University of Nottingham
ICL Institute of Information Technology

Mechanisms for Routinely Tying Cognitive Models to Interactive Simulations

Roberto L. Ong
MSc Information Technology specialized in Intelligent Knowledge-Based Systems

Supervisor
Dr. Frank E. Ritter

Submitted September, 1994, in partial fulfillment of the conditions for the award of the degree of MSc in Information Technology.

.......To HIM

"My love is oft-times low,
My joy still ebbs and flows;
But peace with Him remains the same,
No change Jehovah knows.

I change, He changes not;
God's Christ can never die;
His love, not mine, the resting-place,
His truth, not mine, the tie."

To my parents
who taught me the ways of life
and made me what i am...

Contents

ACKNOWLEDGEMENTS

I began doing this project after a series of marathon discussions with Dr. Frank E. Ritter. Through that short period of time, he took me in as his supervisee. From that time on, sheer long hours of hard work was poured into this project to bring forth what I think is quality and worthwhile work.

I would like to send my deepest and sincere gratitude to Dr. Frank E. Ritter, for his patience, encouragement, numerous advice, and never-ending support throughout the duration of this project. For the numerous meetings we had about work and career advices. For all the help, the humour, the yelling that I sometimes needed and most importantly, for bringing out the best I could possibly be. Frank, what can I say - I take my hat off. Thank you for being a mentor and friend.

To the members of the Cognitive Modelling Unit, Sarah Nichols, Josef Nerb, and Martin Kindsmueller, that was formed during this summer from which I am a part of, thank you for the friendship, the laughter and the companionship. Particular mention is made to Josef for his valuable assistance in Soar and Lisp and for teaching me new things.

To the Psychology Department for providing me with the necessary accounts, software, and assistance - thank you.

I am also grateful to my friends, for all their support and encouragement. In particular to Irene Low, for the humour, the friendship, and for believing in me. And to my special friend, Felicia Teo, thank you for the friendship, the laughter and the inspiration you have given me that I couldn't think of having without.

Finally, to my friends and family back home, though far away yet they make me feel at home. I - THANK YOU.

ABSTRACT

Creating simulations has been hard; creating cognitive models (i.e., Soar models) is equally hard. The interface that would make the two processes talk to each other is also difficult. Several projects have developed systems (or mechanisms) for tying cognitive models to simulations. But these projects fell short in several ways, and they are not powerful enough to be used for interactive real-time tasks.

This project presents a tool and mechanism for routinely tying a Soar model to an interactive simulation. We looked at how to create an environment that allowed both Soar models and subjects to talk to an interactive simulation (e.g., Air Traffic Control). The integration of both the Soar model and the simulation is realized through an interprocess communication (IPC) mechanism in UNIX called *sockets.* The generality, robustness, and the lack of need to modify sockets is what allows the process to be made routine. But the manual, supplying code for Soar and code for Garnet, which is the whole system, is what will make it routine. Thus, it would allow Intelligent Knowledge-Based Systems (IKBSs) to be routinely hooked up to simulations — similar to providing a routine way to get to camp.

The results of doing this project will illustrate the following things: (a) an ATC simulation that displays planes, beacons, and other objects; (b) a Soar model that performs the basic ATC task; (c) an improved socket utility ready for routine use, used by both the ATC simulation and another Soar model; (d) the different methods of interpreting a Soar model doing the ATC task; and (e) new ways of doing Input/Output in Soar.

1. INTRODUCTION

How do humans interact with the real world? It is a process that starts when a human subject first interacts with a real-world task, for example, landing an airplane. He tries to learn how to interpret data (e.g., gauges, compass, and so on) and task instructions. He then applies this knowledge to land planes safely and successfully. Over time, with constant interaction with the task, the subject acquires knowledge to easily land planes. This routine procedure of interacting with the task is particularly important to cognitive modelling. Cognitive models try to model human behaviour. Thus, to interact with the world like human subjects do, they must acquire procedures that make routine interaction with the world easy. Yet, how does one routinely tie a cognitive model to an interactive task like landing airplanes?

As a start, researchers in recent years have been developing systems (or mechanisms) that allowed Soar models to routinely interact with simulations of real-time tasks that both subjects and model can use. Systems like Stopwatch (Rogers, 1994), SimTime (Nelson, 1994), and New-Soar-IO (Pelton, 1994) have offered a way to routinely tie a Soar model to a simulation. They are applications ranging from timing behaviour in Soar to interacting with the external world through simple input/output routines. But with their limitations of just being able to perform simple tasks (New-Soar-IO) and not interacting with a real-time task at all (Stopwatch and SimTime), a new approach must be explored and developed to resolve these limitations. This new system should have the ability to create, (a) a simulation of a real-time interactive task that requires real-time decision making, (b) a Soar model that interacts and learns like a human subject, and (c) a communication (or interface mechanism) channel that is robust, stable and general. The communication channel would provide the mechanism that routinely ties a Soar model to a simulation. It must be expandable and powerful enough to illustrate the *benefits* of routinely tying cognitive models to simulations.

This project explores the possibility of using this routine approach to Simulation (by Soar) tie with a model of human behaviour in an interactive real-time (RT) task – an Air Traffic Control (ATC) simulation. It will be attempting to model the ATC task by creating a simulation, a Soar model,[1] and allowing them talk to each other.[2]

[1]Soar is an example system for cognitive modelling, which will be discussed further in Chapter 2. A certain familiarity is assumed.

[2]For brevity in writing, the use of the word *simulation* will refer to the simulation of the world, *the model* will refer to the Soar model, *the interface* to the communication medium between the model and the simulation, and *the system* to refer to the entire system being developed in this project.

1.1 Soar-Simulation Relationships

Figure 1 shows the three possible relationships between a Soar model and a simulation: (a) as a knowledge-analysis tool, (b) as an interacting performance model, and (c) as an embedded user model.

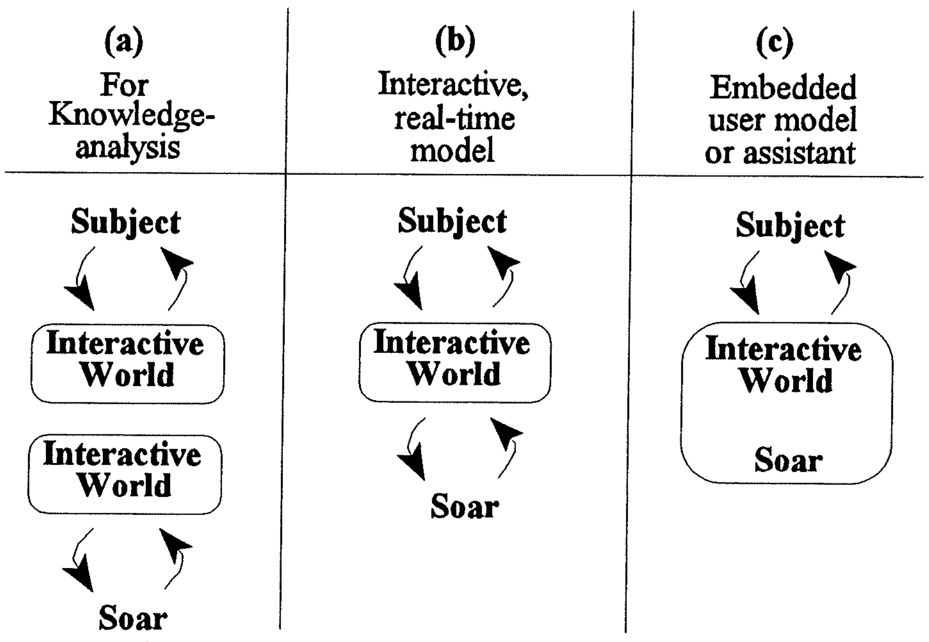


Figure 1. Possible relationships between Soar and a simulation.

The distinctions primarily arise from the way the Soar model interacts with the world. In Figure 1a, two simulations must be created, one for the subject being modelled and one for the model. This kind of relationship is desirable when the subject's world is difficult to model realistically or the subjects have already been run. The problem arise in the increased amount of work to be done (i.e., creating two simulations) and in making the two simulations equivalent. For Figure 1a to be implemented properly, every effort should be made to keep the two simulations equivalent, for example, presenting the same view of the world, providing the same operations, and ensuring that the operations provide the same result (Ritter & Major, 1994). In

Figure 1b, a situation occurs where the model interacts with the same simulation the subject does. In addition to reducing the number of simulations that must be implemented, this ensures that both model and subject see the same world, and provides a rapid feedback loop for developing the model. This approach is particularly important, and a necessary step, if one is to go a step further in creating the situation in Figure 1c. Figure 1c depicts a situation where the model can be used to support the user. This can occur by using the model to choose likely operators to present or highlight, or more likely to allow the model to directly provide assistance to the user by performing sub-tasks. Over time, these approaches will suggest where the interface could be made easier for the subject.

In many cases, the utility and effectiveness of each of these approaches will be dictated on how successfully they are implemented and used. This project provides an opportunity to investigate just how far one could go in implementing one of these approaches, namely 1(b).

1.2 Task Objectives and Requirements

The project explores the possibility of modelling human problem solving that involves extensive interaction with the outside world. The task is to create the system shown in Figure 2, an interactive, real-time model consisting of three parts, (a) the interactive world, (b) the Soar model of the subject, and (c) an interface between the Soar model and the simulation that is reusable and easy to use.

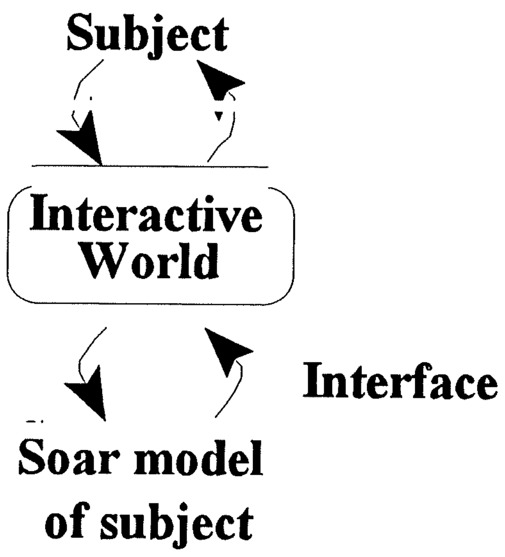


Figure 2. Interactive Performance Model.

The main goal of this undertaking is to implement the above system in an Air Traffic Control (ATC) simulation task as illustrated in Figure 3. The implementation is broken down into three parts, *the ATC simulation* (on the left), *the Soar model* (on the right), and *the interface* (at the bottom).

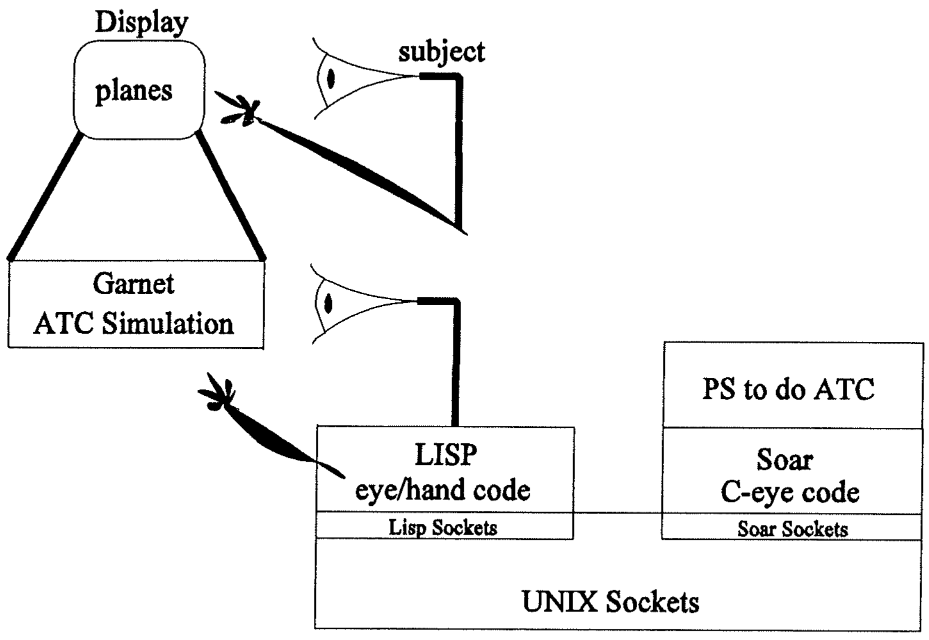


Figure 3. System implementation.

The ATC simulation is the interactive world that this project simulates. It does the handling of incoming planes, landing planes, overflights, and other things covered under a grossly simplified ATC task. This task differs from a real ATC task in several important ways (Nolan, 1990): (a) real ATC uses paper flight strips to keep track of planes' altitudes, heading, and estimates of crossing points in time and space, (b) real ATC uses sectors and zones, (c) real ATC handles planes through voice commands, and a host of other ways. The big displays one often expects, like those found in control towers in movies, are used as static, current display, while the flight strips are illustrated as plane information about what and where things are going to happen. In this task, there will only (initially) be the display, which will be rather dynamic. Table 1 summarize the requirements needed for the creation of the simulation.

Table 1. The requirements needed for the creation of simulation.

- The simulation's actions must be recorded.
- The simulation must update itself approximately every 200 ms (5Hz).
- The subject and the model should see the same thing. This implies a theory of visual processing.
- All controls must be manipulable by model and by a subject.
- A visual display for two types of users, subjects and analysts.

The Soar model is a simple cognitive model that would (initially) be for testing. This model will act as a client (in a UNIX client-server model) and will initiate conversation proceedings with the simulation (this conversation will serve as the medium from which all learning will take place). It can issue commands to the simulation, for example, '***send-scope-info***', and in turn receive data from the simulation depending on the command issued (here, information of all planes seen in the radar scope). The incoming data are interpreted, manipulated, and then deposited on the top state as working memory elements (WMEs) to facilitate learning. The objective of making this model simple is to be able to establish the link between the Soar model and simulation, and provide routine interaction between them.

The interface is the communication link between the simulation and the Soar model. It is the mechanism that provides the most basic communication channel that would link the simulation and the model, and allow two-way communication between them.

We started with a socket utility from Mertz (1994). We then modified and adapted this code, so that it could truly be used as a utility for use in the ATC task and other similar tasks with little (or no) modifications. As a result, this work was generalized and cleaned-up enough so that it could be done routinely. This allowed Intelligent Knowledge-Based Systems (IKBSs) to be hooked up to simulations. Moreover, this work would provide a way to explore a less developed area of Artificial Intelligence (AI) and Psychology, and will offer a way for intelligent agents to have a place to play and learn. In addition, it will also serve as a foundation from which to build systems that model human behaviour in all aspects of doing interactive RT tasks.

In many cases, simulation tasks are likely to change over time. The exact parameters of the simulation are not well specified, and will only be found by building it. The Soar model will also influence task development. Once the system has been created and initially tested, it may suggest ways to modify the simulation, either to explore additional behaviour, or as a way to explore the effect of the world on how the task is solved. Moreover, complexity can be added to the simulation for it to acquire real-time ATC task, and to the Soar model for more complex learning that rivals that of real ATC experts.

In summary, the project's objectives are the following: (a) to extend our understanding of the nature of tying cognitive models to simulations; (b) to provide a general interface between simulations and Soar models; (c) to explore tools for modelling real-time cognitive behaviour by building a simple Soar model of the task; and (d) to learn new ways of doing and interpreting I/O for Soar models.

1.3 Paper Preview

This paper starts by laying out parameters for the tools required in building the system. Existing tools are referenced from a technical report by Ritter and Major (1994)[3] emphasizing the tools chosen for this project with respect to their strengths and weaknesses in comparison to the other tools reviewed in the report. A review of existing systems (that offer a way of routinely tying cognitive models to simulations) then follows describing each of them and providing a summary of their usage. This will also include the lessons that these systems provide and the limitations they have concerning the objectives and requirements layed out for this project. Succeeding chapters discuss, (a) the implementation of the system, (b) a description of how the Soar model runs through a sample run, (c) the analysis of the Soar model, and (d) the usage of the interface mechanism provided by this project and by others. These chapters describe what each part of the system is doing, and how each part is implemented. A description of a sample run (including an analysis) makes it clearer how the task is performed. Also included in these chapters is a discussion of how the interface mechanism has been used by other tasks, stating the degree of generality that this interface provides and additional enhancements that were found to be needed.

Penultimately, we look at the road ahead and take a glimpse of where the project is headed. This direction would illustrate what this project can achieve in actually modelling human behaviour in an ATC task. Finally, the concluding remarks summarize the achievements, failures, and recommendations from this project.

[3]This technical report reviews tools and tool requirements for developing simulations for Soar to play with. It covers discussions on simulation languages, development environments for Soar, and communication channels for developing an Air Traffic Control (ATC) task and the Tower of Nottingham (ToN) task to provide routine interaction between simulation and Soar model.

2. SYSTEM TOOL REQUIREMENTS

Tools form the basic requirement in the performance of any task. They provide valuable assistance to bring a task to a completion within the allotted time and at the right specifications. It is therefore important that one select the right tools (or building blocks in this case). Choosing the right tools mean correct approach to the task leading to its faster completion. Choosing the wrong tools mean doing the task wrongly, sometimes prolonging its completion and in the worst case doing the task all over.

This project like any other task requires that tool requirements be layed out before work is started. Each part of the task, (a) *the simulation*, (b) *the model*, and (c) *the interface* necessitates requirements be specified as the basis of selecting tools. The technical report by Ritter and Major (1994) provides a detailed review of existing tools and requirements specifically for doing this project (and similar projects). This report is summarized here without further reference. The tools used and described here are taken from it. The following sections give a description of the requirements for each part of the task and a summary of the tools used.

2.1 Simulation Languages

Simulation languages come in different varieties, they may be standard programming languages or special languages and toolkits built solely for creating simulations and interfaces. Thus, the choice of the simulation language should be based on compliance with our requirements and specifications. Table 2 summarize the simulation language requirements needed in building the ATC simulation (the simulation).

Table 2. Simulation Language Requirements.

- It should provide visual interface for the subjects and model to use.
- It must update itself approximately every 200 ms (5Hz).
- It should provide real-time information as the task will be dependent on this to carry out real-time decisions.
- It can talk to a Soar model. This will enable both the simulation and Soar model to talk and pass data to each other.
- It should be easy to learn so that many can use it to build simulations for IKBSs to play with.
- It must be cheap or Free.

Garnet

Garnet[4] (Myers et al., 1993) is a user interface development environment for Common Lisp and X11 or Macintosh. It calls itself a "graphical user interface management system," but in reality it can be treated as an object-oriented graphical interface toolkit and building system. Garnet is a large scale system containing many features and parts that help you create graphical, interactive user interfaces. Some of these features[5] include, (a) a custom-oriented programming system that uses a prototype-instance model, (b) automatic constraint maintenance, and (c) a built-in, high-level input event handling.

[4]GARNET is an acronym for Generating an Amalgam of Real-time, Novel Editors and Toolkits.

[5]Garnet-FAQ (Myers, 1994) gives a detailed overview of Garnet features, hardware requirements, and frequently asked questions.

Garnet is the preferred simulation language because it could meet the requirements in Table 2. The simulation will need to run on what is natural for the model (UNIX for Soar) and what is natural for the subject (this will be a system that can provide good timing, such as the Macintosh or PC). The fact that Garnet is written in Lisp (such that learning the language will be easy because of this author's familiarity with Lisp and with most AI researchers' familiarity) gives advantages of the many features in Lisp, including speed of developing code.

Garnet provides users with an object-oriented programming environment that makes use of objects, gadgets, interactors, and packages.[6] It works as a prototype-instance model that allows graphical objects to be created by instantiating an object or group of objects (e.g., aggregadgets). For example, to create a representation of a prototype plane, an aggregadget must be created consisting of two small circles and a text tag (i.e., the representation of a plane used in this system). After the prototype has been created, you just need to create instances of that aggregadget to create more planes, say 10, 20 or more. And to make the planes move, an interactor is created and attached to the aggregadget. Interactors are Garnet tools used for creating user interfaces. They allow specific actions (e.g., change an object's position) to be done in respond to user actions, such as a button press or a mouse click. In our case, an animator interactor (further discussed in Chapter 4) is used. This interactor (as used in our simulation) makes its associated plane to move continuously across the radar scope. Thus, it makes building user interfaces easy. This feature and many others make Garnet a good tool to use (especially if one knows Lisp), and one that is easily implemented.

[6]The Garnet Reference Manual Version 2.2 (Myers et al., 1993) provides a complete description of Garnet, its features, and usage.

Other simulation languages like cT, SUIT and GALAXY are alternate tools that can be used in this task. cT is a Pascal-like language that includes a visual display and simple user-interface components. It is a low-level language that gives an advantage of easily programming it. Also, the same code (including the displays) can be run from UNIX machines to Macintoshes to PCs (i.e., it is quite portable). However, this language when used to create simulations needs a lot of coding (therefore unsuitable) because it is relatively a low-level language and supports no structures. SUIT is a C-based system that promises to be easy to use. If a Lisp-based system ends up being scrapped, this system would be a logical choice. GALAXY, the last of theses alternatives, appears to be a very useful system (based on its features and reviews), but for reasons of cost, it will not be used. These alternatives and many others could be used instead of Garnet, but availability, and being familiar with Common Lisp makes Garnet the best choice. In fact, the choice of the simulation language is a trade-off among: speed of coding, familiarity with the language, and the features provided. If all ends up being the same, their availability and cost will be the deciding factor.

