseline with SRS/IRS*. Wright-Patterson AFB, OH: Armstrong Laboratory, Logistics Research Division.

BBN. (1993b). *OASYS Concept of Operations* (unpublished draft). Wright-Patterson AFB, OH: Armstrong Laboratory, Logistics Research Division.

Brachman, R.J., & Schmolze, J.G. (1985). An overview of the KL-ONE knowledge representation system. *Cognitive Science, 9,* 197-216.

Corker, K.M., & Cramer, N.L. (1989a). *Methodology for Evaluation of Automation Impacts on Tactical Command and Control (C2) Systems: Implementation* (AFHRL-TR-90-9). Wright-Patterson AFB, OH: Armstrong Laboratory, Logistics Research Division.

Corker, K.M., & Cramer, N.L. (1989b). *Methodology for Evaluation of Automation Impacts on Tactical Command and Control (C2) Systems: Domain Selection and Approach* (AFHRL-TR-89-17). Wright-Patterson AFB, OH: Armstrong Laboratory, Logistics Research Division.

Corker, K.M., & Cramer, N. L. (1991). *Methodology for Evaluation of Automation Impacts on Tactical Command and Control (C2) Systems* (AFHRL-TR-91). Wright-Patterson AFB, OH: Armstrong Laboratory, Logistics Research Division.

Deutsch, S.E., Hudlicka, E., Adams, M.J., & Feehrer, C.E. (1993a). *Research, Development, Training, and Evaluation (RDTE) Support: Delivery Order #1 - Computational Cognitive Models* (AL/HR-TP-1993-0072). Wright-Patterson AFB, OH: Armstrong Laboratory, Logistics Research Division.

Deutsch, S.E., Adams, M.J., Abrett, G.A., Cramer, N.L., & Feehrer, C.E. (1993b). *Research, Development, Training, and Evaluation (RDTE) Support: Operator Model Architecture (OMAR) Software Functional Specification* (AL/HR-TP-1993-0027). Wright-Patterson AFB, OH: Armstrong Laboratory, Logistics Research Division.

Deutsch, S.E., & Palmucci, J. (1992). *Information Flow - Interim Report* (BBN Report No. 7687). Cambridge, MA: Bolt, Beranek, and Newman, Inc.

Dreyfus, H.L., & Dreyfus, S.J. (1986). *Mind over machine*. New York: Free Press.

Edelman, G.M. (1987). *Neural darwinism: The theory of neuronal group selection.* New York: Basic Books.

Hintzman, D.L., & Harty, A.L. (1990). Item effects in recognition and fragment completion: Contingency relations vary for different subsets of words. *Journal of Experimental Psychology: Learning, Memory, and Cognition, 16,* 955-969.

Holyoak, K.J. (1991). Symbolic connectionism: Toward third-generation theories of expertise. In K.A. Ericsson & J. Smith (Eds.), *Toward a General Theory of Expertise: Prospects and Limits.* Cambridge, MA: Cambridge University Press.

Koestler, A. (1976). *Janus: A summing up.* New York: Random House.

Logan, G.D. (1988). Toward an instance theory of automatization. *Psychological Review, 95,* 492-527.

Logan, G.D. (1992). Shapes of reaction-time distributions and shapes of learning curves: A test of the instance theory of automaticity. *Journal of Experimental Psychology: Learning, Memory and Cognition, 18,* 883-914.

Orasanu, J.M. (in press). Shared problem models and flight crew performance. In N. McDonald, N. Johnston, & R. Fuller (Eds.), *Aviation Psychology in Practice.* Aldershot, UK: Ashgate.

Orasanu, J.M. (1990). *Shared Mental Models and Crew Decision Making.* Princeton, NJ: Princeton University, Cognitive Science Laboratory.

Posner, M.I., & Peterson, S.E. (1990). The attention system of the human brain. *Annual Review of Neuroscience, 13,* 25-42.

Rasmussen, J. (1983). Skills, rules, and knowledge; signals, signs, and symbols; and other distinctions in human performance models. *IEEE Transactions on Systems, Man, and Cybernetics,* SMC-13, 257-266.

Riccio, D.C., Rabinowitz, V.C., & Axelrod, S. (1994). Memory: When more is less. *American Psychologist, 49,* 917-926.

Steele, Jr., G. (1990). *Common LISP: The language* (2nd ed.). Bedford, MA: Digital Press.

Young, M.J. (1992). *A Cognitive Architecture for Human Performance Process Model Research.* (AL-TP-1992-0054). Wright-Patterson AFB, OH: Armstrong Laboratory, Logistics Research Division.

Young, M.J. (1993a). *Successively Approximating Human Performance* (AL-TP-199-00). Wright-Patterson AFB, OH: Armstrong Laboratory, Logistics Research Division.

Young, M.J. (1993b). *Human Performance Models as Semi-Autonomous Agents.* (unpublished draft). Wright Patterson AFB, OH: Armstrong Laboratory, Logistics Research Division.

ACRONYMS

AI	Artificial Intelligence
AIRT	Automation Impacts Research Testbed
AL-HRD	Armstrong Laboratory - Human Resources Directorate
ANN	Artificial Neural Network
ATC	Air Traffic Control
BBN	Bolt, Beranek, and Newman, Inc.
CAD	Computer-Aided Design
CLIM	Common LISP Interface System
CLIM 2	Common LISP Interface System 2
CLOS	Common LISP Object System
GUI	Graphic User Interface
HPP	Human Performance Process
KL-ONE	Knowledge Language
KREME	Knowledge Representation Editing and Modeling Environment
LISP	List Processor
OASYS	Operability Assessment System
OMAR	Operator Model Architecture
PIASTRE	Piastre is a Structure Editor
RDT&E	Research, Development, Training, and Evaluation
SCORE	Simulation Core
SFL	Simple Frame Language
SOP	Standard Operating Procedure
SRS/IRS	Software Requirements Specification/Initial Requirements Specification
SSS	System Segment Specification
TNGS	Theory of Neuronal Group Selection
TP	Technical Paper
TR	Technical Report

VORTAC Very High Frequency Omnidirectional Range/Tactical Air Navigation

WAO Weapons Assignment Officer

U. S. Government Printing Office 1995 750-071/00137