2.2 Cognitive Modelling Tools

Two prominent cognitive modelling tools are Anderson's ACT-R and Newell et al.'s Soar. Both architectures provide production-learning capabilities and hierarchical goal structures as integral parts of the theory. Both allow us to model the process of analogical reasoning. And importantly, both allow us to build models containing only knowledge that could, at least in principle, be learned using mechanisms described within the respective theory.

For this project, Soar will be used because of available resources (software and people who know and can teach Soar), and the existence of a Soar community that provides assistance with regards to modelling. However, a good extension for this project would be to devise a way of porting the system to ACT-R.

Soar and (SDE)

Allan Newell (1992) calls Soar, "an exemplar of the unified theories of cognition, a cognitive architecture, which is realized as a software system." Waldrop (1988a) says, "it is a computer program that can learn from experience— and that may also explain the basic mechanisms of thought." In fact, Soar is a theory implemented into a language, that seems to provide a general model of human thought.[7]

According to Newell (1992), unified theories of cognition are single sets of mechanisms that cover all of cognition — problem solving, decision making, routine action, memory, learning, skill, perception, motor activity, language, motivation, emotion, imagining, dreaming, daydreaming, and so on (Newell, 1992). In Newell's view, Soar is the prototype of how researchers should devise theories of human cognition that encompass reasoning, learning, perception, motor control, cognitive development, emotion, and perhaps even such ineffable qualities as awareness, all within a single coherent framework. "Even if the mind has parts, modules, components, or whatever, they all mesh up together to produce behaviour (Waldrop, 1988a)," Newell says. "It is one mind that minds them all (Waldrop, 1988a p 27)."

[7]Waldrop (1988a, 1988b) gives a very good introduction to Soar and the Unified Theory of Cognition (UTC). These two articles as well as Newell (1992) are strongly suggested for reading for anyone to get acquainted with Soar and UTCs.

The computer program known as Soar is basically a program that learns while solving problems – any kind of problem if given the knowledge. It is a problem-solving architecture taking the form of a hierarchical set of *problem spaces*, the cognitive arena where all mental activity being devoted to a given task takes place. A problem space in turn consists of a set of *states*, which describe the situation at any given moment, and a set of *operators*, which describe how the problem-solver can change the situation from one state to another. When for some reason, the process of applying operators to states cannot directly proceed, Soar recognizes an *impasse*, and sets up a subgoal with its own problem space whose aim is to resolve the impasse and allow processing to resume in the original space. In chess, for example, the problem space would be "a chess game," a state would consist of a specific configuration of pieces on the chess board, and an operator would consist of a legal move, such as "Knight to King-4." The task of the problem-solver is to search a sequence of operators that will take it from a given initial state (say, with the pieces lined up for the start of the chess game) to a given solution state (the opponent's king in checkmate). More generally, says Newell, Soar fulfils a basic requirement of any intelligent system: its reasoning is directed in pursuit of its goals, a reasoning that occurs in human problem solving.

The knowledge in Soar is realized (or implemented) in a "production system" architecture. The idea here is to encode each bit of knowledge as a condition-action rule of the form, "IF *this* is the case, THEN do that." At every elaboration cycle,[8] these production rules fire when their conditions are met. All firing rules then express opinions such as "operator Q1 (take your opponent's queen) is better than operator Q2 (take your opponent's pawn)," or "operator Q7

[8]The Soar manual (Laird et al., 1993) provides a detailed introduction to the innards of Soar and its features.

(sacrifice your bishop) is best." in the form of preferences. These preferences are then evaluated and the operator with the best preference (in this case operator Q7) is applied. (these are, of course, naïve rules for chess.)

One distinctive feature of Soar is its learning mechanism called *chunking*. The basic idea of chunking is simple: whenever Soar resolves an impasse, it remembers how. More precisely, Soar encodes the results of its problem-solving as a new condition-action rule a "chunk" – and then stores it away in memory where it operates like any other rule. Its conditions are the relevant contents of working memory at the time the impasse arose; its action is the new solution. The next time it encounters a similar impasse, it can leap directly to the solution without repeating the intervening steps. And thus Soar can spontaneously pass from the slow, painful, trial-and-error problem-solving characteristics of a novice, to the near-instantaneous insight characteristic of an expert. It does not have to be programmed with expertise, because it learns expertise. In terms of human problem solving– chunking gives an accurate methodology on how human tackles a problem. When they encounter a problem along the way, they devise steps to solve it and then stores it in their brain. The next time they encounter the same problem, they just remember the solution and apply it without going through the trouble of devising the steps themselves. Figure 4 illustrates the idea of chunking.

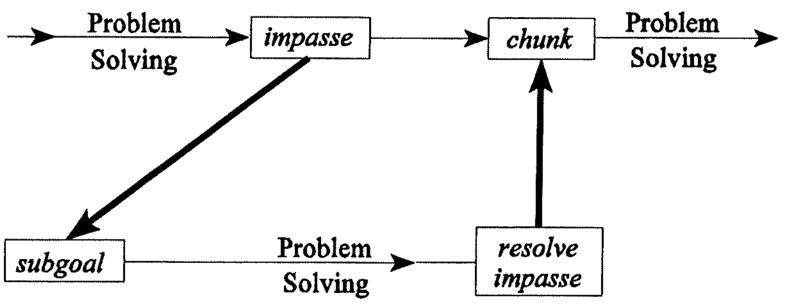


Figure 4. Chunking– Learning from experience. When Soar reaches an impasse in its problem-solving– that is, when it does not know what to do next– it automatically sets up a subgoal to resolve the impasse. When it succeeds, it goes back to where it left off and simultaneously encodes a new "chunk" of knowledge that will keep it from ever having to suffer that particular impasse again [adapted from Waldrop, 1988a].

Finally, Soar release version 6[9] and later having been written in C provides facilities for supporting routine interaction to the outside world. These facilities allow a programmer to go into the innards of the Soar program and create necessary commands and functions to provide routine interaction to the outside world.[10] Moreover, with Soar Development Environment (SDE)[11] accompanying Soar development, it allows easier and faster coding.

[9]Soar version 5 (and below) having been written in Lisp had little (or no) problem with regards to interacting with the world. But for version 6 (and later) we now need a way for Soar to interact routinely.

[10]The Soar manual (Laird et al., 1993) gives a detailed description of Soar's I/O interface. It includes features that lets the user add basic C code, I/O commands and interface routines.

[11]The SDE README file (Hucka, 1994) gives a detailed description of the features of SDE.

2.3 Soar-Simulation Communication Channels

Interprocess communication (IPC) concerns the mechanism by which independent processes communicate with each other. This communication medium provides the interface necessary for a simulation to talk to a model, and is particularly important if it is to be provided as a mechanism (and utility) that routinely tied the model and simulation together.

As the Soar model (the model) runs, it must provide the simulation with task commands and receive inputs from the task simulation (the simulation). The amount of information in the ATC task passed (initially) between the model and simulation are not likely to be large. At present, the input from the simulation solely represents the result of visual scanning of the simulator's display, but a more ambitious goal is to incorporate a verbal input (which is not part of the project reported here). Table 3 provides a summary of the requirements for the choice of a communication medium.

Table 3. Interprocess Communication Requirements.

- It has the capability to see and log data.
- It must be flexible to handle any kind of data and modifiable.
- It is robust and stable.
- It could communicate processes across machines within a network. This will provide portability of the entire system.
- It must be easy to build and use.

Table 4 provides a summary (taken from Ritter and Major, 1994) of the features of various communication mediums. The choice of the medium is dependent on compliance of the requirements above and below. In the ATC task, sockets were chosen.

	Bandwidth	**Ease of Creation**	**Flexibility**	**Robustness**
Plain Files	Bad	Good	Bad	OK
Sockets	Good	OK	OK	Good
GNU-Emacs Processes	OK	OK	Good	Maybe Bad
Joint Compilation	V. Good	Bad	OK	V. Good
Apple Events	Good	Bad	Bad	OK

Table 4. Features of various communication channels.

Sockets

Sockets are an IPC mechanism that allows processes to talk to each other across different machines. It is this across-network capability that makes them useful. For example, the **rlogin** utility, which allows a user on one machine to log into a remote host, is implemented using sockets.

Process communication via sockets is based on the UNIX client-server model. One process, known as a *server* process, creates a socket whose name is known by other *client* processes. It listens for incoming requests for connection from the client processes. The client processes then talk to the server process via a connection to its named socket. To do this, a client process first creates an unnamed socket and requests that it be connected to the server's named socket. Depending on the availability of connection set by the server process, a successful connection returns one file descriptor to the client and one to the server, both of which can be used for reading and writing. Once a socket connection is made, processes can start their conversation. Figure 5 shows an illustration of a socket connection.

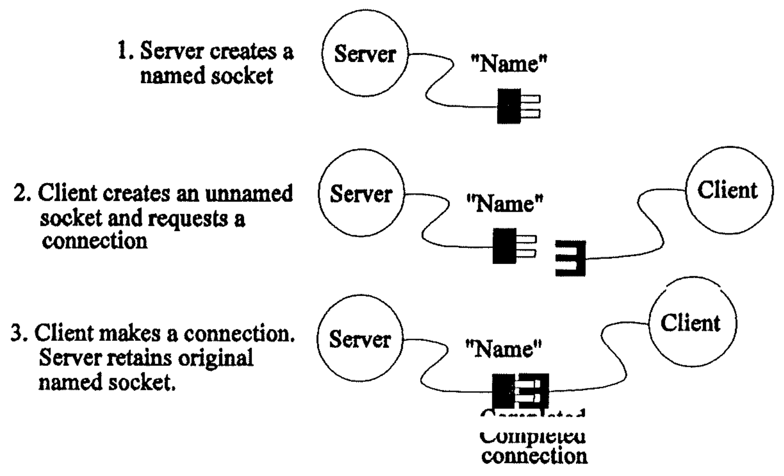

1. Server creates a named socket
Server
"Name"
2. Client creates an unnamed socket and requests a connection
Server
"Name"
Client
3. Client makes a connection. Server retains original named socket.
Server
"Name"
Client
Completed connection

Figure 5. The socket connection [adapted from Glass, 1993].

Sockets are similar to files in how processes read and write to them, but they do not have disk space allocated to them. They provide a stable, and robust communication link between the model and the simulation across machines in the network. They also provide a mechanism that can routinely tie a model to a simulation. Other communication mediums do exist (see Ritter and Major, 1994), but they're not good enough to satisfy the criteria in Table 4. Choosing sockets as the medium is just a question of implementing a mechanism that provides a robust, flexible and stable way of communication with a good bandwidth, and one that complies with the requirements in Table 3 and 4.

3. EXISTING SYSTEMS FOR ROUTINELY TYING COGNITIVE MODELS TO SIMULATIONS

This section examines three systems that have been developed by researchers in their endeavour to provide a mechanism that routinely tie a Soar model to interactive simulations. It gives an overview of what these systems do and what generality they provide. After a description of these systems: Stopwatch (Rogers, 1994), SimTime (Nelson, 1994), and New-Soar-IO (Pelton, 1994), we present a summary on how these systems fair in terms of power, usage and providing routine connections between models and simulations.

3.1 Stopwatch

Stopwatch, by Seth Rogers (1994), is a timer function that accurately times behaviour in Soar. It is similar to a stopwatch (hence the name), in that a user must provide a length of time in seconds (say, 5 seconds) and the system will count up until it reaches that time. All counting or timing is done on a base time that is the current time of the operating system (i.e., the time returned when *gettimeofday* is issued at the UNIX prompt). To activate Stopwatch, you must propose and implement an operator called '**wait**'. Then you must tell it how long should the timer be by giving it an argument *arg1* containing a float constant in seconds.

At the start of each elaboration cycle, the input function **check_stopwatch**, (a) creates working memory elements[12] (WMEs) on the top-state containing the input link, the *timer-done* flag (set to NO, since no timing has been done), and the *target-time* (set to INACTIVE, timing

[12]The ceation of WMEs is only done at the start of the first elaboration cycle or during an init-soar. This may also be called the initialization stage of each program run.

has not started) and (b) checks to see if *target-time* has been set to a number (i.e., not INACTIVE) in which time the timer starts running. The output function **start_stopwatch** done at the end of each elaboration cycle checks if an operator '**wait**' has been proposed and implemented. Once implemented, the value of argument *arg1* is taken, added to the system time (i.e., the seconds part return when *gettimeofday* is issued) and set as the *target-time*. Succeeding elaboration cycles activate the timer until it reaches the *target-time* when it is reset back to INACTIVE state.

The timing is done by getting the seconds part of the time return by *gettimeofday* and comparing it to the *target-time* (stored as a global variable). When *target-time* is exceeded, (a) the *timer-done* flag is set to YES (i.e., timing has been done) and added as a WME, (b) the *target-time* is set to INACTIVE, and (c) the whole process repeats. It should be noted that timing is done in seconds. For example, if you want to know when x seconds have elapsed, you put a structure like this on the top-state:

```
(S1 ^timer-link T1)
(T1 ^command wait ^arg1 x)
```

Then during every input cycle, Soar checks if the timer is done, and if it is, adds a WME like (O1 ^timer-done yes), where O1 is an input-link object.

The Stopwatch system is the kind of system that we create when we're just starting to routinely tie cognitive models to simulations. It is a very simple system (as we shall see later) that provides us with a tool that times behaviour on a specified length of real-time (in seconds). It is robust and routine, but provides the smallest amount of interaction (almost non-existent).

Although simple, it would definitely pass as a preliminary system and would count on our learning experience to reach what we are trying to achieve in this project, but it needs to be extended to do what we need. One possible extension is to add a feature that lets the user set the base time/start time from which to start counting. But in the long run, Stopwatch would have to be made more complex (not just a mere simulation of a timer/stopwatch) and interactions to the outside world would have to be added.

3.2 SimTime

The SimTime[13] system simulates the passage of real time (hence the name) as a function of the cognitive behaviour of a Soar model. It allows you to monitor the time, tell how to increment the time and how you want your system to use or respond to the passage of time. In order for SimTime to monitor the time, you must tell it the initial time for your simulation in either of two way. The simplest way to set the initial time is to use the command compiled into Soar '***simtime set <num>***', where ***<num>*** is an integer value in whatever units you wish to use (milliseconds is traditional). In this case, you must explicitly reset the time when you wish to give it another value. The other way is to place an initial time on the first line of an "event-list" file; in this way, the time will be set automatically when you perform an init-soar. After this first line, the event-list may contain blank lines, comment lines (beginning with a semicolon), and event lines. The first column of an event line indicates a time (in arbitrary units, usually ms). The second column indicates an "event type", and must match to the name of an event type in your code. The remainder of each line is passed as an argument to the C function that implements the event.

[13]The SimTime user's guide (Nelson, 1994) provides a detailed description of SimTime, its command set, installation procedures, and some useful notes on using the system.

The description of how SimTime should increment the time is given to SimTime as a "timing-list". In this file, which is read in with a separate command, each operator may be given an independent duration. Or, the special term "OPERATOR" may be used to specify a time for all operators, regardless of their name. Alternatively the special terms "ELABORATION" and "DECISION", if included, give a base time to be added for each elaboration cycle and decision cycle, respectively. These methods may be employed in any combination; their effects will be additive. Negative operator durations are legal, but of questionable value, unless the elaboration/decision mechanism is being used as well.

Finally, there are a few different ways in which the system may use the time value. By issuing the command '***simtime wme on***', a working memory element containing the current time value will be deposited into the top-state of the running model. While this breaks the world/brain barrier, it is a convenient way of using the time value to perform interesting processing within the model, through elaboration productions for instance. A less controversial mechanism allows external events to happen at specified times, which may also place information into working memory. There are two options here. A powerful mechanism requires that you write your own C code. You define event names which may appear in the event-list that is read in from a file. Once these events are defined and tied to C functions that implement them, you can readily change the series of events that happen during any given Soar run. However, if you don't want to write C code, a less powerful but standard mechanism is to use the special events ADD-WME and REMOVE-WME (defined in SimTime) to let you get something small running quickly.[14]

[14]The syntax for these commands and other commands in the command set of SimTime are all described in the manual (Nelson, 1994).

SimTime as we say, is a notch up on the stopwatch system. It takes the timing capability of Stopwatch, made it more flexible (e.g., you can set the start time, endtime) and complex to measure the usage of time by a cognitive model. One simple example in its manuals shows a cognitive model monitor the passage of time in WMEs, and respond appropriately to it. A more complex example depicts a Soar model taking data from an event-list and a timing-list, together with some user-defined event-names. Once these events are defined and tied to C functions that implement them, you can readily change the series of events happening during any given Soar run. Indeed, SimTime makes full use of the passage of real time, which is particularly important if the cognitive behaviour you are modelling is time dependent. For example, in an ATC task wherein a slight variance in time may cause or abort a catastrophic plane crash.

The SimTime system can make extensive interactions with the outside world (in this case, the Soar model) only in a certain way, that is, if the outside world wishes. If the model is programmed to use and respond to the passage of time, it may do so. Otherwise, it can just sit and watch time pass, doing nothing. This obvious flaw could have been avoided if the outside world is interactive and provides a reactive environment. Clearly, a system should be explored and developed to provide interaction with an interactive task and New-Soar-IO (Pelton, 1994) is a step in that direction.

3.3 New-Soar-IO

New-Soar-IO by Gary Pelton (1994) describes a set of mechanisms that enable a Soar user to emulate the interaction their program would have with the external world without actually writing Soar I/O code. It consists of tools and features that effectively captures some (maybe less) features of Stopwatch and SimTime, added modifications and build it as a foundation for Soar to talk to the external world. It interacts with the external world by testing it, and depending on the response changes the top state appropriately (i.e., establishing a two-way communication interaction).

On the installation of the state in the top-context, New-Soar-IO adds an *external-world* link and an (^input-cycle-number 0) to the *external-world* link. The external-world-wme containing the *external-world* link gets deposited into the top-state with a unique id (in this case, the id starts with the letter 'I'). Subsequent input cycles then remove the counter *input-cycle-number*, add 1 to it, and add it back to the *external-world.* The *external-world* link can be used by the Soar model to tie anything that has to do with the external world and the *input-cycle-number* (i.e., a counter) as a clock mechanism that might be used to delay the response to actions in the external world. An example of a delayed response is pulling back on the stick in an aircraft. This doesn't immediately change your direction. Here, productions would later adjust the aircraft WMEs.

On every subsequent output cycle, if a new intention is added, the old intention is removed from the *external-world* (if it exists). A copy of the new intention is then added to the *external-world* with the same structure and constants (as the original) but all identifiers have been replaced. For example, if the top-state contains

(S1 ^external-world I1 ^intention I2)

(I1 ^input-cycle-number 2 ^intention I3)

(I2 ^go-home yes)

Adding a new intention gets rid of the old one (if it exists), and adds a new one to *external-world.*

```
(S1 ^external-world I1 ^intention I4)
(I1 ^intention I5 ^input-cycle-number 3)
(I4 ^go-home no)
```

New-Soar-IO succeeded in providing a mechanism for routinely tying a Soar model to an interactive simulation (in New-Soar-IO, the external world). It talks to the external world by providing a clock mechanism and enabling it to emulate the interaction between model and external world. This system is an easy way to start if all modelling done is just for simple tasks (e.g., turning a light switch on and off). But when considering huge real-time interactive task, like the ATC task, the disadvantages precludes its use. To start with, New-Soar-IO can only interact with a simulation written in Soar which are very simple, temporary systems. And for all intent and purposes, creating simulations in Soar is a mess because the production system was not intended for general simulation, and there is no visual display. In order for a Soar model to play with a real-time interactive simulation, the simulation must reside and be developed in its natural environment (i.e., with the visual graphics and all) and just provide a mechanism of tying this two distinct processes together.

3.4 Summary

The three systems presented provides an overview of the steps taken by researchers in their endeavours to provide a mechanism to routinely tie a Soar model to an interactive simulation. As noticeable, each system presented (and reviewed) increased in complexity and is a notch better than the previous one. This stepped presentation was done to show the steps taken in scaling the task of routinely tying cognitive models (i.e., Soar models) to interactive simulations. Indeed, each of these systems provided valuable lessons that paved the way for the next generation of systems to be explored and developed.

Illustrating these systems in analogy, first, the Stopwatch system provided timing capabilities for Soar. Second, this timing capability was taken, made more flexible, and powerful to create SimTime. Since all SimTime provides is essentially a one-way communication utility, New-Soar-IO was developed to solve this limitation. New-Soar-IO provided a clock mechanism and a two-way interaction between model and world. One valuable lesson learned from all of these is that to create systems that provide routine interactions between a Soar model and a simulation, the Soar model and the simulation should reside on environments where they work best and well suited, and just provide the mechanism that make the two talk. This means that the model should live in Soar and the simulation (depending on the language) in a separate environment. Moreover, the interface mechanism must be made general so that other simulations might used it with little or no modifications.

Indeed, these systems are testaments that the mechanism for routinely tying a cognitive model to interactive simulations is not far fetched. But these systems must be extended or a new system developed if we are to model cognitive behaviour in a highly interactive task. Over time, this new system by providing routine interaction would eventually allow us to develop models that learn and think the way humans interact with the outside world.

4.0 SOAR-GARNET — Nott-a-bad System

The system implementation of this project is illustrated in Figure 6 (same as Figure 3). On the left is the entire simulation,[15] on the right the Soar model, and the interface mechanism at the bottom which links the two sides together.[16] We discuss each box in turn.

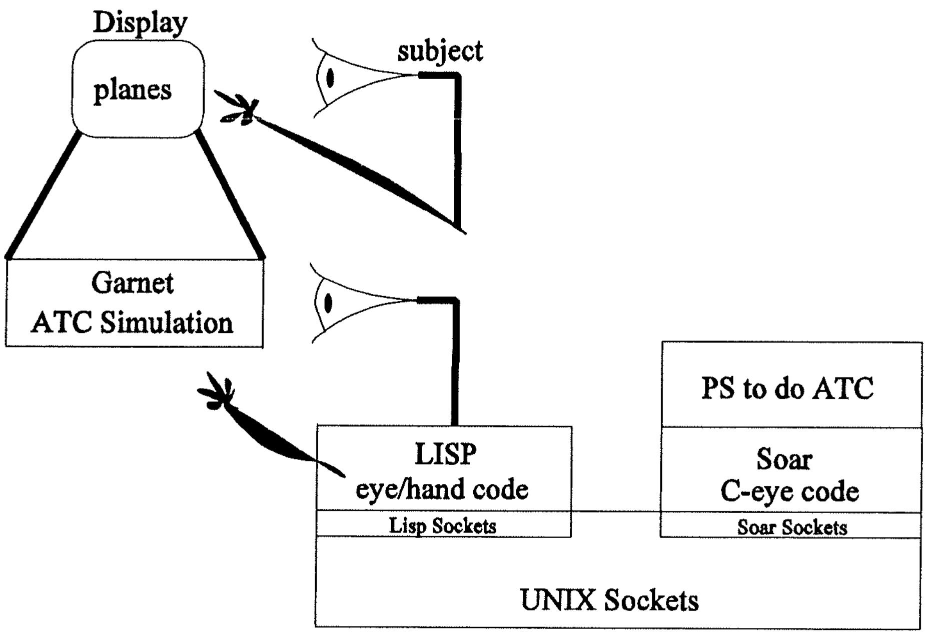


Figure 6. System implementation.

4.1 ATC Simulation

The ATC simulation was implemented using Garnet objects and gadgets[17], such as menubars, windows, text and interactors. Once implemented, these objects are used in Lisp functions to

[15] Appendix A contains the release notes about the Soar-Garnet system implemented in this project. It gives information on where this code is available and how to use it.

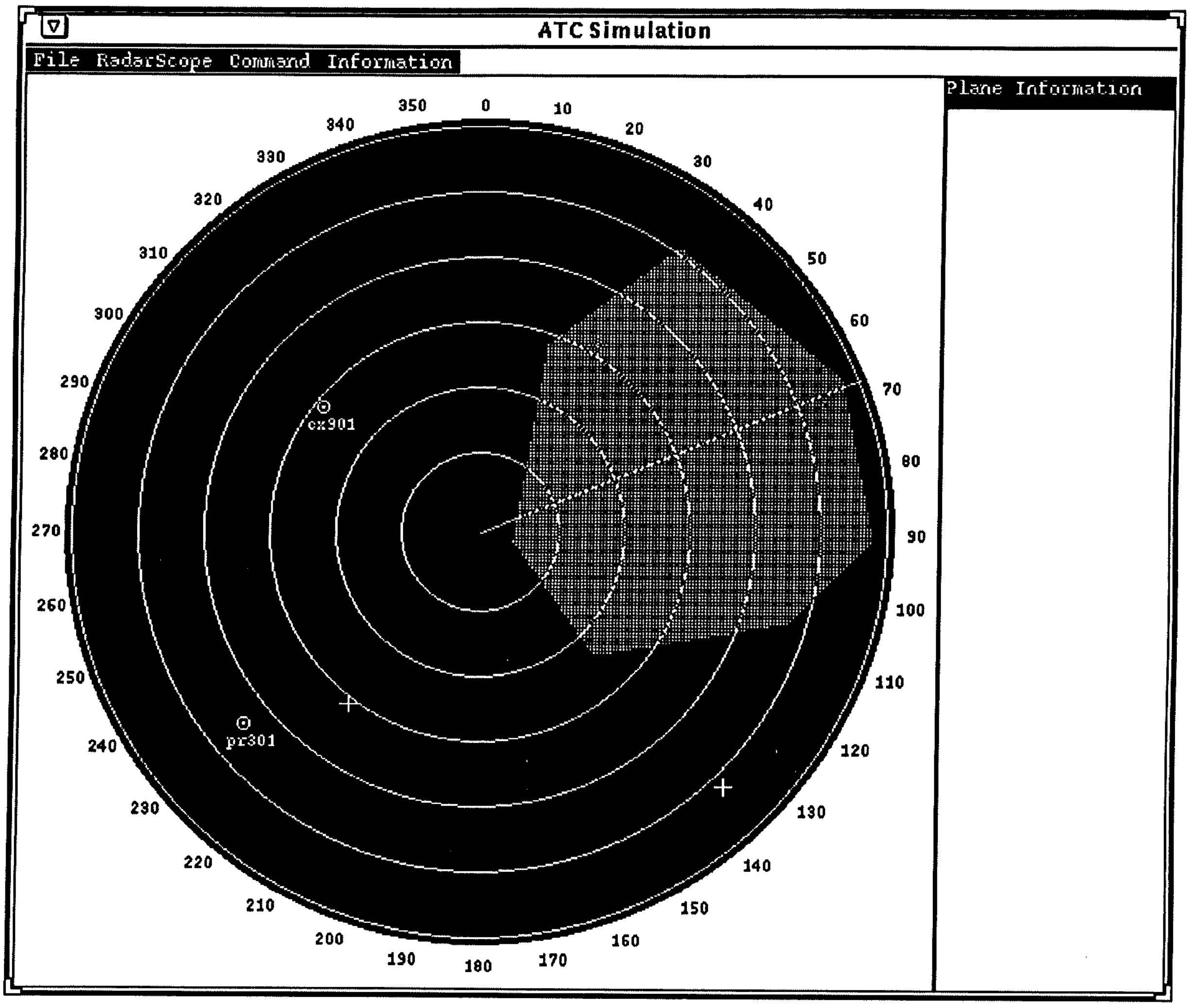
[16] Appendix C contains the release notes about the interface mechanism used in this project. It provides information on where this mechanism (called Mertz-Ong-Nerb-Gary Socket Utility [MONGSU]) is available, how to use it, and other stuff.

[17] Garnet objects and gadgets are tools provided by Garnet in creating simulations with user interface. The Garnet manual by Myers et al. (1993) gives a complete description what these are and many more.

create routines that make up the ATC display and the simulation itself. For example, aggregadgets (consisting of different objects put into one) were created for prototype objects of a radar scope, a plane, and a beacon. Garnet functions were then defined to create components such as the menubar (including functions for the menubar selection), the radar screen, the radar sweep, and others[18] that make up the ATC display window (i.e., the console that displays all these components).

Figure 7 shows what the ATC window looks like. It consists of a menubar and a radar-scope that displays planes, beacons, and weather fragments all created using Garnet objects and tools. It is through this window that human subjects interact with the simulation by using the mouse to execute their desired action (e.g., show beacons). The menubar at the top, allows the user to load a file (the plane database file), quit the simulation, display beacons, weather fragments and other selections needed in an ATC simulation. At the moment, the beacons, weather fragments, and sweep are just there to provide a real ATC-look, however, future extensions could make use of these objects as obstacles for landing planes and other tasks. Since this is just a simple simulation, this is just enough for our purpose. Other menu selections are provided but contain no functions. These missing functions are provided because they will be needed in future ATC simulations.

[18]Appendix E.1 gives a detailed run through of the ATC-Garnet code. It summarizes how the code was developed through the structure of the code.



Legend:

⊙ — Plane

+ — Beacon

⬢ — Weather fragments

Figure 7. Nott-Air Traffic Control (ATC) simulation display window.

To illustrate, each selection in the menubar have corresponding function calls. These functions include **load-file**, **quit**, **show-beacon**, **show-weather**, and others implemented in Lisp. As an example, **load-file** when selected calls **read-db-plane-file** and opens a "plane.db" file. It reads the file ("plane.db") containing the information about planes such as flight number, position and altitude and use it to create generic plane prototypes and attached to it an animator interactor that gives the look of real-planes constantly moving across the radar. These plane structures are concatenated into a list and then placed in the global variable *LIST-OF-PLANES*.

The objective of the planes is to move toward the center of the scope, which represents the airport. This movement was made possible by the animator interactor which calls a particular function (that changes the x and y position) every 0.5 second.[19] The change in x and y coordinates is determined by providing utility functions[20] that compute the distance between the center (the airport) and the current position of the plane in x , y coordinates. Using this value and the velocity of the plane (from the db file), the distance travelled by the plane in x and y every 0.5 second is computed. This change in position is then used to determine the new position of the plane.

Aside from changing position and distances, the animator interactor also checks if a plane has reached a specified distance form the airport. Once this distance is reached (say 20 miles), the interactor calls a function that changes the visibility attribute of the plane making it disappear from the radar scope. This approach allows us to show that a particular plane has already landed, which

[19]0.5 second is the update frequency set in the animator interactor in which the function of changing the position of the plane is called.

[20]Utility functions provide functions that support the running of the simulation. It includes function that compute the distance between plane and airport (called calculate-distance-from-airport), and other related functions. The ATC-Garnet source code in Appendix E gives complete details of what there functions are.

is what planes are supposed to do in our simulation. All changes done within the animator interactor as the planes continuously moves are all reflected on the *LIST-OF-PLANES* (such as x and y position, and distance from the airport).

4.2 Lisp eye/hand code

The Lisp eye/hand code supports the functionalities required in doing the simple ATC task. It provides functions related to viewing and manipulating the simulation such as looking through the scope and landing a plane. The main routine defined there (in "ATC-fn.lisp") runs the Lisp socket code and the simulation, and creates a parallel process that always accepts a client (if there isn't one) and reads in data from the socket stream. The data are list structures that are parsed by Lisp functions to extract command names, identification, and arguments passed in by the Soar model. In a similar manner, outgoing data resulting from the command received by the simulation are constructed into lists and passed back to Soar. Right now, the functions only include **send-scope-info**, **lower-plane-altitude**, and **land-the-plane**.

The heart of this code was to create a process that does continuous Soar interaction. This was made possible by writing a main routine that creates a socket, runs the simulation and creates a separate Lisp process (called **soar-read-loop-process**) that reads incoming data (command, id and arguments) continuously. For example, if incoming data looks something like:

(socketout-link (name send-scope-info) (id <id>)).

The above list is parsed, the command name and unique identification id extracted (arguments are also extracted if they exist). This would result to the equivalent function, in this case, **send-scope-info** (i.e., the function tied to command name '***send-scope-info***') to be called. The function

send-scope-info gets the list of planes and their attributes from the global variable *LIST-OF-PLANES* (a Garnet structure) and converts it into a list to be passed back to Soar.[21] In addition, the simulation also sends the list (done-external-operator <id>) to let Soar know which command it issued has been done by Lisp. This scheme prevents conflicts from happening if multiple commands are sent to the simulation at the same time for there's no other way of knowing which commands got done and which are still pending.

The process of reading and evaluating data from the socket stream is a continuous process. This process is only terminated when (a) the client is no longer connected, (b) a shutdown-socket-stream command is issued in the Garnet prompt, or (c) in the worst case, the Garnet simulation crashes.

4.3 The Soar Model

Turning to the right side of Figure 6, we see the Soar model and the Soar C eye-code. The Soar model performs the basic functions that a real human ATC does (only simplified), such as looking at the scope and issuing command instructions. In addition, it uses Soar's default rules[22] (or knowledge) as part of its rule count (a total of 139) which are loaded together with the ATC Soar model (this will prove to be important later). The basic algorithm used in this model is shown in

[21]'***send-scope-info***' is issued with no arguments as it only looks for planes in the scope. Other commands like '***lower-plane-altitude***' and '***land-the-plane***' are issued with a flight number as argument (for the plane they want to lower the altitude or land). The functions for performing these commands changes the attributes of the plane they are addressing the command to which are then updated in the *LIST-OF-PLANES*.

[22]Chapter 7 of the Soar manual Version 6 (Laird et al., 1993) discusses how default knowledge (or rules) influences Soar's performance of a task. These default rules number approximately 107 and are contained in a file "default.soar".

Table 5 below (details of which are discussed when we examine a sample run). It must be clear in here that Soar is not a procedural language, as such, the ATC Soar model itself is not procedural. We only provide an approximate procedural representation (or algorithm) of the Soar model to explain how the model works.

```
get-planes: look-scope
IF first look THEN set first-look flag and continue
do
[       set looked and read flag and make plane-info current
        propose operator for each plane
        compare operators
        focus on most important
        IF there is something to do THEN do it: that is, apply operator
        ELSE reset looked flag and
                propose operator look-scope
]
until no more planes that are supposed to land or
        no more existing planes
```

Table 5. Algorithm of the Soar model.

The above algorithm works as follows. On the installation of the top context (i.e., installation of goal, problem space, desired state, and initial states necessary in programming Soar[23]), operator '**look-scope**' gets proposed and implemented. This operator implementation results to the command '***send-scope-info***' to be sent to *socketout-link* together with arguments (if any) and a unique identifier.[24] The *first-look* flag is set to take note of the very first implementation of '**look-scope**' (since there are still no data from the simulation at that point).

[23]The Soar manual by Laird et al. (1993) gives a detailed description of the rules and procedures in writing Soar productions.

[24]Any command issued has an accompanying unique identifier that tells Soar if the command it issued is already done or not (as Lisp returns the command it does through its identifier).

Subsequent implementation of '**look-scope**' ignores this flag (and the production that takes note of this) having been set and just proceeds to the issuance of the command itself.

The implementation of any external operator always creates an operator no-change (OP NC) impasse because the operator has not yet received data from the simulation. During this impasse, an operator called '**watching-command-process**' (similar to the default '**wait**' operator in Soar) is selected and implemented several times until Soar has received a command confirmation (i.e., the list (done-external-operator <id>)). After receiving this information, the external operator terminates.

After the plane information has been read (the data returned from the simulation through '**look-scope**') and put into the top-state, a series of operator proposals and implementation occurs depending on the state of the selected plane and its attributes. Below is a list of conditions that lead to particular operators being selected for each plane.

[a] if (^control[25] under) and (^status[26] land)

then operator '**there-are-planes**' gets proposed since the particular plane is in the process of landing.

This operator when implemented sets the *looked* flag to NO and the model looks again.

The above operator when simultaneously proposed with '**no-more-planes-to-land**' is always preferred. This ensures that the model does not terminate if there are still planes in the process of landing.

[25]This attribute determines what particular stage is a plane in. Its value is either *under* (under our control), *descend* (has already descended), *handed* (handed to the approach controller), or *out* (of our control).

[26]This attribute determines if a plane is going to land (which we need to take care of) or is just doing an over flight (which we do not care at the moment). Its value is either *land* or *over*.

[b] if (^control under), (^status land) and plane (^distance <= 200)

then operator '**lower-altitude**' is proposed since the plane has reached the first approach fix.

If several planes satisfy the above condition, an operator is proposed for each plane and the plane with the smallest distance from the airport is preferred. This operator when implemented issues the command '***lower-plane-altitude***' to the simulation that lowers the *altitude* of the plane to 5000 meters and change the *control* to descend.

[c] if (^control under), (^status land) and plane (^distance <= 80)

then operator '**land-plane**' is propose since the plane has reached the final approach fix and is ready to land.

If several planes satisfy this condition, an operator is proposed for each plane and the plane with the smallest distance from the airport gets priority. This operator when implemented issues the command '***land-the-plane***' and hands the control over to the approach controller.

If operators '**lower-altitude**' and '**land-plane**' are proposed at the same time, '**land-plane**' is preferred because it is more important to take care of landing planes (high risk) than lowering the altitude of a plane that is still a distance away.

[d] if plane has (^control under), and (^status over)

then operator '**no-more-planes-to-land**' which has three variants gets proposed. The three variants allow the operator to be proposed (a) if the plane has already landed and handed over to the approach controller, (b) if the plane is doing an overflight, or (c) no more planes exist in the radar scope.

This operator when implemented sets the state to the desired state (i.e., ^planes-to-land none), which subsequently leads to termination.

Lastly, if the preferred operator in conditions [a] and [b] (which is always '**there-are-planes**') gets proposed at the same time with the preferred operator between [c] and [d], the chosen operator between [c] and [d] is always preferred. Otherwise (with either no more planes to land or planes in intermediate stages) ATC-Soar just does nothing and looks again. All commands issued for all conditions except condition [a] always proposes to look again after the command is performed by setting a *looked* flag to NO that enables operator '**look-scope**' to be proposed again. This process repeats until the desired state is achieved – there are no more planes to land.

Figure 8 and 9 shows two structural diagrams of the ATC Soar model. The main distinction is the level of detail they describe the inner workings of the ATC Soar model. Figure 8 illustrates a minimal view of the model, which is what Soar experts normally create. This diagram is quite vague especially for novices in Soar (and those who knows nothing about Soar) as there is no clear illustration of the flow of how the model works. Figure 9 shows a more detailed description of how the Soar model works. Through this illustration, it was able to show in greater details how the model runs, which for novices provides a lot of help in understanding it. But to consistently provide this kind of detail to all Soar models entails a lot of work, which for Soar experts is a bit too much and a mess to create especially for large systems. Since there are no standards in creating such diagrams for Soar models, its creation will just be based on the model's size and the level of detail that the modeller wants its readers (and himself) to know.

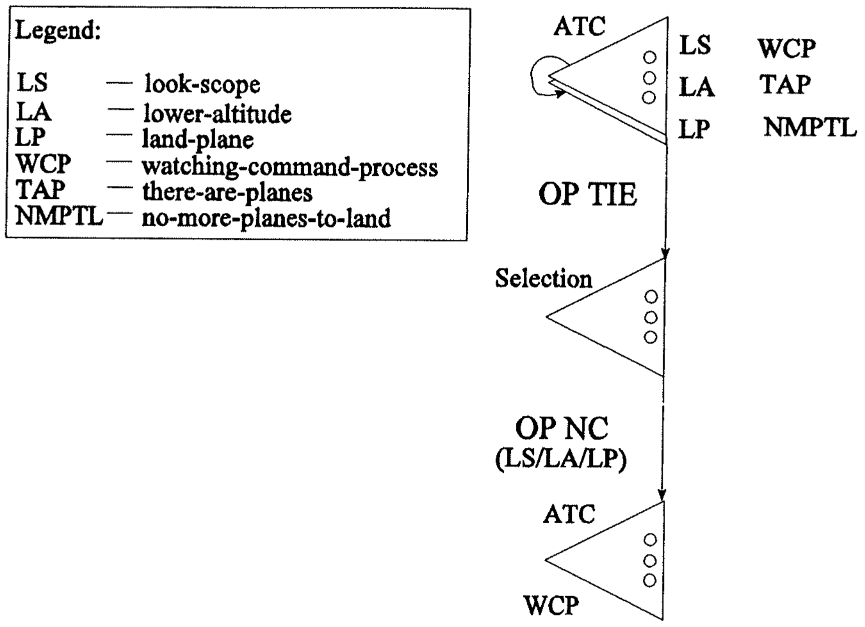


Figure 8. Minimal view of the ATC Soar model.

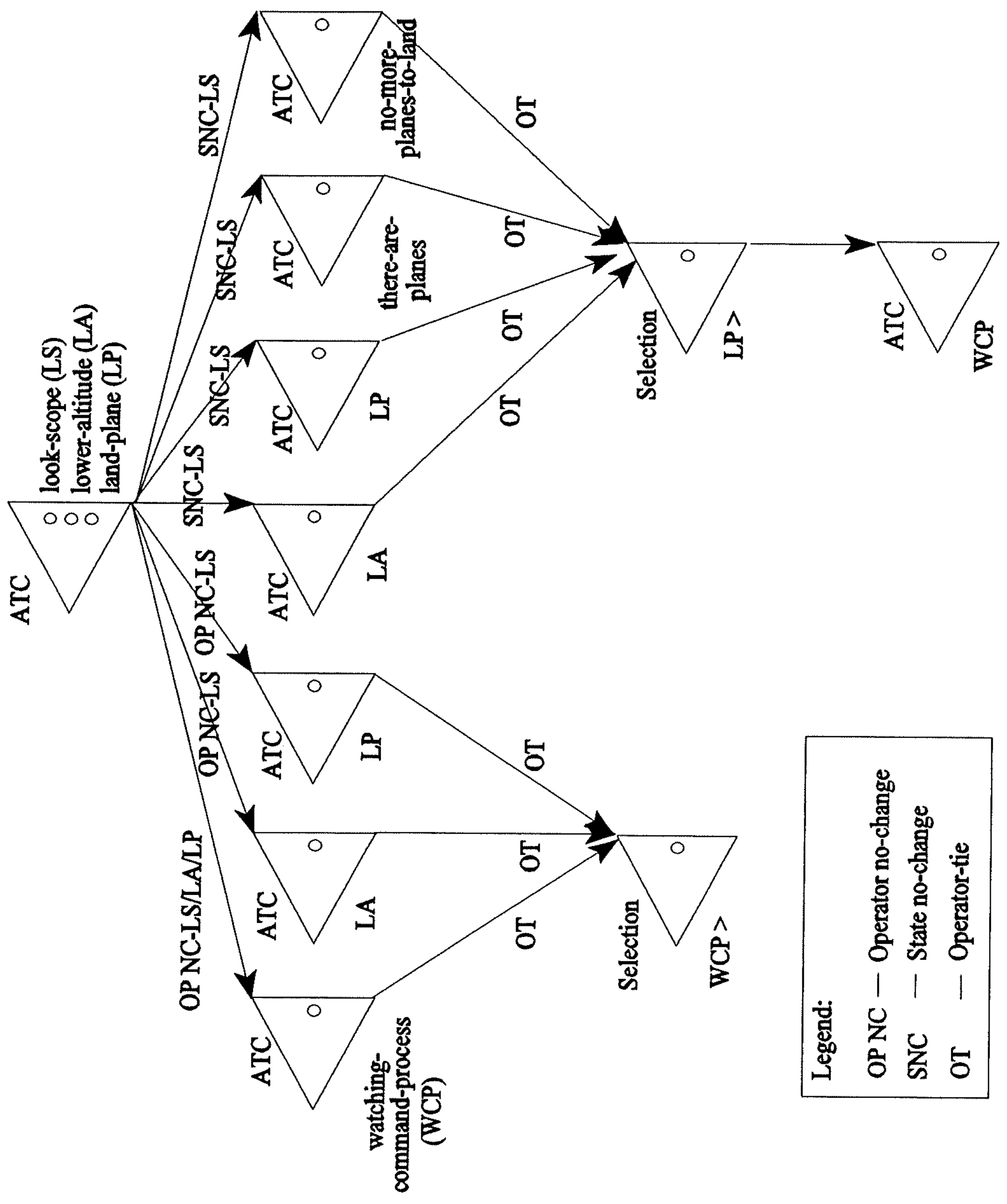


Figure 9. Structural diagram of the ATC Soar model.

4.4 Soar C eye-code

One box below the ATC Soar model is the Soar C eye-code (which is actually part of the socket code) that does the job of sending data (such as commands, ids and arguments) as a list from the output link down the socket to the simulation. This C-code does the parsing procedure[27] automatically for incoming and outgoing data from Soar through *socketout-link*. The parsing procedure works by reading each incoming character and categorised if it is part of a symbol, integer, float, constant, or a substructure (nested list). After categorisation, the extracted symbols, integers, float-constants and/or substructure are deposited as WMEs in the top-state. At the other end, any changes done to *socketout-link* results in the value of *socketout-link* (and its substructure) to be formed into a list (using the same parser above) and then sent down the socket to the simulation. This procedure is done continuously until such time that the goal (i.e., in our case, to land all planes until there is none) is achieved or the link between the simulation and model is terminated.

4.5 UNIX Sockets

The backbone of this entire system is the communication channel implemented in UNIX sockets.[28] This interface mechanism is divided into the standard socket code, the Lisp socket code, and the Soar socket code. The socket code was originally written by Joseph Mertz (1994) and was modified, generalized and adapted by the author for use in and as part of this project. Table 6 lists the modifications made to the initial socket code.

[27]We only try to give an overview of how the Soar parser (written in C) works. For details of the parsing procedure, please refer to Appendix G.2 ("soar-socket.c").

[28]Appendix G provides the entire socket code (i.e., Mertz-Ong-Nerb-Gary Socket Utility [MONGSU]) in Soar and Lisp. This socket code is originally created by Joseph Mertz and Gary Pelton, and was modified, generalized and adapted by the author and Josef Nerb, hence the name MONGSU.

Table 6. Modifications to Mertz' socket code.

▸ Added global variables to make functions more readable, understandable and general.
▸ Modified error messages to make them more clear and constructive.
▸ Documented statements for use in launching a process in Lisp.
▸ Modified some functions for use in launching a process in Lisp.
▸ Modified global variables of Soar to make the code compatible with Soar 6.2.3.
▸ Modified existing command line commands (written by Mertz) in Soar to make it act as a server.
▸ Added new command line commands in Soar to make it act as a client.
▸ Modified and added help for the new commands in Soar.
▸ Added additional functions in Lisp for it to launch a soar-read-loop-process.
▸ Added miscellaneous functions for testing the code before each release.
▸ Created release notes for the code.

The whole idea behind these different parts of the socket source code is to implement the communication protocol shown in Figure 10. The different parts of the entire socket code comprise a portion to implement this protocol.

4.5.1 Standard Socket Code

The standard socket code ("std-soar-socket.c") defines the basic functions that support socket communication. Functions like creating a socket (create_socket), binding (bind_socket), listening (listen_socket), accepting (accept_socket) and connecting (connect_socket) are all defined as separate routines to be used by any programming language that wants to use sockets (as long as they can declare these functions as foreign functions like Lucid Lisp does). These functions are used as standard functions for use in Lisp and Soar.

4.5.2 Lisp Socket Code

The Lisp socket code ("stdio.lisp" and "socket.lisp") provides low-level support in Lisp (in this case Lucid). It uses the previous standard socket code (defined as foreign functions) to create a function **make-socket-stream** that incorporates all necessary functions in order for Lisp (and the Garnet simulation) to act as a server. These functions include the creation of a raw socket, binding the socket and listening for incoming requests for a connection. Moreover, these functions contains error checking facilities that doubly act as debugging tools, and testing facilities that can test your Lisp and Soar code as either client or server.[29]

4.5.3 Soar Socket Code

The Soar socket code uses the same standard socket functions as does the Lisp socket code. In addition, it creates new Soar commands (i.e., command-line commands) that uses these functions to support socket communication. These new commands include '***init-socket-io***', '***init-socket-server***', '***close-socket-io***', '***shutdown-socket-io***' and '***socket-output-link***', all of which are defined in the Soar socket code ("soar-socket.c"). These new Soar commands are accessible in Soar through the command line (as they are compiled together with Soar) and through the on-line help.

[29]Appendix C shows the release notes for the socket code (MONGSU, 1994). It describes procedures on how to obtain, install, use and test the code.

4.5.4 Soar-Simulation Communication Protocol

In our Soar-Garnet system, the simulation was made the server to service client Soar models. Normally, one might make Soar models act as the server, not in this case however. The following grounds support our choice, (a) in a real ATC task, the task itself is continuous while the human subject (as the Soar model) interacts only when it wants to, (b) if the Soar model was made the server, there could be a huge waste of time waiting for the simulation to connect because the model can not run without a connection, which is not at all practical if you think of the way a real ATC works, (c) the model is the more naturally viewed as a user of the simulation and not the other way around, and (d) Soar can be easily turn on and off, while the simulation takes a lot of time to come up.

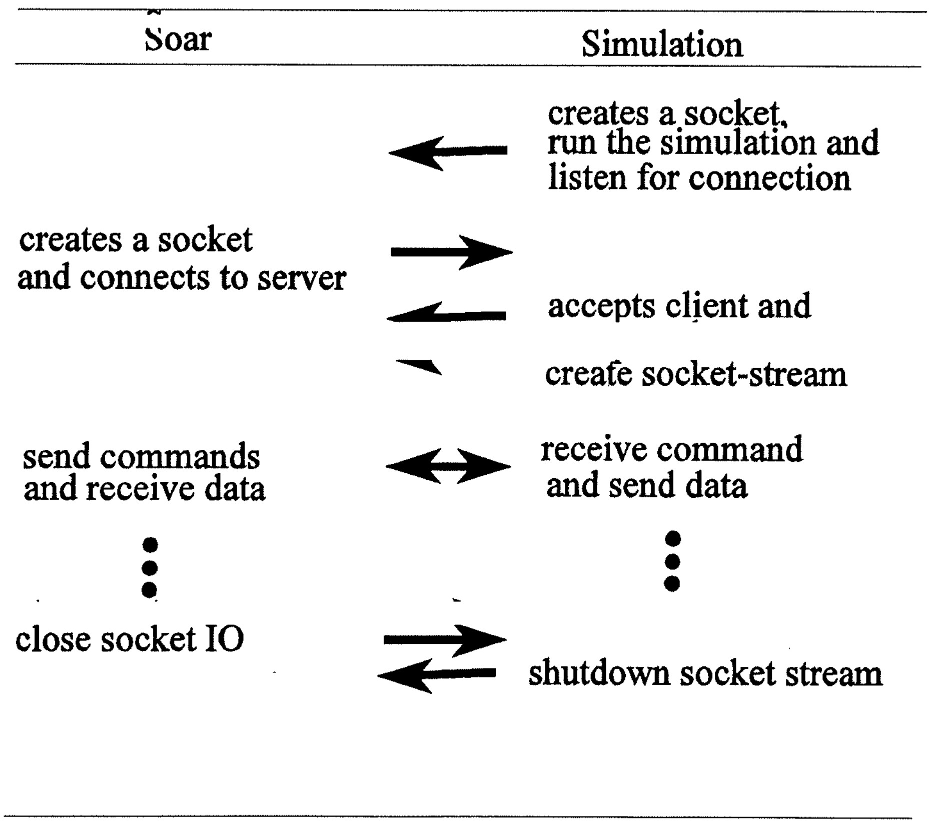

Soar
Simulation
creates a socket, run the simulation and listen for connection
creates a socket and connects to server
accepts client and
creafe socket-stream
send commands and receive data
receive command and send data
close socket IO
shutdown socket stream

Figure 10. Steps in initiating and terminating the Soar-Simulation communication protocol.

Figure 10 shows the protocol followed by the Soar-Garnet system during a normal Soar-simulation communication.[30] As can be seen, when the simulation starts running it does the initializations, the creation of a socket and a process that continuously listens (i.e., in a loop) for incoming requests for connection (in this case only one connection is allowed at a time) to be accepted, subsequently creating a socket stream (from which all data are sent and received). At the moment, only one client is allowed to connect at a time. If multiple clients are desired such as the system illustrated in Figure 11, all that is needed is to change the DEFAULT_QUEUE_LENGTH[31] to the number of connections desired.

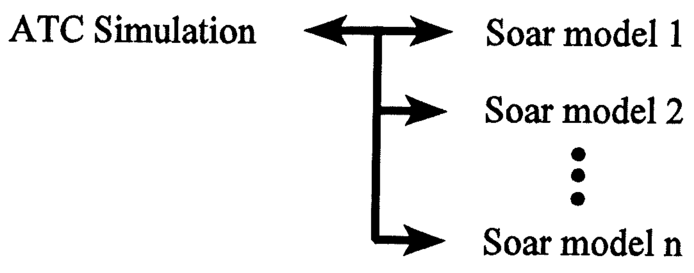


Figure 11. Multiple Soar-Garnet system.

Once the simulation has done his part of the communication interface, Soar then creates a socket and connects to the server (the simulation). After the connection is successful, conversation between the model and the simulation starts and goes on until the goal is achieved. It must be noted that Soar always start the talking and the simulation just respond and goes on with whatever it is doing (moving the planes continuously). If the goal is achieved, the process can be repeated again, or the link can be closed (if preferred).

[30]The subject can interact with the simulation without the need of a protocol.

[31]This variable defined in "std-soar-socket.c" defines the number of clients that are allowed to connect to the server.

5. A SAMPLE RUN

To illustrate the implemented system, we take an example of landing two planes. The objective is to land two planes shown in Table 7. There are actually two traces of sample runs in existence, one with an error (Appendix D.1) and the other with the fix in place (Appendix D.2).[32] For reasons which shall be clear later, we will discuss the modified trace of the sample run in Appendix D.1 (with error) which is illustrated in Figure 12. Concluding this chapter would be the discussion on the fix that corrected the error.

Flight Number	x position	y position	Altitude	Velocity	Control	Status
CX901	100	150	20000	4	under	land
PR301	450	220	25000	3	under	land

Table 7. Database of planes for sample run.

The trace begins with the creation of the context-slots containing the goal G1, problem-space P1 (ATC) and state S1 during the first two decision cycles. The desired state is to land planes until (^planes-to-land none). This desired state is initially set to YES at the creation of the top-state, which also includes (^first-look no), (^looked no), and (^current-planes none). On the third decision cycle, operator '**look-scope**' gets proposed, and a *first-look* flag is used to take note of its first implementation. This implementation then results to the list (socketout-link (id <constant-symbol>) (name send-scope-info)) to be sent to the simulation. Subsequent implementation of '**look-scope**' will ignore the *first-look* flag (and the production that does this), that is, it wouldn't fire anymore and just proceed accordingly.

[32] "ATC.soar" contains the bug-free source code for the Soar model with the fix highlighted within.

```
      0: ==>G: G1
      1:     P: P1 (atc)
      2:     S: S1
      3:     O: O1 (look-scope)
 write successful socket 3

      4:     ==>G: G2 (operator no-change)
      5:         P: P1 (atc)
      6:         S: S1
      7:         O: O2 (watching-command-process)
    199:         O: O2 (watching-command-process)
    200:     O: O7 (lower-altitude)
   Flight number pr301 has reached First approach fix.
   Descending to 5000 meters.
 write successful socket 3

    201:     ==>G: G3 (operator no-change)
    202:         P: P1 (atc)
    203:         S: S1
    204:         O: O10 (watching-command-process)
    219:         O: O10 (watching-command-process)
    220:     O: O9 (look-scope)
 write successful socket 3

    221:     ==>G: G4 (operator no-change)
    222:         P: P1 (atc)
    223:         S: S1
    224:         ==>G: G5 (operator tie)
    225:             P: P4 (selection)
    226:             S: S2
    227:             O: O19 (evaluate-object O15 (lower-altitude))
    228:             ==>G: G6 (operator no-change)
    229:                 P: P1 (atc)
    230:                 S: D3
    231:                 O: C4 (lower-altitude)
   Flight number pr301 has reached First approach fix.
   Descending to 5000 meters.
    232:                 ==>G: G7 (operator no-change)
    233:                     P: P1 (atc)
    234:                     S: D3
    235:                     O: O23 (watching-command-process)
    241:                     O: O23 (watching-command-process)
    242:     O: O25 (there-are-planes)
   There are planes to land.
    243:     O: O28 (look-scope)
 write successful socket 3
```

Figure 12. Modified trace of sample run in Appendix D.1.

5.1 The IO Pattern

During the fourth decision cycle, an operator no-change impasse occurs, resulting for a subgoal G2 to be created with the same problem-space P1 and state S1 as the top-goal G1. This impasse occurs because operator '**look-scope**' has not yet terminated (i.e., no input-link has been received and no confirmation from Lisp that the command has been done) resulting to '**watching-command-process**' to be selected a number of times.[33] These so-called wait operators ('**watching-command-process** operators selected several times') are added because the IPC and the simulation does not respond as quickly as a Soar decision cycle. Soar must wait for a response.

The selection and subsequent implementation of '**watching-command-process**' would continue to persist until such time that an input link (i.e., the list of plane information) and a confirmation list (done-external-operator <id>) gets passed back from Lisp and deposited on the top-state. Consequently, operator '**look-scope**' will be terminated and the plane information put in *current-planes*. It must be noted here that this pattern of having an external operator implemented,[34] a no-change impasse occurring on a OP NC subgoal, then a series of operators, checks to see that the external operator has been terminated. This pattern often occurs throughout the whole trace (and hence will be referred as '**the IO Pattern**'). In addition, operator '**look-scope**' will always be proposed and implemented if no other operator gets selected. Operator

[33]In the modified trace in Figure 12, the series of '**watching-command-process**' operators are cut down leaving just the start and ending implementation (in terms of decision cycles). To count the number of decision cycles spent waiting, subtract the decision cycle number of the start from the end.

[34]These external operators have a distinct attribute ^*external* set to YES and they include '**look-scope**', '**lower-altitude**' and '**land-plane**'.

'**look-scope**' is the default operator that gets implemented when no other operator external or otherwise is implemented. This is achieved by setting *looked* flag to NO after every operator implementation, which then triggers the proposal of '**look-scope**' and subsequent implementation. The operator implementation of '**look-scope**' resets back *looked* to YES.

After receiving data from the simulation sometime later (after 200 DCs), operator '**look-scope**' terminates. On decision cycle 200, operator '**lower-altitude**' gets proposed and implemented on plane with flight number PR301 (i.e., it has reached the first approach fix, thus the printed message). This operator was proposed because PR301 has (^status land), (^control under), and plane (^distance <= 200). The pattern of having an operator no-change impasse occurs again similar to the IO Pattern until such time that '**lower-altitude**' is terminated. If it happens that multiple planes satisfy the conditions for proposing '**lower-altitude**' (same with '**land-plane**', '**there-are-planes**' and '**no-more-planes-to-land1[2 and 3]**'), operator '**lower-altitude**' will be proposed for each plane and the plane with the smallest distance from the airport (i.e., attribute *^distance*) is chosen and the operator '**lower-altitude**'[35] is applied.

5.2 The Daydream Pattern

An interesting thing happens after the implementation of '**lower-altitude**', followed by '**look-scope**' is the occurrence of an operator tie impasse in subgoal G5. We didn't actually know why this happened initially, but through rigorous tracing we were able to find out. It seems that this impasse occurred because, after the implementation of '**lower-altitude**' (or the other external

[35] A detailed description of how the ATC Soar model works in general is given in Section 4.2.

operator '**land-plane**'), the data (i.e., the *^altitude* and *^control)* modified by the simulation as a result of the issued command (in this case, '**lower-plane-altitude**') can not yet be seen until after '**look-scope**' is implemented and the new input-link received. As a result, after '**look-scope**' was implemented, '**lower-altitude**' is again proposed (because the conditions are met as the old input-link is still considered current) together with the '**watching-command-process**' operator resulting in the operator tie impasse. A random selection between these two operators (a default Soar approach to resolving a tie impasse) allows '**lower-altitude**' to be selected and evaluated resulting to another operator no-change impasse on subgoal G6. The default response[36] to this OP NC impasse is to recreate the context that led to the tie, and select the object being evaluated ('**lower-altitude**'). This object evaluation then leads to another operator no-change impasse presumably after writing on the top state for IO, but in this case to the local state resulting to a series of '**watching-command-process**' operators to be implemented (as this is the only applicable operator left). These operators from the changing world persist until the new input-link and command confirmation is received by Soar that terminates the entire series of subgoals because it makes '**look-scope**' most preferred (this pattern will now be referred to as '**the Daydream Pattern**'[37] to distinguish it from '**the IO Pattern**'). On the next decision cycle (decision cycle 242), operator '**there-are-planes**' gets implemented that signals that there are still planes in the process of landing (and are intermediate stages in their landing process) and the cycle of looking should continue. As a result of this, '**look-scope**' is implemented and the IO Pattern is again manifested.[38]

[36]Section 7.3.2 of the Soar manual Version 6 (Laird et al, 1993) discusses the evaluation subgoal in relation to the evaluate-object operator.

[37]This Daydream Pattern does not occur in Appendix D.2 (because the fix is already in place), instead only the IO Pattern is manifested all throughout the trace which is correct.

[38]For every implementation of '**look-scope**' except if preceded by other external operators ('**lower-altitude**' and '**land-plane**') will result to the IO Pattern to be manifested.

This entire process of happenings (which is an error) sort of tells something about the inner workings of Soar. This was a very welcomed (benign) error because it told us that default rules in Soar were getting more robust (allowing it to run), which traditionally would make it crash upon stumbling to an error. Furthermore, on a theoretical point of view, it seems like '**look-scope**' was looking in some uncharted, imaginary space during the series of impasses in the Daydream Pattern that was brought back to reality when '**there-are-planes**' was implemented. For a time, it looks the model was imagining (or daydreaming) and then something in the outside world brought its attention back. If the occurrence of this Daydream Pattern is further investigated and examined, it would certainly provide an interesting hypothesis on how we could create and implement imagination in Soar.

5.3 The Rest[39]

The whole cycle of implementing '**look-scope**' when planes are in intermediate stages of the landing process would always result to '**there-are-planes**' to be implemented. This would constantly remind the model the there are still existing planes in the process of landing. Noticeably, the rest of the trace continues in a manner of implementing the following scenarios, (a) the implementation of '**look-scope**' and '**there-are-planes**', (b) the implementation of either '**lower-altitude**' or '**land-plane**', and (c) the occurrences of the IO pattern and the Daydream Pattern. So, after flight number PR301 has descended, CX901 followed suit. A series of '**look-scope**' and '**there-are-planes**' gets implemented, then operator '**land-plane**' is implemented on PR301 (i.e., it has reached the final approach fix) with (^status land) (^control descend) and plane

[39]Please refer to Appendix D.1 for the rest of the sample run.

(^distance >= 80). After a short while, CX901 also reached the final approach fix. And one final '**look-scope**' suggests that both PR301 and CX901 have already been cleared for final approach and have been handed over to the approach controller, resulting to '**no-more-planes2**' (one of the variants of '**no-ore-planes-to-land**')to be proposed and implemented. The implementation of '**no-more-planes-to-land2**' results to *planes-to-land* to be set to NONE, thereby achieving the desired state, and subsequently terminating the Soar program.

5.4 The Fix

In considering the error found in the sample run in Appendix D.1, a fix was added to the Soar model that resulted to the correct output run on Appendix D.2. The fix was to add a production rule which states that operator '**watching-command-process**' is always better than any other operator proposed simultaneously. This rule ensures that a new input link containing the modified data will always be received first before other external operators can be proposed. As a consequence, the Daydream Pattern is now removed retaining just the IO Pattern.

6. ANALYSIS OF THE MODEL DURING A SAMPLE RUN

Taking the sample run in Appendix D.2 (the error-free sample run), we analyse how the Soar model performs its ATC functions. In particular, we are interested how much time was spent in performing the external operators '**look-scope**', '**lower-altitude**' and '**land-plane**'. In addition, we also want to determine the time spent waiting (i.e., operator '**watching-command-process**'). In doing this, we implement two methods of data analysis from the sample run. The *choices* analysis and *time-in-slot* analysis.

These two methods are just two of many several methods that could be use to analyze Soar data. They were chosen to interpret data and compare their analyses to determine which tells how Soar works to a high level of degree in accuracy. In fact, there are actually no right or wrong methods because each method interprets the data differently and tells a different story of what is going on. Therefore, the choice of methods to be used is discretionary. Furthermore, it must be noted that the length of the sample run in terms of the number of decision cycles can be different even though the same number of planes and plane information are being used. This is due to the variability of the time the simulation takes to interact with the model, but a typical number is about 1000 decision cycles. Thus, the results presented in these analyses are just approximate of how the model behaves for this type of data.

6.1 Choices Analysis

In *choices* analysis, we count the number of operator applications of an external operator and aggregate them. This allows each choice (i.e., each operator implementation) to be counted as a distinct choice. Table 8 shows the numbers counted for the sample run in appendix D.2 and their percentages based on the total number of decision cycles – 1056.

Operator	No. of Operator Applications	Percentage (%)
look-scope	35	3.3
lower-altitude	2	0.2
land-plane	2	0.2
watching	867	82.1
miscellaneous	119	11.3
non-op selections	31	2.9
Total	1056	100

Table 8. Choices analysis.

In Table 8 are the numbers for '**look-scope**', '**lower-altitude**', '**land-plane**', '**watching-command-process**', miscellaneous and non-operator selections. The miscellaneous category covers all internal operators (e.g., '**there-are-planes**'), while non-op selections includes the creation of goals, problem-spaces and states. We are only interested in the application of external operators that performs the ATC task and therefore do not care about these miscellaneous operators and non-operator selections except for their contribution to the total number of decision cycles in the sample run.

As seen from Table 8, operator '**watching-command-process**' took the bulk of the total number of operator applications (a total of 867) compared to the external operators which is just a mere total of 39. These numbers results to percentages which when translated to a pie graph (seen in Figure 13) prints a good picture of how the model works. Figure 13 illustrates these percentages.

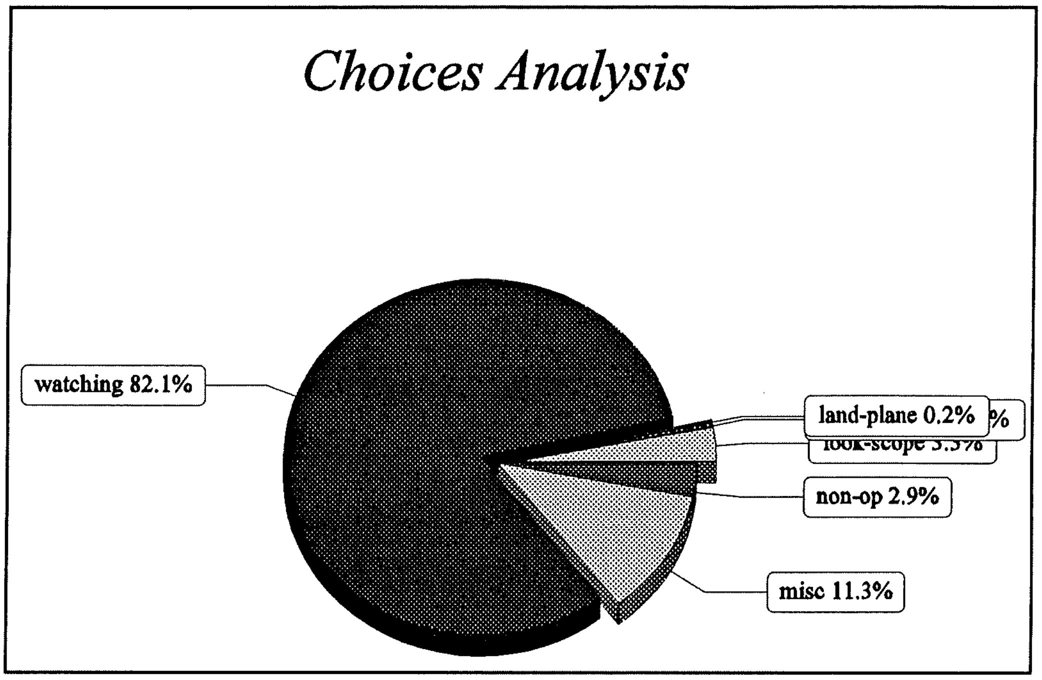


Figure 13. Pie graph of *choices* analysis.

Referring to the graph above, 82.1% of the time was spent on watching a command to be process and a negligible 3.3% for '**look-scope**' (and 0.2% each for '**lower-altitude**' and '**land-plane**'). These figures suggest that human subjects also spend approximately 82% percent of their time watching for something to be processed (i.e., wait) while the direct issuance of instructions or commands just takes a negligible amount. This is absurd of course, because a lot of the time spent watching for a command to be processed was brought about by looking at the scope (because there will be no watching if there is no looking or no command was issued). So future analysis should examine higher level operations that lead to the external command.

The analysis of having the watching be tied up to an external operator is not illustrated here making the *choices* analysis to be a doubtful data interpreter (for this ATC task). A more accurate method (as we shall soon see) would be the *time-in-slot* analysis.

6.2 Time-in-slot Analysis

In considering the *time-in-slot* method, we consider the length of time that an external operator is performed. This means we count the number of decision cycles that the operator was in the operator slot (i.e., implemented) to the time it got removed (i.e., terminated), which may include operators to be implemented within this time. Now, taking another look at the sample run in Appendix D.2, we could see that during the implementation of an external operator, '**watching-command-process**' gets implemented and terminated in series as a sub-operator to its higher-level operator (e.g., '**look-scope**'). Therefore, the number of decision cycles that took '**watching-command-process**' to be implemented must be added as part of '**look-scope**' (or other external operators) and not counted on its own. This is because the time that '**look-scope**' was in the operator slot included the amount of time for '**watching-command-process**' to be implemented and terminated in succession. Table 9 shows the computed number of decision cycles for each operator (inclusive of the '**watching-command-process**' embedded as a sub-implementation) and their corresponding percentages to the total of 1056.

Operator	No. of Decision Cycles	Percentage (%)
look-scope	837	79.3
lower-altitude	35	3.3
land-plane	34	3.2
miscellaneous	119	11.3
non-op selections	31	2.9
Total	1056	100

Table 9. Time-in-slot analysis.

The computation of the numbers in Table 9 is a straightforward formula of adding the number of decision cycles that an operator was implemented (normally 1) and the number of sub-operators implemented within this implementation. Taking the sample run as an example, we first take the number of decision cycles to implement '**look-scope**' which is equal to 1 (decision cycle 3) and add the number of decision cycles that '**watching-command-process**' was implemented as a result of implementing '**look-scope**' which is 169 (decision cycle 7 to 175). This total added with the other implementation of '**look-scope**' gives a value of 837 decision cycles.[40] The same thing is done for '**lower-altitude**' and '**land-plane**'. Figure 14 translates Table 9 to a pie graph.

[40]As with the *choices* analysis, we only consider external operators and 'watching-command-process' in computing the number of decision cycles that takes an external operator to be implemented.

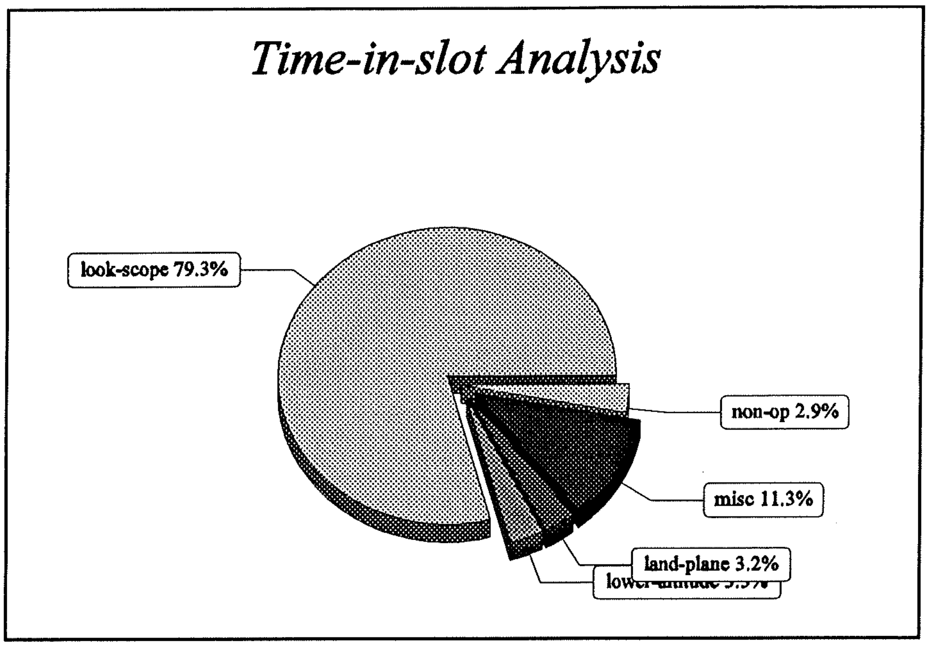


Figure 14. Pie graph of *time-in-slot* analysis.

It is apparent in Figure 14 (and for this analysis) that the time spent in watching a command to be processed was the consequence of waiting for external operators to be implemented. It is also clear that a lot of time (79.3% to be exact) was spent looking at the scope and monitoring the incoming planes. This interpretation gives a more accurate and precise picture of how the ATC task is performed by human subjects. Clearly, a human controller spends a lot of time looking at the scope and included in this time is the amount of time he needs to wait for data (e.g., through perception or serially examining physical data such as flight strips).

Now we can safely say that the *time-in-slot* method is a better measure of how the ATC Soar model works. The reasons are (a) its interpretation of Soar data makes a more believable approximation of its human counterpart, and (b) the operator selections implemented as a consequent of a higher level operator being implemented was considered as part of the higher level operator.

The results of these analyses points to two distinct problems. From a technical point of view, the results imply that the Soar model (and Soar itself) is fast in terms of processing speed compared to the simulation (in Garnet/Lisp) because of the need for '**watching-command-process**' operators to be implemented within the application of an external operator. But in a theoretical point of view, any external operator that is implemented must cover all the watching (or waiting) necessary until that operator is terminated without the need for other operators like '**watching-command-process**' to be implemented. These problems when solved would give a totally new and necessary outlook for the way systems of routinely tying a Soar model (in particular) to interactive simulations will be developed.

7. DEMONSTRATION OF UTILITY AND LESSONS LEARNED

The system created in this project has the unique quality of being able to provide a mechanism (i.e., the Soar-Garnet interface) to routinely tie a Soar model to an interactive simulation (e.g., ATC). This chapter examines how the interface is used by the ATC task and the Tower of Nottingham task, – a complex blocks world task modelled by Josef Nerb (1994). It illustrates how this mechanism satisfies their needs, in what way could it be modified or upgraded to be more useful and functional to their respective task. Also included is the generality that this mechanism provides, so that other tasks would be able to used it with less modifications or better yet none at all.

7.1 Our Simple ATC Task

The ATC task is to create a grossly simplified air traffic control simulation. The simulation would handle planes under its controlled air space, land them (if it wants to land) and hand them over to another controller. The situation here is that the Soar model does the job of handling planes by issuing commands to the simulation and the simulation would respond to these commands appropriately. Furthermore, the Soar model will be simple as well in order to provide a working model that routinely ties the Soar model to the ATC simulation.

In this task, we created a loop function **soar-read-loop-process** that always checks for data from an open socket stream and read that data (if there is). Essentially, there are two processes that need to run in parallel, the simulation and the Soar read-loop process. To go about this, we took the reading process (**soar-read-loop-process**) and made it a background process running in tandem with the Garnet process (the simulation) in the foreground. This approach facilitated the **main-routine** to create a server socket, run the simulation while the **soar-read-loop-process** listens for incoming requests for connection and reads in data from the client (the Soar model) in a loop in itself. In addition, it provided a parsing procedure that works on data structures familiar to both model and simulation – a **list**.

The list data structure used in this interface was just perfect for both Soar and Garnet/Lisp. The simulation developed in Lisp works best with lists, while Soar has working memory elements (WMEs), that is, a hierarchical list of data structures consisting of identifiers, attributes and symbols. Moreover, the data being passed back and forth between the model and the simulation are commands and plane data in list structures, which makes it convenient to have a parsing procedure that is done automatically for incoming and outgoing data to and from Soar.

During testing, it was found out that Soar was fast in processing speed compared to the simulation. Therefore, '**watching-command-process**' operator was added to delay Soar while it waited for data in response to the command issued to the simulation. This approach is not neat, especially if waiting takes a long time where the '**watching-command-process**' operator gets proposed and implemented a number of times. This can lead to a long line of these operators filling up your screen (which is annoying) suggesting that perception in this case takes too long.

Another problem is that data coming from the simulation gets piled up in the top-state as garbage and stacks up Soar's memory (which can cause stack overflow) since a new input link gets added to the top-state everytime incoming data is read. This problem is noticeable if the running model has a lot of interaction with the simulation. Also there's no utility that automatically removes old input-links if a new input-link is added in the top-state. If such a utility exists, then there is no old input-links to worry about only the recent input-link, thus eliminating the problem of stack overflow. In the long run, this capability of being able to remove old input-links must be added to the socket utility (and made automatic) if we want to run Soar for extended periods. Probably, human air traffic controllers just retain the most recent data (or the last 2 or 3), and not the entire data from the start of task (because ATC is a continuous work). Memorization may occur but this would explicitly copy inputs from the top state.

7.2 ToN— A complex blocks world task

This task is to build a tower out of blocks that fit together like simple puzzles pieces to create a ziggurat (stepped pyramid). Two pieces fit together to create a half level; two half-levels make a level; and five levels make the tower along with a capping piece. The task was created by Wood (Wood, 1976 #542 cited in Ritter & Major, 1994), who has compiled an extensive set of regularities of children's behaviour while solving it and while they were taught how to solve it by teachers and other children.

The interface (MONGSU) provided a communication link similar to our ATC task. But in this task, the data being passed is large due to the complexity of the world and the frequency of the interactions between the Soar model and the simulation. Data needs to passed about which hand (right and left) holds which block, which block to put with which block, which block create a layer and so on, until the pyramid gets built. In fact, the usage of the socket utility is exactly the same as the ATC task except for their world.

Because this simulation passes more information, it was this task that lead to the discovery that old input links are not being replaced by new ones. This situation is compounded when the input-links are getting piled up in the top-state which led to stack overflow because of huge data being accepted by the Soar model. It was therefore suggested that there should be a way to replace old data structures of an input link automatically everytime a new input link is added to the top-state. This facility is not yet provided by MONGSU (i.e., the interface), so old WMEs must be explicitly removed and the new WMEs added on the top-state, otherwise it would cause data overflow.

Another feature that should be added to the interface mechanism is the problem of waiting for incoming data which is a big problem when Soar is communicating with Lisp. Soar is relatively fast and Lisp and sockets (we noticed) is slow causing extensive waiting on the part of Soar. Something could and should probably be done during this long period of waiting, — perhaps further processing, reflecting, or clearing the desk. It may also simply be an effect of a time-sharing system. The delay represents the minimum time slice that can be given to a processor. In any case, it would be interesting to know what causes this sometimes considerable delay.

8. THE ROAD AHEAD

The system developed in this project (shown in Figure 6) illustrates what you can do once you routinely tie a Soar model to an interactive simulation. The model was able do basic tasks such as looking at the scope, lowering the altitude of a plane and landing a plane. It also demonstrated the use of a robust interface mechanism that provided the link between the Soar model and the ATC simulation via sockets. But we can imagine a higher level system implementation that does a lot more. Figure 15 illustrates such a system.

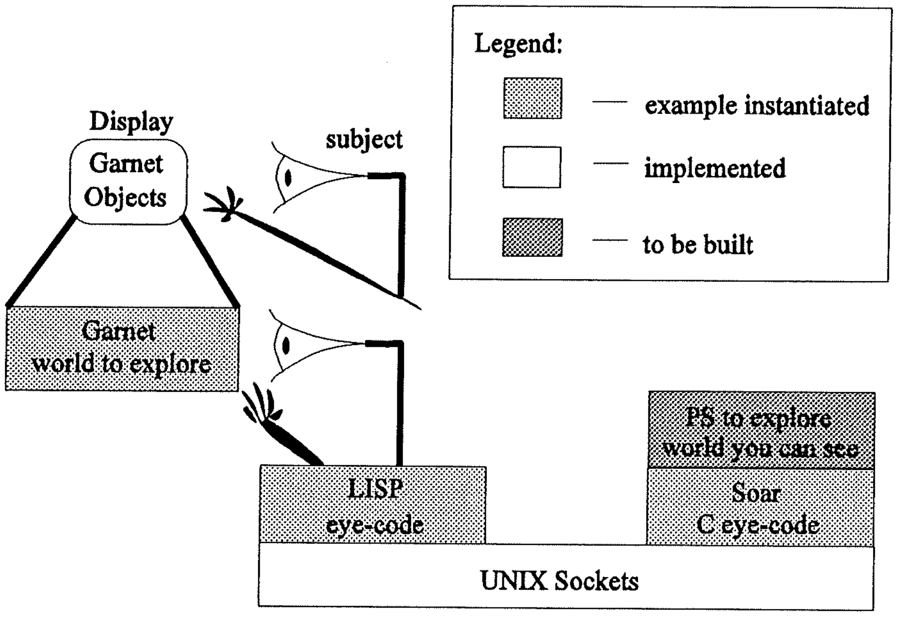


Figure 15. System Implementation – the road ahead.

The figure above is essentially our system implementation in Figure 6. Boxes in dotted-grids are implemented as part of our example instantiation which is the ATC task while the plain box (white in colour) is the implementation of a routine mechanism that provides interaction between Soar model and simulation. The main distinction between our system (Figure 6) and the system above (Figure 15) is in the way data is manipulated within the respective system — the grey box. In Figure 6, low-level data manipulation gives a scenario wherein the data being manipulated by the Soar model are pre-arranged or pre-fixed types of information passed to it by the simulation, such as the plane information stored in a global variable. In other words, it only allows data to be passed between the model and the simulation on a pre-fixed level without spontaneity, which in our case are data gathered through perceptual knowledge. This approach is desirable and a necessary prelude (especially for our system) to provide a system that routinely ties a cognitive model to a simulation. It provides us with a fixed set of known manipulations and objects to start with — similar to having a list of fixed questions and fixed answers known to conversing parties. But a higher-level objective is to be able to model human behaviour interacting with a new interface, or set of interfaces. In this case, the model should have a way of manipulating the Garnet objects themselves, must be able to learn and look at any objects.

The high-level manipulation of data of the system in Figure 15 arises in the way the Soar model manipulates objects in the ATC simulation. To start with, the Soar model should be able to manipulate and modify the attributes of the planes without much intervention from the simulation – similar to having fingers and claws on the Garnet objects (i.e., planes). It is not clear at the moment how big or important this system when implemented but a set of known manipulations and objects (i.e., our implemented system) may be a good start. Furthermore, what we are after in implementing this system is to develop task knowledge of performing air traffic control task not just knowledge through perception. This would allow the new system to learn new facts, new representations, strategies and operations, eventually enabling the Soar model to explore anything it can see and get hold of in its world.

9. CONCLUDING REMARKS

This project presents a tool and mechanism that routinely ties a Soar model to an ATC simulation. This was achieved by creating a simulation of an ATC task, a Soar model of the subject, and the use of an interface mechanism that made the two communicate with each other. The ATC simulation illustrated the model of a grossly simplified ATC task that was able to display planes, beacons and other objects, while the Soar model was able to perform some basic ATC functions. Through the sample runs, the Soar-Garnet system was able to show what benefits routine interaction between the Soar model and the ATC simulation provides. These benefits include (a) learning new ways of doing I/O with Soar by writing C code that creates new commands in Soar to use sockets, (b) learning new ways of interpreting a Soar model, such as the *choices* and *time-in-slot* analysis, (c) providing a socket-based mechanism that allowed Soar to play with a simulation, and most importantly, (d) providing a model that starts to model human behaviour in the ATC task.

Throughout this project, it was found out that interface mechanism using sockets (adapted, modified, and made into a utility) is very robust, stable, general and routine. Clearly, it was able to show the generality it provides across two applications (ATC task and ToN task) with little or no modifications. Moreover, if used more often on other tasks, it should become even easier to use.

Indeed, this is the beginning of something that could revolutionalize cognitive modelling as a means of emulating human behaviour doing a real-time interactive task through making interaction with the world routine. If that is the case, efforts should be made to address the problems found in its usage, such as the problem of old WMEs piling up in Soar's memory (causing stack overflow), and not being able to remove old WMEs by new WMEs. Moreover, the socket utility should be used and tested across different applications to continuesly update and modify it for it to be more general to substantiate results found in this project.

Finally, the work done here should be extended to achieve what we imagine is the ultimate system in routinely tying cognitive models to interactive simulations (the system shown in Figure 15). This high-level system is just a glimpse away, and its importance would only be clear through further examinations, investigations and development. This would eventually lead to a system that could see, grab, and manipulate whatever objects exists in the world, and a way of providing multiple worlds for the simulation to interact with. But essentially, what we've done here is to lay the groundwork from which to build and further extend this kind of system and other similar systems that learns from the world and that provide routine interaction between a Soar model and a simulation. Over time, this approach would paved the way for bringing us a step closer in emulating human behaviour doing interactive real-time tasks.

REFERENCES

Glass, G. (1993). *UNIX for Programmers and Users: A Complete Guide*. Englewood Cliffs: Prentice-Hall, Inc.

Guthrie, Wade (1994). *Platform Independent Graphical User Interface (PIGUI) Review.* Monthly posting Internet newsgroup: "**comp.windows**".

Hucka, M. (1994). *Soar Developement Environment - Release 0.10.* README file. Department of Electrical Engineering and Computer Science — University of Michigan, Ann Arbor MI. FTPable from **centro.soar.cs.cmu.edu**: "/afs/cs/project/soar/public/Soar6/sde-0.10.tar.Z".

Kernighan, B.W. and Ritchie, D.M. (1978). *The C Programming Language*. Englewood Cliffs: Prentice-Hall, Inc.

Laird, J.E., Congdon, C.B., Altmann, E. and Doorenbos, R. (1993). *Soar User's Manual: Version 6.* Edition 1. Pittsburgh PA: Carnegie-Mellon University, School of Computer Science.

Martin, D., Prata, S. and Waite, M. (1984). *C Primer Plus.* Indianapolis: Howard W. Sams & Co., Inc.

Myers, B.A. (1994). *Garnet Toolkit Frequently Asked Questions.* Periodic posting Internet newsgroup: "**comp.windows.garnet.**"

Myers, et al. (1993). *The Garnet Reference Manuals Revised for Version 2.2.* Pittsburgh PA: Carnegie-Mellon University, School of Computer Science. (CMU-CS-90-117-R4)

Nelson, G. (1994). *SimTime.* Unpublished computer program. The Soar Group, School of Computer Science — Carnegie-Mellon University, Pittsburgh PA. FTPable from **centro.soar.cs.cmu.edu**: "/afs/cs/project/soar/public/Soar6/user-library/SimTime.tar.Z".

Newell, A. (1992). "Precis of Unified theories of cognition." *Behavioral and Brain Sciences, 15,* 425-437.

Nolan, M.S. (1990). *Fundamentals of Air Traffic Control.* Belmont CA: Wadsworth Publishing Co., Inc.

Pelton, G. (1994). *New-Soar-IO.* Unpublished computer program. The Soar Group, School of Computer Science — Carnegie-Mellon University, Pittsburgh PA. FTPable from **centro.soar.cs.cmu.edu**: "/afs/cs/project/soar/member/gap/soar6/new-soar-io/default-io.c".

Ritter, F.E. and Larkin, J.H. (1994). "Developing Process Models as Summaries of HCI Action Sequences." *Human-Computer Interaction* (in press).

Ritter, F.E. and Major, N. (1994). "Developing Simulations for Soar to Play With." *ESRC Centre for Research and Development, Instruction and Training.* Technical Report No. 18.

Rogers, S. (1994). *Stopwatch.* Unpublished computer program. Department of Electrical Engineering and Computer Science — University of Michigan, Ann Arbor MI. FTPable from **flamingo.eecs.umich.edu**: "/u/srogers/soar/soar6.2/stopwatch.c".

Steele, G.L., Jr. (1990). *Common LISP: The Language.* Second Edition. Bedford MA: Digital Press Inc.

Waldrop, M.M. (1988a). "Toward a Unified Theory of Cognition." *Science, 241*, 27-29.

Waldrop, M.M. (1988b). "Soar: A Unified Theory of Cognition?" *Science, 241*, 296-298.

Appendix A

ATC — README

```
                         ATC Package
            Air Traffic Control Simulation Package
                         Release 1.0

                        Roberto L. Ong
                   itxrlo@unicorn.nott.ac.uk
                      September 15, 1994


INTRODUCTION
============

This is release notes for a simple Air Traffic Control (ATC) task and a simple
Soar model to drive it, implemented in Garnet, Lisp, and Soar, tied together
with Unix sockets.  It includes (a) how to obtain this code, (b) the structure
of the code, (c) a description of the contents of this package, (d) how to
set-up the ATC code, and (e) how to get help.

The ATC code illustrates the basic utility in routinely tying a Soar model to
an interactive simulation using a refined socket packaged called MONGSU
(Mertz-Ong-Nerb-Gary Socket Utility).  It includes the Garnet simulation, the
Soar model, and the general socket code.


OBTAINING AND INSTALLING THE PACKAGE
====================================

The ATC code is available via anonymous ftp from host 128.243.40.7
(unicorn.ccc.nott.ac.uk, but many machines don't know it, so you may wish to
use the numbers) in the directory "/pub/lpzfr" (From within ftp only the part
of the tree rooted at /usr/ftp is visible).

If you would like help with using anonymous ftp, feel free to send me a
message and I will explain the procedure.


ATC CODE STRUCTURE
==================

The ATC code structure is divided into three parts: the simulation, the model,
and the interface (i.e., sockets).  Below is an illustration of the ATC code
structure.


The Simulation:                          The Model:
    The Garnet ATC simulation                PS to do ATC task
    The Lisp helper functions                Soar top-state attribute-value
    (also called Lisp eye/hand code)         (also called Soar C eye-code)
```

```
                         The Interface:
            Lisp sockets                  Soar sockets
                        Standard sockets
```

CONTENTS OF THIS PACKAGE
========================

The files included in this package are:

ATC-README
This file contains the release notes about the ATC source code. It describes how to obtain, install and use the source code.

ATC-files.load
This file loads the necessary files to run the ATC simulation.

ATC-Garnet.lisp
This file contains the Garnet part of the ATC simulation code. It includes the creation of objects, the functions to manipulate them, and the main routine to run the simulation.

ATC-fn.lisp
The core of the communication link with Soar. It contains the functions that carry out conversation proceedings with a Soar model. It also contains functions that support data evaluation and creation. The main routine in this file (a) launches a process that creates a socket and listens for incoming requests for a connection, (b) runs the ATC simulation, and (c) evaluate data coming from Soar (in this case, commands). Commands coming from Soar (in a list) are evaluated and resulting data (also in lists) are sent back to Soar.

Database files:
These files contain databases of beacon and plane locations and other attributes that are loaded as plane information for the ATC simulation. They are quite simple. Follow the procedures below on how to add them.

plane.db
A sample file will look something like:

```
2
CX901 200 200 20000 5 under land
PR301 100 400 25000 6 under over
```

The first line contains the number of planes. Succeeding lines contain data for each plane. The data for each row are:

flight-number x-position y-position altitude velocity control status

beacon.db
A sample file will look something like:

```
2
100 100
200 200
```

The first line contains the number of beacons. Succeeding lines contain the x and y locations of each beacon.

```
HOW TO SET UP THE ATC CODE
==========================

This code runs with the latest version of Soar 6.2.3 (non-nnpscm), and with
Garnet 2.2 running in a Lisp Lucid image. The following set-up procedures can
be used to set it up at the Psychology department from within the University
of Nottingham.  Other users will need modify it for their own site.

HOW TO SET UP THE LISP CODE
[0] Start a garnet-lucid-image by
    -> loading emacs and do M-x garnet, or
    -> loading "/psyc/lang/garnet/2.2/garnet-lucid-image"
    (Outside of Nottingham, you need to get a lisp image, get garnet, compile
    garnet, load it, and make an image.)

[1] (load "/psyc/teaching/UG/otherug/itxro/garnet/ATC-files.load)

[2] Run the main-routine:
    (main-routine)

     This call will printout the following lines below and start the ATC
     simulation.

    Created Socket with id 6
    Created socket 6.
    Bound socket 6 to port 1282
                           ^^^*
    Listening at socket 6
    Listening at socket with id 6
    #<Process Soar-read-loop-process 144148E>

    *This port number is needed by Soar to set up a socket link to Lisp.

[3] To load planes, choose "File" from the menu-bar and select "Load-File".
    This will load the plane database.

[4] When you finish, good UNIX hygiene suggests that you close the socket
    stream with (shutdown-socket-stream)

NOTE: If something goes wrong, or an error occurs, you must REMEMBER to close
      the socket stream with

              (shutdown-socket-stream)

      To restart the entire socket process, go to step [2].
      Unless, your garnet-image crashes, go to step [0].
      In either case, you then must restart Soar again.
      (See the set-up procedure below)


HOW TO SET UP THE SOAR SOCKET CODE
[0]  Start a Soar program by compiling Soar together with MONGSU.
     Instructions on how to go about this can be found in the release notes
     of MONGSU (Mertz-Ong-Nerb-Gary Socket Utility).

[1]  Load Soar default rules.

[2]  Load ATC.soar ("/psyc/teaching/UG/otherug/itxro/garnet/ATC.soar")
     Instructions on how to obtain and run this code can be found in the
     release notes of ATC-Soar.
```

```
[3] After Garnet is up and ATC code is loaded, type:

       init-socket-io server-name port-number

       ;; this is a new Soar command created by MONGSU
       ;; where:
       ;;   server-name is the server where garnet runs
       ;;   port-number is the number return by the Lisp socket code

[4] you can now run the program

NOTE: REMEMBER to issue the command: close-socket-io
      whenever you are finish using the sockets and want to terminate link
      with Lisp/Garnet.


HOW TO GET HELP
===============

If you require any assistance regarding this package, please drop an email to
<itxrlo@unicorn.nott.ac.uk>


Roberto L. Ong       <itxrlo@unicorn.nott.ac.uk>
September 15, 1994   - University Park
```

Appendix B

ATC-Soar — README

```
                         ATC-Soar
              Air Traffic Control Soar Model
                       Release 1.0

                     Roberto L. Ong
               itxrlo@unicorn.nott.ac.uk
                   September 27, 1994
```

INTRODUCTION
============

This is release notes for a simple Air Traffic Control (ATC) Soar model that drives the ATC simulation. It includes (a) how to obtain and run this code, and (b) how to get help.

The ATC Soar model illustrates the basic (and simple) functions in doing the ATC task. It includes the external operators 'look-scope', 'lower-altitude' and 'land-plane' when implemented issues commands to the simulation to do basic tasks such as looking at the scope and landing a plane. It also includes a 'watching-command-process' operator that act as a wait and add delay to Soar in order for it to receive data coming from the slow Simulation. In addition, a bunch of other operators (non-external) also exist to support the performance of this simple ATC task. It is extensible so that users can modify and add rules to do more complex tasks, such as the detection of a plane crash or near misses.

It must be noted that Soar's default rules are loaded and used by this model as part of its total rule count (a total of 139).

OBTAINING AND RUNNING THE PACKAGE
=================================

The ATC Soar code is available via anonymous ftp from host 128.243.40.7 (unicorn.ccc.nott.ac.uk, but many machines don't know it, so you may wish to use the numbers) in the directory "/pub/lpzfr" (From within ftp only the part of the tree rooted at /usr/ftp is visible).

If you would like help with using anonymous ftp, feel free to send me a message and I will explain the procedure.

HOW TO RUN THE ATC SOAR MODEL:

This code runs with the latest version of Soar 6.2.3 (non-nnpscm) with Mertz-Ong-Nerb-Gary Socket Utility (MONGSU) compiled together. The following procedures can be used to run this model at the Psychology department from within the University of Nottingham. Other users will need modify it for their own site.

```
[1] Start a Soar program by
    -> loading emacs and do M-x soar, or
    -> loading "/psyc/teaching/UG/otherug/itxro/soar6.2.3/bin/non-nnpscm
                /sun4/soar"
    (Outside of Nottingham, you need to get a Soar program 6.2.3 and MONGSU,
     and compile it together.)

[2] Load Soar default rules.

[3] Load ATC.soar ("/psyc/teaching/UG/otherug/itxro/garnet/ATC.soar")
    ("ATC.soar" is the source code for the ATC Soar model.)

[4] you can now run the program
     (It is assumed here that a socket link has already been established with
     the ATC simulation. If link does not yet exist, follow the procedures in
     setting up the ATC simulation package from which this is a part of.)


HOW TO GET HELP
===============

If you require any assistance regarding this package, please drop an email to
<itxrlo@unicorn.nott.ac.uk>


Roberto L. Ong      <itxrlo@unicorn.nott.ac.uk>
September 27, 1994  - University Park
```

Appendix C

MONGSU — README

```
                         MONGSU
           Mertz-Ong-Nerb-Gary Socket Utility
                       Release 1.0

                     Roberto L. Ong
                 itxrlo@unicorn.nott.ac.uk
                   September 15, 1994
```

INTRODUCTION
============

This is a manual for the socket code that implements Soar and Lisp I/O using Unix sockets. It includes (a) how to obtain and install the package, (b) the contents of this package, (c) the necessary changes to the makefile (for compiling Soar) and hooks.c, and (d) where to get help.

This socket code provides the basic utilities needed for creating Soar I/O in C and Lisp I/O in Common Lisp. It supports the direct and routine use of Unix sockets by providing functions such as create socket, bind socket, etc. It supports passing lists of structures to Soar that automatically end up in working memory without writing additional C code, and defines additional commands for Soar.

It is extensible, so that users can write their own C commands and functions, for example, to allow a different parsing procedure of incoming and outgoing data other than a list structure. It should make the use of sockets more routine. It also works with systems besides lisp if they will pass down the socket lists of WMEs, and accept lists in return.

OBTAINING AND INSTALLING THE PACKAGE
====================================

The Socket code is available via anonymous ftp from host 128.243.40.7 (unicorn.ccc.nott.ac.uk, but many machines don't know it, so you may wish to use the numbers) in the directory "/pub/lpzfr" (From within ftp only the part of the tree rooted at /usr/ftp is visible).

If you would like help with using anonymous ftp, feel free to send me a message and I will explain the procedure.

To install the system:

[1] You must first have a Soar Release 6.2 (or later) in your home directory (or some other directory in your system). Copy the necessary makefiles to your "<soar-directory>".

[2] Then copy the Soar socket codes to your "<soar-directory>/user/src" directory and the Lisp socket codes to your "<lisp-directory>". All Soar socket codes have their filenames contain the word "soar", while the Lisp codes have their filenames contain the extension ".lisp", so there should be no trouble in determining them.

[3] Edit the makefile (in "<soar-directory>/make.body") to include the necessary path to your <soar-directory>. You also have to put the necessary changes to hooks.c in "<soar-directory>/src/hooks.c".

[4] Execute "make" in your "<soar-directory>". This will compile the Soar socket code in .c files together with the modified hooks.c file.

N.B. You can copy "make.body" and "hooks.c" (modified for your convenience) from the ftp directory.

[5] Load the "socket.lisp" and "stdio.lisp" from within Lisp (we use a Garnet Lucid Image). You can now start calling the functions defined in these files to start your communication.

FILES
=====

The files included in the package are:

README
This file contains the release notes about the Mertz-Ong-Nerb-Gary Socket Utility (MONGSU). It describes how to obtain, install and use the source code.

make.body
This file contains the body of the makefile that will be used for Soar to be compiled together with Soar socket code. It already exists in your <soar-directory> (it comes with the Soar program), you just have to incorporate the necessary changes to it or use the one included in this package.

hooks.c
This file contains the necessary changes in order for some C functions to be called during specific time during compilation. It already exists in your <soar-directory> (it comes with the Soar program), you just have to incorporate the necessary changes to it or use the one included in this package.

std-soar-socket.c
This file contains the basic open, listen, bind, etc routines for sockets. It is used by both the Soar sockets and Lisp sockets (as std-soar-socket.o).

soar-socket.c
The core of the sockets code for Soar. It contains the necessary functions for Soar to act as a client (in a UNIX client-server model), the input and output functions, the additional commands, and the parsing routines. The parsing routines parses incoming data (in lists) and build them as WMEs, while WMEs are parse into lists for outgoing data.

I communicate with a Lisp process, so the following files are provided:

socket.lisp
This package contains the functions that create a socket for Lisp to act as a server or a client. It supports Lucid, Allegro, and CMU-Lisp.

stdio.lisp
This file contains the utilities that support Lisp socket communication. It includes read and write functions and other utilities. User-defined functions supporing Lisp sockets are placed here.

There are also some small programs for doing socket testing.

clienttest.c clienttest.out
The source and the executable file provides a communication link with a Lisp server (hence the name, clienttest) and tests it. Follow the testing procedure below.

servertest.c servertest.out
The source and the executable file provides a communication link with a Lisp client (hence the name, servertest) and tests it. Follow the testing procedure below.

TESTING PROCEDURE
=================

[1] Start-up a Lisp program.

N.B. It is advised that you used Lucid-4.1 because all source code were tested using it. However, other Lisp program would suffice (if you don't have Lucid) with perhaps some modifications to the code.

[2] Load the files "socket.lisp" and "stdio.lisp".

<u>To test a Lisp server:</u>

[3] Type: (set-up-socket-as-server)
This creates a socket as a server process.

[4] From a Unix prompt, type: clienttest.out server-name port-number
where:
server-name is the host name where your Lisp process is running.
port-number is the port number returned when you set up a Lisp server process.

[5] (accept-client-and-make-socket-stream) accepts a request from a client process (in this case, clienttest.out) and creates a socket-stream.

[6] (do-stuff) accepts a list structure defined in clienttest.c, evaluates and prints it.

An output looks something like:
```
Received message: (+ 1 2)
Evaluated message: 3
Closed socket with id 6
Socket 6 closed.
Channel interface is shutdown.
nil
```

To test a Lisp client:

[3] From a Unix prompt, type: servertest.out
This creates a server process in Unix and returns a port-number.

[4] Type: (set-up-socket-as-client "server-name" port-number) on the Lisp prompt. This creates a socket for the client process.
where:

```
  "server-name" is the name of the host where your Lisp process
                is running in quotes.
  port-number   is the number returned by running servertest.out.
```

[5] Type: (test-write)
This command writes some strings into the socket while the server process in Unix receives and prints it out.

A sample output looks something like:

```
-->hello-there
-->this-is-a-test
Ending connection
```

To test a Soar client to a Lisp server:

[3] Load Soar (6.2.3) with the necessary files loaded (see installation procedure).

[4] In Lisp, type: (set-up-socket-as-server)
This would return a port-number for use within Soar.

[5] In Soar, type: init-socket-io server-name port-number
where:

```
  server-name is the host name where your Lisp process is running.
  port-number is the port number returned when you set up a Lisp server
              process.
```

[6] Type (accept-client-and-make-socket-stream) in Lisp to accept the Soar client process and create a socket stream.

[7] Load any program in Soar (e.g., you could load example programs included in the Soar package).

[8] In Lisp, use "write-message" to send a list structure to Soar and instantiate it on the top-state.

A sample command looks like:
(write-message '(input-link hello))
This would create an attribute in the top-state (of your loaded Soar program) named ^input-link with value hello.

N.B. REMEMBER to (shutdown-socket-stream) when you finished running the Lisp process and "close-socket-io" with the Soar process.

<u>To test a Soar server to a Lisp client:</u>

[3] Load Soar (6.2.3) with the necessary files loaded (see installation procedure).

[4] In Soar, type: init-socket-server
This will return a port number for used by the Lisp client.

[5] In Lisp, type: (set-up-socket-as-client "server-name" port-number)

[6] Load any program in Soar (e.g., you could load example programs included in the Soar package).

```
[7] In Lisp, use "write-message" to send a list structure to Soar and
    instantiate it on the top-state.

    A sample command looks like:
    (write-message '(input-link hello))
    This would create an attribute in the top-state (of your loaded Soar
    program) named ^input-link with value hello.

N.B. REMEMBER to (close-socket-io) when you finished running the Lisp process
     and "shutdown-socket-io" with the Soar process.


HOW TO GET HELP
===============

If you require any assistance regarding this package, please drop an email to
<itxrlo@unicorn.nott.ac.uk>

Roberto L. Ong       <itxrlo@unicorn.nott.ac.uk>
September 15, 1994   - University Park
```

Appendix D

A Sample Run

D.1 Sample Run 1 (Wed Sep 14 13:21:03 1994)

```
Soar 6.2.3

Bugs and questions should be sent to soar-bugs@cs.cmu.edu
The current bug-list may be obtained by sending mail to
soar-bugs@cs.cmu.edu with the Subject: line "bug list".

This software is in the public domain, and is made available AS IS.
Carnegie Mellon University, The University of Michigan, and
The University of Southern California/Information Sciences Institute
make no warranties about the software or its performance, implied
or otherwise.

Type "help" for information on various topics.
Type "quit" to exit.  Use ctrl-c to stop a Soar run.
Type "soarnews" for news.
Type "version" for complete version information.

Loading /psyc/teaching/UG/otherug/itxro/.init.soar


Loading
/psyc/teaching/UG/otherug/itxro/soar6.2.3/default/non-nnpscm/default.soar
****************************************************************************
********************************

Loading /psyc/teaching/UG/otherug/itxro/garnet/ATC.soar
********************************

Soar> init-socket-io upsyc 3200
Created Socket with id 3
Connecting to server with name upsyc
Connecting to server with port #3200
connected to socket 3
Soar> d

     0: ==>G: G1
     1:    P: P1 (atc)
     2:    S: S1
     3:    O: O1 (look-scope)
write successful socket 3

     4:    ==>G: G2 (operator no-change)
     5:       P: P1 (atc)
     6:       S: S1
     7:       O: O2 (watching-command-process)
     8:       O: O2 (watching-command-process)
```

```
  9:         O: O2 (watching-command-process)
 10:         O: O2 (watching-command-process)
 11:         O: O2 (watching-command-process)
 12:         O: O2 (watching-command-process)
 13:         O: O2 (watching-command-process)
 14:         O: O2 (watching-command-process)
 15:         O: O2 (watching-command-process)
 16:         O: O2 (watching-command-process)
 17:         O: O2 (watching-command-process)
 18:         O: O2 (watching-command-process)
 19:         O: O2 (watching-command-process)
 20:         O: O2 (watching-command-process)
 21:         O: O2 (watching-command-process)
 22:         O: O2 (watching-command-process)
 23:         O: O2 (watching-command-process)
 24:         O: O2 (watching-command-process)
 25:         O: O2 (watching-command-process)
 26:         O: O2 (watching-command-process)
 27:         O: O2 (watching-command-process)
 28:         O: O2 (watching-command-process)
 29:         O: O2 (watching-command-process)
 30:         O: O2 (watching-command-process)
 31:         O: O2 (watching-command-process)
 32:         O: O2 (watching-command-process)
 33:         O: O2 (watching-command-process)
 34:         O: O2 (watching-command-process)
 35:         O: O2 (watching-command-process)
 36:         O: O2 (watching-command-process)
 37:         O: O2 (watching-command-process)
 38:         O: O2 (watching-command-process)
 39:         O: O2 (watching-command-process)
 40:         O: O2 (watching-command-process)
 41:         O: O2 (watching-command-process)
 42:         O: O2 (watching-command-process)
 43:         O: O2 (watching-command-process)
 44:         O: O2 (watching-command-process)
 45:         O: O2 (watching-command-process)
 46:         O: O2 (watching-command-process)
 47:         O: O2 (watching-command-process)
 48:         O: O2 (watching-command-process)
 49:         O: O2 (watching-command-process)
 50:         O: O2 (watching-command-process)
 51:         O: O2 (watching-command-process)
 52:         O: O2 (watching-command-process)
 53:         O: O2 (watching-command-process)
 54:         O: O2 (watching-command-process)
 55:         O: O2 (watching-command-process)
 56:         O: O2 (watching-command-process)
 57:         O: O2 (watching-command-process)
 58:         O: O2 (watching-command-process)
 59:         O: O2 (watching-command-process)
 60:         O: O2 (watching-command-process)
 61:         O: O2 (watching-command-process)
 62:         O: O2 (watching-command-process)
 63:         O: O2 (watching-command-process)
 64:         O: O2 (watching-command-process)
 65:         O: O2 (watching-command-process)
 66:         O: O2 (watching-command-process)
 67:         O: O2 (watching-command-process)
 68:         O: O2 (watching-command-process)
 69:         O: O2 (watching-command-process)
```

```
     70:         O: O2 (watching-command-process)
     71:         O: O2 (watching-command-process)
     72:         O: O2 (watching-command-process)
     73:         O: O2 (watching-command-process)
     74:         O: O2 (watching-command-process)
     75:         O: O2 (watching-command-process)
     76:         O: O2 (watching-command-process)
     77:         O: O2 (watching-command-process)
     78:         O: O2 (watching-command-process)
     79:         O: O2 (watching-command-process)
     80:         O: O2 (watching-command-process)
     81:         O: O2 (watching-command-process)
     82:         O: O2 (watching-command-process)
     83:         O: O2 (watching-command-process)
     84:         O: O2 (watching-command-process)
     85:         O: O2 (watching-command-process)
     86:         O: O2 (watching-command-process)
     87:         O: O2 (watching-command-process)
     88:         O: O2 (watching-command-process)
     89:         O: O2 (watching-command-process)
     90:         O: O2 (watching-command-process)
     91:         O: O2 (watching-command-process)
     92:         O: O2 (watching-command-process)
     93:         O: O2 (watching-command-process)
     94:         O: O2 (watching-command-process)
     95:         O: O2 (watching-command-process)
     96:         O: O2 (watching-command-process)
     97:         O: O2 (watching-command-process)
     98:         O: O2 (watching-command-process)
     99:         O: O2 (watching-command-process)
    100:         O: O2 (watching-command-process)
    101:         O: O2 (watching-command-process)
    102:         O: O2 (watching-command-process)
    103:         O: O2 (watching-command-process)
    104:         O: O2 (watching-command-process)
    105:         O: O2 (watching-command-process)
    106:         O: O2 (watching-command-process)
    107:         O: O2 (watching-command-process)
    108:         O: O2 (watching-command-process)
    109:         O: O2 (watching-command-process)
    110:         O: O2 (watching-command-process)
    111:         O: O2 (watching-command-process)
    112:         O: O2 (watching-command-process)
    113:         O: O2 (watching-command-process)
    114:         O: O2 (watching-command-process)
    115:         O: O2 (watching-command-process)
    116:         O: O2 (watching-command-process)
    117:         O: O2 (watching-command-process)
    118:         O: O2 (watching-command-process)
    119:         O: O2 (watching-command-process)
    120:         O: O2 (watching-command-process)
    121:         O: O2 (watching-command-process)
    122:         O: O2 (watching-command-process)
    123:         O: O2 (watching-command-process)
    124:         O: O2 (watching-command-process)
    125:         O: O2 (watching-command-process)
    126:         O: O2 (watching-command-process)
    127:         O: O2 (watching-command-process)
    128:         O: O2 (watching-command-process)
    129:         O: O2 (watching-command-process)
    130:         O: O2 (watching-command-process)
```

```
131:        O: O2 (watching-command-process)
132:        O: O2 (watching-command-process)
133:        O: O2 (watching-command-process)
134:        O: O2 (watching-command-process)
135:        O: O2 (watching-command-process)
136:        O: O2 (watching-command-process)
137:        O: O2 (watching-command-process)
138:        O: O2 (watching-command-process)
139:        O: O2 (watching-command-process)
140:        O: O2 (watching-command-process)
141:        O: O2 (watching-command-process)
142:        O: O2 (watching-command-process)
143:        O: O2 (watching-command-process)
144:        O: O2 (watching-command-process)
145:        O: O2 (watching-command-process)
146:        O: O2 (watching-command-process)
147:        O: O2 (watching-command-process)
148:        O: O2 (watching-command-process)
149:        O: O2 (watching-command-process)
150:        O: O2 (watching-command-process)
151:        O: O2 (watching-command-process)
152:        O: O2 (watching-command-process)
153:        O: O2 (watching-command-process)
154:        O: O2 (watching-command-process)
155:        O: O2 (watching-command-process)
156:        O: O2 (watching-command-process)
157:        O: O2 (watching-command-process)
158:        O: O2 (watching-command-process)
159:        O: O2 (watching-command-process)
160:        O: O2 (watching-command-process)
161:        O: O2 (watching-command-process)
162:        O: O2 (watching-command-process)
163:        O: O2 (watching-command-process)
164:        O: O2 (watching-command-process)
165:        O: O2 (watching-command-process)
166:        O: O2 (watching-command-process)
167:        O: O2 (watching-command-process)
168:        O: O2 (watching-command-process)
169:        O: O2 (watching-command-process)
170:        O: O2 (watching-command-process)
171:        O: O2 (watching-command-process)
172:        O: O2 (watching-command-process)
173:        O: O2 (watching-command-process)
174:        O: O2 (watching-command-process)
175:        O: O2 (watching-command-process)
176:        O: O2 (watching-command-process)
177:        O: O2 (watching-command-process)
178:        O: O2 (watching-command-process)
179:        O: O2 (watching-command-process)
180:        O: O2 (watching-command-process)
181:        O: O2 (watching-command-process)
182:        O: O2 (watching-command-process)
183:        O: O2 (watching-command-process)
184:        O: O2 (watching-command-process)
185:        O: O2 (watching-command-process)
186:        O: O2 (watching-command-process)
187:        O: O2 (watching-command-process)
188:        O: O2 (watching-command-process)
189:        O: O2 (watching-command-process)
190:        O: O2 (watching-command-process)
191:        O: O2 (watching-command-process)
```

```
   192:       O: O2 (watching-command-process)
   193:       O: O2 (watching-command-process)
   194:       O: O2 (watching-command-process)
   195:       O: O2 (watching-command-process)
   196:       O: O2 (watching-command-process)
   197:       O: O2 (watching-command-process)
   198:       O: O2 (watching-command-process)
   199:       O: O2 (watching-command-process)
   200:    O: O7 (lower-altitude)
  Flight number pr301 has reached First approach fix.
  Descending to 5000 meters.
write successful socket 3

   201:    ==>G: G3 (operator no-change)
   202:       P: P1 (atc)
   203:       S: S1
   204:       O: O10 (watching-command-process)
   205:       O: O10 (watching-command-process)
   206:       O: O10 (watching-command-process)
   207:       O: O10 (watching-command-process)
   208:       O: O10 (watching-command-process)
   209:       O: O10 (watching-command-process)
   210:       O: O10 (watching-command-process)
   211:       O: O10 (watching-command-process)
   212:       O: O10 (watching-command-process)
   213:       O: O10 (watching-command-process)
   214:       O: O10 (watching-command-process)
   215:       O: O10 (watching-command-process)
   216:       O: O10 (watching-command-process)
   217:       O: O10 (watching-command-process)
   218:       O: O10 (watching-command-process)
   219:       O: O10 (watching-command-process)
   220:    O: O9 (look-scope)
write successful socket 3

   221:    ==>G: G4 (operator no-change)
   222:       P: P1 (atc)
   223:       S: S1
   224:       ==>G: G5 (operator tie)
   225:          P: P4 (selection)
   226:          S: S2
   227:          O: O19 (evaluate-object O15 (lower-altitude))
   228:          ==>G: G6 (operator no-change)
   229:             P: P1 (atc)
   230:             S: D3
   231:             O: C4 (lower-altitude)
  Flight number pr301 has reached First approach fix.
  Descending to 5000 meters.
   232:             ==>G: G7 (operator no-change)
   233:                P: P1 (atc)
   234:                S: D3
   235:                O: O23 (watching-command-process)
   236:                O: O23 (watching-command-process)
   237:                O: O23 (watching-command-process)
   238:                O: O23 (watching-command-process)
   239:                O: O23 (watching-command-process)
   240:                O: O23 (watching-command-process)
   241:                O: O23 (watching-command-process)
   242:    O: O25 (there-are-planes)
  There are planes to land.
```

```
   243:      O: O28 (look-scope)
write successful socket 3

   244:      ==>G: G8 (operator no-change)
   245:         P: P1 (atc)
   246:         S: S1
   247:         O: O31 (watching-command-process)
   248:         O: O31 (watching-command-process)
   249:         O: O31 (watching-command-process)
   250:         O: O31 (watching-command-process)
   251:         O: O31 (watching-command-process)
   252:         O: O31 (watching-command-process)
   253:         O: O31 (watching-command-process)
   254:         O: O31 (watching-command-process)
   255:         O: O31 (watching-command-process)
   256:         O: O31 (watching-command-process)
   257:         O: O31 (watching-command-process)
   258:         O: O31 (watching-command-process)
   259:         O: O31 (watching-command-process)
   260:         O: O31 (watching-command-process)
   261:         O: O31 (watching-command-process)
   262:         O: O31 (watching-command-process)
   263:         O: O31 (watching-command-process)
   264:      O: O35 (there-are-planes)
  There are planes to land.
   265:      O: O38 (look-scope)
write successful socket 3

   266:      ==>G: G9 (operator no-change)
   267:         P: P1 (atc)
   268:         S: S1
   269:         O: O41 (watching-command-process)
   270:         O: O41 (watching-command-process)
   271:         O: O41 (watching-command-process)
   272:         O: O41 (watching-command-process)
   273:         O: O41 (watching-command-process)
   274:         O: O41 (watching-command-process)
   275:         O: O41 (watching-command-process)
   276:         O: O41 (watching-command-process)
   277:         O: O41 (watching-command-process)
   278:         O: O41 (watching-command-process)
   279:         O: O41 (watching-command-process)
   280:         O: O41 (watching-command-process)
   281:         O: O41 (watching-command-process)
   282:         O: O41 (watching-command-process)
   283:         O: O41 (watching-command-process)
   284:         O: O41 (watching-command-process)
   285:         O: O41 (watching-command-process)
   286:      O: O45 (there-are-planes)
  There are planes to land.
   287:      O: O48 (look-scope)
write successful socket 3

   288:      ==>G: G10 (operator no-change)
   289:         P: P1 (atc)
   290:         S: S1
   291:         O: O51 (watching-command-process)
   292:         O: O51 (watching-command-process)
   293:         O: O51 (watching-command-process)
   294:         O: O51 (watching-command-process)
   295:         O: O51 (watching-command-process)
```

```
    296:         O: O51 (watching-command-process)
    297:         O: O51 (watching-command-process)
    298:         O: O51 (watching-command-process)
    299:         O: O51 (watching-command-process)
    300:         O: O51 (watching-command-process)
    301:         O: O51 (watching-command-process)
    302:         O: O51 (watching-command-process)
    303:         O: O51 (watching-command-process)
    304:         O: O51 (watching-command-process)
    305:         O: O51 (watching-command-process)
    306:         O: O51 (watching-command-process)
    307:      O: O58 (lower-altitude)
  Flight number cx901 has reached First approach fix.
  Descending to 5000 meters.
write successful socket 3

    308:      ==>G: G11 (operator no-change)
    309:         P: P1 (atc)
    310:         S: S1
    311:         O: O61 (watching-command-process)
    312:         O: O61 (watching-command-process)
    313:         O: O61 (watching-command-process)
    314:         O: O61 (watching-command-process)
    315:         O: O61 (watching-command-process)
    316:         O: O61 (watching-command-process)
    317:         O: O61 (watching-command-process)
    318:         O: O61 (watching-command-process)
    319:         O: O61 (watching-command-process)
    320:         O: O61 (watching-command-process)
    321:         O: O61 (watching-command-process)
    322:         O: O61 (watching-command-process)
    323:         O: O61 (watching-command-process)
    324:         O: O61 (watching-command-process)
    325:         O: O61 (watching-command-process)
    326:         O: O61 (watching-command-process)
    327:         O: O61 (watching-command-process)
    328:      O: O60 (look-scope)
write successful socket 3

    329:      ==>G: G12 (operator no-change)
    330:         P: P1 (atc)
    331:         S: S1
    332:         ==>G: G13 (operator tie)
    333:            P: P13 (selection)
    334:            S: S3
    335:            O: O70 (evaluate-object O66 (lower-altitude))
    336:            ==>G: G14 (operator no-change)
    337:               P: P1 (atc)
    338:               S: D5
    339:               O: C11 (lower-altitude)
  Flight number cx901 has reached First approach fix.
  Descending to 5000 meters.
    340:              ==>G: G15 (operator no-change)
    341:                  P: P1 (atc)
    342:                  S: D5
    343:                  O: O74 (watching-command-process)
    344:                  O: O74 (watching-command-process)
    345:                  O: O74 (watching-command-process)
    346:                  O: O74 (watching-command-process)
    347:                  O: O74 (watching-command-process)
    348:                  O: O74 (watching-command-process)
```

```
   349:    O: O76 (there-are-planes)
  There are planes to land.
   350:    O: O79 (look-scope)
write successful socket 3

   351:    ==>G: G16 (operator no-change)
   352:       P: P1 (atc)
   353:       S: S1
   354:       O: O82 (watching-command-process)
   355:       O: O82 (watching-command-process)
   356:       O: O82 (watching-command-process)
   357:       O: O82 (watching-command-process)
   358:       O: O82 (watching-command-process)
   359:       O: O82 (watching-command-process)
   360:       O: O82 (watching-command-process)
   361:       O: O82 (watching-command-process)
   362:       O: O82 (watching-command-process)
   363:       O: O82 (watching-command-process)
   364:       O: O82 (watching-command-process)
   365:       O: O82 (watching-command-process)
   366:       O: O82 (watching-command-process)
   367:       O: O82 (watching-command-process)
   368:       O: O82 (watching-command-process)
   369:       O: O82 (watching-command-process)
   370:       O: O82 (watching-command-process)
   371:    O: O86 (there-are-planes)
  There are planes to land.
   372:    O: O89 (look-scope)
write successful socket 3

   373:    ==>G: G17 (operator no-change)
   374:       P: P1 (atc)
   375:       S: S1
   376:       O: O92 (watching-command-process)
   377:       O: O92 (watching-command-process)
   378:       O: O92 (watching-command-process)
   379:       O: O92 (watching-command-process)
   380:       O: O92 (watching-command-process)
   381:       O: O92 (watching-command-process)
   382:       O: O92 (watching-command-process)
   383:       O: O92 (watching-command-process)
   384:       O: O92 (watching-command-process)
   385:       O: O92 (watching-command-process)
   386:       O: O92 (watching-command-process)
   387:       O: O92 (watching-command-process)
   388:       O: O92 (watching-command-process)
   389:       O: O92 (watching-command-process)
   390:       O: O92 (watching-command-process)
   391:       O: O92 (watching-command-process)
   392:       O: O92 (watching-command-process)
   393:    O: O96 (there-are-planes)
  There are planes to land.
   394:    O: O99 (look-scope)
write successful socket 3

   395:    ==>G: G18 (operator no-change)
   396:       P: P1 (atc)
   397:       S: S1
   398:       O: O102 (watching-command-process)
   399:       O: O102 (watching-command-process)
   400:       O: O102 (watching-command-process)
```

```
    401:       O: O102 (watching-command-process)
    402:       O: O102 (watching-command-process)
    403:       O: O102 (watching-command-process)
    404:       O: O102 (watching-command-process)
    405:       O: O102 (watching-command-process)
    406:       O: O102 (watching-command-process)
    407:       O: O102 (watching-command-process)
    408:       O: O102 (watching-command-process)
    409:       O: O102 (watching-command-process)
    410:       O: O102 (watching-command-process)
    411:       O: O102 (watching-command-process)
    412:       O: O102 (watching-command-process)
    413:       O: O102 (watching-command-process)
    414:       O: O102 (watching-command-process)
    415:    O: O106 (there-are-planes)
  There are planes to land.
    416:    O: O109 (look-scope)
write successful socket 3

    417:    ==>G: G19 (operator no-change)
    418:       P: P1 (atc)
    419:       S: S1
    420:       O: O112 (watching-command-process)
    421:       O: O112 (watching-command-process)
    422:       O: O112 (watching-command-process)
    423:       O: O112 (watching-command-process)
    424:       O: O112 (watching-command-process)
    425:       O: O112 (watching-command-process)
    426:       O: O112 (watching-command-process)
    427:       O: O112 (watching-command-process)
    428:       O: O112 (watching-command-process)
    429:       O: O112 (watching-command-process)
    430:       O: O112 (watching-command-process)
    431:       O: O112 (watching-command-process)
    432:       O: O112 (watching-command-process)
    433:       O: O112 (watching-command-process)
    434:       O: O112 (watching-command-process)
    435:       O: O112 (watching-command-process)
    436:    O: O116 (there-are-planes)
  There are planes to land.
    437:    O: O119 (look-scope)
write successful socket 3

    438:    ==>G: G20 (operator no-change)
    439:       P: P1 (atc)
    440:       S: S1
    441:       O: O122 (watching-command-process)
    442:       O: O122 (watching-command-process)
    443:       O: O122 (watching-command-process)
    444:       O: O122 (watching-command-process)
    445:       O: O122 (watching-command-process)
    446:       O: O122 (watching-command-process)
    447:       O: O122 (watching-command-process)
    448:       O: O122 (watching-command-process)
    449:       O: O122 (watching-command-process)
    450:       O: O122 (watching-command-process)
    451:       O: O122 (watching-command-process)
    452:       O: O122 (watching-command-process)
    453:       O: O122 (watching-command-process)
    454:       O: O122 (watching-command-process)
    455:       O: O122 (watching-command-process)
    456:        O: O122 (watching-command-process)
    457:        O: O122 (watching-command-process)
    458:     O: O126 (there-are-planes)
  There are planes to land.
    459:     O: O129 (look-scope)
write successful socket 3

    460:     ==>G: G21 (operator no-change)
    461:        P: P1 (atc)
    462:        S: S1
    463:        O: O132 (watching-command-process)
    464:        O: O132 (watching-command-process)
    465:        O: O132 (watching-command-process)
    466:        O: O132 (watching-command-process)
    467:        O: O132 (watching-command-process)
    468:        O: O132 (watching-command-process)
    469:        O: O132 (watching-command-process)
    470:        O: O132 (watching-command-process)
    471:        O: O132 (watching-command-process)
    472:        O: O132 (watching-command-process)
    473:        O: O132 (watching-command-process)
    474:        O: O132 (watching-command-process)
    475:        O: O132 (watching-command-process)
    476:        O: O132 (watching-command-process)
    477:        O: O132 (watching-command-process)
    478:        O: O132 (watching-command-process)
    479:     O: O136 (there-are-planes)
  There are planes to land.
    480:     O: O139 (look-scope)
write successful socket 3

    481:     ==>G: G22 (operator no-change)
    482:        P: P1 (atc)
    483:        S: S1
    484:        O: O142 (watching-command-process)
    485:        O: O142 (watching-command-process)
    486:        O: O142 (watching-command-process)
    487:        O: O142 (watching-command-process)
    488:        O: O142 (watching-command-process)
    489:        O: O142 (watching-command-process)
    490:        O: O142 (watching-command-process)
    491:        O: O142 (watching-command-process)
    492:        O: O142 (watching-command-process)
    493:        O: O142 (watching-command-process)
    494:        O: O142 (watching-command-process)
    495:        O: O142 (watching-command-process)
    496:        O: O142 (watching-command-process)
    497:        O: O142 (watching-command-process)
    498:        O: O142 (watching-command-process)
    499:        O: O142 (watching-command-process)
    500:        O: O142 (watching-command-process)
    501:        O: O142 (watching-command-process)
    502:        O: O142 (watching-command-process)
    503:        O: O142 (watching-command-process)
    504:        O: O142 (watching-command-process)
    505:        O: O142 (watching-command-process)
    506:        O: O142 (watching-command-process)
    507:        O: O142 (watching-command-process)
    508:        O: O142 (watching-command-process)
    509:        O: O142 (watching-command-process)
    510:        O: O142 (watching-command-process)
    511:      O: O142 (watching-command-process)
    512:      O: O142 (watching-command-process)
    513:      O: O142 (watching-command-process)
    514:      O: O142 (watching-command-process)
    515:      O: O142 (watching-command-process)
    516:      O: O142 (watching-command-process)
    517:      O: O142 (watching-command-process)
    518:      O: O142 (watching-command-process)
    519:      O: O142 (watching-command-process)
    520:      O: O142 (watching-command-process)
    521:      O: O142 (watching-command-process)
    522:      O: O142 (watching-command-process)
    523:      O: O142 (watching-command-process)
    524:      O: O142 (watching-command-process)
    525:      O: O142 (watching-command-process)
    526:      O: O142 (watching-command-process)
    527:      O: O142 (watching-command-process)
    528:      O: O142 (watching-command-process)
    529:      O: O142 (watching-command-process)
    530:      O: O142 (watching-command-process)
    531:      O: O142 (watching-command-process)
    532:      O: O142 (watching-command-process)
    533:      O: O142 (watching-command-process)
    534:      O: O142 (watching-command-process)
    535:      O: O142 (watching-command-process)
    536:      O: O142 (watching-command-process)
    537:      O: O142 (watching-command-process)
    538:      O: O142 (watching-command-process)
    539:      O: O142 (watching-command-process)
    540:      O: O142 (watching-command-process)
    541:      O: O142 (watching-command-process)
    542:   O: O146 (there-are-planes)
  There are planes to land.
    543:   O: O149 (look-scope)
write successful socket 3

    544:   ==>G: G23 (operator no-change)
    545:      P: P1 (atc)
    546:      S: S1
    547:      O: O152 (watching-command-process)
    548:      O: O152 (watching-command-process)
    549:      O: O152 (watching-command-process)
    550:      O: O152 (watching-command-process)
    551:      O: O152 (watching-command-process)
    552:      O: O152 (watching-command-process)
    553:      O: O152 (watching-command-process)
    554:      O: O152 (watching-command-process)
    555:      O: O152 (watching-command-process)
    556:      O: O152 (watching-command-process)
    557:      O: O152 (watching-command-process)
    558:      O: O152 (watching-command-process)
    559:      O: O152 (watching-command-process)
    560:      O: O152 (watching-command-process)
    561:      O: O152 (watching-command-process)
    562:      O: O152 (watching-command-process)
    563:      O: O152 (watching-command-process)
    564:   O: O156 (there-are-planes)
  There are planes to land.
    565:   O: O159 (look-scope)
write successful socket 3
```

```
   566:     ==>G: G24 (operator no-change)
               P: P1 (atc)
               S: S1
               O: O162 (watching-command-process)
               O: O162 (watching-command-process)
               O: O162 (watching-command-process)
               O: O162 (watching-command-process)
               O: O162 (watching-command-process)
               O: O162 (watching-command-process)
               O: O162 (watching-command-process)
               O: O162 (watching-command-process)
               O: O162 (watching-command-process)
               O: O162 (watching-command-process)
               O: O162 (watching-command-process)
               O: O162 (watching-command-process)
               O: O162 (watching-command-process)
               O: O162 (watching-command-process)
               O: O162 (watching-command-process)
               O: O162 (watching-command-process)
   585:     O: O166 (there-are-planes)
  There are planes to land.
   586:     O: O169 (look-scope)
write successful socket 3

   587:     ==>G: G25 (operator no-change)
   588:        P: P1 (atc)
   589:        S: S1
   590:        O: O172 (watching-command-process)
   591:        O: O172 (watching-command-process)
   592:        O: O172 (watching-command-process)
   593:        O: O172 (watching-command-process)
   594:        O: O172 (watching-command-process)
   595:        O: O172 (watching-command-process)
   596:        O: O172 (watching-command-process)
   597:        O: O172 (watching-command-process)
   598:        O: O172 (watching-command-process)
   599:        O: O172 (watching-command-process)
   600:        O: O172 (watching-command-process)
   601:        O: O172 (watching-command-process)
   602:        O: O172 (watching-command-process)
   603:        O: O172 (watching-command-process)
   604:        O: O172 (watching-command-process)
   605:        O: O172 (watching-command-process)
   606:        O: O172 (watching-command-process)
   607:     O: O176 (there-are-planes)
  There are planes to land.
   608:     O: O179 (look-scope)
write successful socket 3

   609:     ==>G: G26 (operator no-change)
   610:        P: P1 (atc)
   611:        S: S1
   612:        O: O182 (watching-command-process)
   613:        O: O182 (watching-command-process)
   614:        O: O182 (watching-command-process)
   615:        O: O182 (watching-command-process)
   616:        O: O182 (watching-command-process)
   617:        O: O182 (watching-command-process)
   618:        O: O182 (watching-command-process)
   619:        O: O182 (watching-command-process)
   620:        O: O182 (watching-command-process)
   621:         O: O182 (watching-command-process)
   622:         O: O182 (watching-command-process)
   623:         O: O182 (watching-command-process)
   624:         O: O182 (watching-command-process)
   625:         O: O182 (watching-command-process)
   626:         O: O182 (watching-command-process)
   627:         O: O182 (watching-command-process)
   628:      O: O186 (there-are-planes)
  There are planes to land.
   629:      O: O189 (look-scope)
write successful socket 3

   630:      ==>G: G27 (operator no-change)
   631:         P: P1 (atc)
   632:         S: S1
   633:         O: O192 (watching-command-process)
   634:         O: O192 (watching-command-process)
   635:         O: O192 (watching-command-process)
   636:         O: O192 (watching-command-process)
   637:         O: O192 (watching-command-process)
   638:         O: O192 (watching-command-process)
   639:         O: O192 (watching-command-process)
   640:         O: O192 (watching-command-process)
   641:         O: O192 (watching-command-process)
   642:         O: O192 (watching-command-process)
   643:         O: O192 (watching-command-process)
   644:         O: O192 (watching-command-process)
   645:         O: O192 (watching-command-process)
   646:         O: O192 (watching-command-process)
   647:         O: O192 (watching-command-process)
   648:         O: O192 (watching-command-process)
   649:         O: O192 (watching-command-process)
   650:      O: O196 (there-are-planes)
  There are planes to land.
   651:      O: O199 (look-scope)
write successful socket 3

   652:      ==>G: G28 (operator no-change)
   653:         P: P1 (atc)
   654:         S: S1
   655:         O: O202 (watching-command-process)
   656:         O: O202 (watching-command-process)
   657:         O: O202 (watching-command-process)
   658:         O: O202 (watching-command-process)
   659:         O: O202 (watching-command-process)
   660:         O: O202 (watching-command-process)
   661:         O: O202 (watching-command-process)
   662:         O: O202 (watching-command-process)
   663:         O: O202 (watching-command-process)
   664:         O: O202 (watching-command-process)
   665:         O: O202 (watching-command-process)
   666:         O: O202 (watching-command-process)
   667:         O: O202 (watching-command-process)
   668:         O: O202 (watching-command-process)
   669:         O: O202 (watching-command-process)
   670:         O: O202 (watching-command-process)
   671:         O: O202 (watching-command-process)
   672:      O: O206 (there-are-planes)
  There are planes to land.
   673:      O: O209 (look-scope)
write successful socket 3
```

```
    674:    ==>G: G29 (operator no-change)
               P: P1 (atc)
    675:       S: S1
    676:       O: O212 (watching-command-process)
    677:       O: O212 (watching-command-process)
    678:       O: O212 (watching-command-process)
    679:       O: O212 (watching-command-process)
    680:       O: O212 (watching-command-process)
    681:       O: O212 (watching-command-process)
    682:       O: O212 (watching-command-process)
    683:       O: O212 (watching-command-process)
    684:       O: O212 (watching-command-process)
    685:       O: O212 (watching-command-process)
    686:       O: O212 (watching-command-process)
    687:       O: O212 (watching-command-process)
    688:       O: O212 (watching-command-process)
    689:       O: O212 (watching-command-process)
    690:       O: O212 (watching-command-process)
    691:       O: O212 (watching-command-process)
    692:       O: O212 (watching-command-process)
    693:    O: O219 (land-plane)
  Flight number pr301 has reached Final approach fix.
  Handing to Approach controller.
write successful socket 3

    694:    ==>G: G30 (operator no-change)
    695:       P: P1 (atc)
    696:       S: S1
    697:       O: O222 (watching-command-process)
    698:       O: O222 (watching-command-process)
    699:       O: O222 (watching-command-process)
    700:       O: O222 (watching-command-process)
    701:       O: O222 (watching-command-process)
    702:       O: O222 (watching-command-process)
    703:       O: O222 (watching-command-process)
    704:       O: O222 (watching-command-process)
    705:       O: O222 (watching-command-process)
    706:       O: O222 (watching-command-process)
    707:       O: O222 (watching-command-process)
    708:       O: O222 (watching-command-process)
    709:       O: O222 (watching-command-process)
    710:       O: O222 (watching-command-process)
    711:       O: O222 (watching-command-process)
    712:       O: O222 (watching-command-process)
    713:    O: O221 (look-scope)
write successful socket 3

    714:    ==>G: G31 (operator no-change)
    715:       P: P1 (atc)
    716:       S: S1
    717:       ==>G: G32 (operator tie)
    718:          P: P44 (selection)
    719:          S: S4
    720:          O: O231 (evaluate-object O227 (land-plane))
    721:          ==>G: G33 (operator no-change)
    722:             P: P1 (atc)
    723:             S: D7
    724:             O: C29 (land-plane)
  Flight number pr301 has reached Final approach fix.
  Handing to Approach controller.
    725:             ==>G: G34 (operator no-change)
    726:                P: P1 (atc)
```

```
   727:                    S: D7
   728:                    O: O235 (watching-command-process)
   729:                    O: O235 (watching-command-process)
   730:                    O: O235 (watching-command-process)
   731:                    O: O235 (watching-command-process)
   732:                    O: O235 (watching-command-process)
   733:                    O: O235 (watching-command-process)
   734:    O: O236 (there-are-planes)
  There are planes to land.
   735:    O: O240 (look-scope)
write successful socket 3

   736:    ==>G: G35 (operator no-change)
   737:       P: P1 (atc)
   738:       S: S1
   739:       O: O243 (watching-command-process)
   740:       O: O243 (watching-command-process)
   741:       O: O243 (watching-command-process)
   742:       O: O243 (watching-command-process)
   743:       O: O243 (watching-command-process)
   744:       O: O243 (watching-command-process)
   745:       O: O243 (watching-command-process)
   746:       O: O243 (watching-command-process)
   747:       O: O243 (watching-command-process)
   748:       O: O243 (watching-command-process)
   749:       O: O243 (watching-command-process)
   750:       O: O243 (watching-command-process)
   751:       O: O243 (watching-command-process)
   752:       O: O243 (watching-command-process)
   753:       O: O243 (watching-command-process)
   754:       O: O243 (watching-command-process)
   755:       O: O243 (watching-command-process)
   756:    O: O250 (land-plane)
  Flight number cx901 has reached Final approach fix.
  Handing to Approach controller.
write successful socket 3

   757:    ==>G: G36 (operator no-change)
   758:       P: P1 (atc)
   759:       S: S1
   760:       O: O253 (watching-command-process)
   761:       O: O253 (watching-command-process)
   762:       O: O253 (watching-command-process)
   763:       O: O253 (watching-command-process)
   764:       O: O253 (watching-command-process)
   765:       O: O253 (watching-command-process)
   766:       O: O253 (watching-command-process)
   767:       O: O253 (watching-command-process)
   768:       O: O253 (watching-command-process)
   769:       O: O253 (watching-command-process)
   770:       O: O253 (watching-command-process)
   771:       O: O253 (watching-command-process)
   772:       O: O253 (watching-command-process)
   773:       O: O253 (watching-command-process)
   774:       O: O253 (watching-command-process)
   775:       O: O253 (watching-command-process)
   776:    O: O252 (look-scope)
write successful socket 3

   777:    ==>G: G37 (operator no-change)
   778:       P: P1 (atc)
```

```
    779:       S: S1
    780:       ==>G: G38 (operator tie)
    781:          P: P49 (selection)
    782:          S: S5
    783:          O: O262 (evaluate-object O258 (land-plane))
    784:          ==>G: G39 (operator no-change)
    785:             P: P1 (atc)
    786:             S: D9
    787:             O: C34 (land-plane)
   Flight number cx901 has reached Final approach fix.
   Handing to Approach controller.
    788:             ==>G: G40 (operator no-change)
    789:                P: P1 (atc)
    790:                S: D9
    791:                O: O266 (watching-command-process)
    792:                O: O266 (watching-command-process)
    793:                O: O266 (watching-command-process)
    794:                O: O266 (watching-command-process)
    795:                O: O266 (watching-command-process)
    796:                O: O266 (watching-command-process)
    797:    O: O268 (no-more-planes-to-land)
   There are no more planes to land.
  goal G1 achieved
  goal atc achieved
   Goal atc succeeded.
System halted.
```

D.2 Sample Run 2 (Thu Sep 15 13:01:56 1994)

```
Soar 6.2.3

Bugs and questions should be sent to soar-bugs@cs.cmu.edu
The current bug-list may be obtained by sending mail to
soar-bugs@cs.cmu.edu with the Subject: line "bug list".

This software is in the public domain, and is made available AS IS.
Carnegie Mellon University, The University of Michigan, and
The University of Southern California/Information Sciences Institute
make no warranties about the software or its performance, implied
or otherwise.

Type "help" for information on various topics.
Type "quit" to exit.  Use ctrl-c to stop a Soar run.
Type "soarnews" for news.
Type "version" for complete version information.

Loading /psyc/teaching/UG/otherug/itxro/.init.soar


Loading
/psyc/teaching/UG/otherug/itxro/soar6.2.3/default/non-nnpscm/default.soar
****************************************************************************
********************************

Loading /psyc/teaching/UG/otherug/itxro/garnet/ATC.soar
*********************************

Soar> init-socket-io upsyc 3296
Created Socket with id 4
Connecting to server with name upsyc
Connecting to server with port #3296
connected to socket 4
Soar> d

     0: ==>G: G1
     1:    P: P1 (atc)
     2:    S: S1
     3:    O: O1 (look-scope)
write successful socket 4

     4:    ==>G: G2 (operator no-change)
     5:       P: P1 (atc)
     6:       S: S1
     7:       O: O2 (watching-command-process)
     8:       O: O2 (watching-command-process)
     9:       O: O2 (watching-command-process)
    10:       O: O2 (watching-command-process)
    11:       O: O2 (watching-command-process)
    12:       O: O2 (watching-command-process)
    13:       O: O2 (watching-command-process)
    14:       O: O2 (watching-command-process)
    15:       O: O2 (watching-command-process)
    16:       O: O2 (watching-command-process)
    17:       O: O2 (watching-command-process)
    18:       O: O2 (watching-command-process)
    19:       O: O2 (watching-command-process)
    20:       O: O2 (watching-command-process)
    21:       O: O2 (watching-command-process)
    22:       O: O2 (watching-command-process)
```

```
   23:         O: O2 (watching-command-process)
   24:         O: O2 (watching-command-process)
   25:         O: O2 (watching-command-process)
   26:         O: O2 (watching-command-process)
   27:         O: O2 (watching-command-process)
   28:         O: O2 (watching-command-process)
   29:         O: O2 (watching-command-process)
   30:         O: O2 (watching-command-process)
   31:         O: O2 (watching-command-process)
   32:         O: O2 (watching-command-process)
   33:         O: O2 (watching-command-process)
   34:         O: O2 (watching-command-process)
   35:         O: O2 (watching-command-process)
   36:         O: O2 (watching-command-process)
   37:         O: O2 (watching-command-process)
   38:         O: O2 (watching-command-process)
   39:         O: O2 (watching-command-process)
   40:         O: O2 (watching-command-process)
   41:         O: O2 (watching-command-process)
   42:         O: O2 (watching-command-process)
   43:         O: O2 (watching-command-process)
   44:         O: O2 (watching-command-process)
   45:         O: O2 (watching-command-process)
   46:         O: O2 (watching-command-process)
   47:         O: O2 (watching-command-process)
   48:         O: O2 (watching-command-process)
   49:         O: O2 (watching-command-process)
   50:         O: O2 (watching-command-process)
   51:         O: O2 (watching-command-process)
   52:         O: O2 (watching-command-process)
   53:         O: O2 (watching-command-process)
   54:         O: O2 (watching-command-process)
   55:         O: O2 (watching-command-process)
   56:         O: O2 (watching-command-process)
   57:         O: O2 (watching-command-process)
   58:         O: O2 (watching-command-process)
   59:         O: O2 (watching-command-process)
   60:         O: O2 (watching-command-process)
   61:         O: O2 (watching-command-process)
   62:         O: O2 (watching-command-process)
   63:         O: O2 (watching-command-process)
   64:         O: O2 (watching-command-process)
   65:         O: O2 (watching-command-process)
   66:         O: O2 (watching-command-process)
   67:         O: O2 (watching-command-process)
   68:         O: O2 (watching-command-process)
   69:         O: O2 (watching-command-process)
   70:         O: O2 (watching-command-process)
   71:         O: O2 (watching-command-process)
   72:         O: O2 (watching-command-process)
   73:         O: O2 (watching-command-process)
   74:         O: O2 (watching-command-process)
   75:         O: O2 (watching-command-process)
   76:         O: O2 (watching-command-process)
   77:         O: O2 (watching-command-process)
   78:         O: O2 (watching-command-process)
   79:         O: O2 (watching-command-process)
   80:         O: O2 (watching-command-process)
   81:         O: O2 (watching-command-process)
   82:         O: O2 (watching-command-process)
   83:         O: O2 (watching-command-process)
```

```
 84:        O: O2 (watching-command-process)
 85:        O: O2 (watching-command-process)
 86:        O: O2 (watching-command-process)
 87:        O: O2 (watching-command-process)
 88:        O: O2 (watching-command-process)
 89:        O: O2 (watching-command-process)
 90:        O: O2 (watching-command-process)
 91:        O: O2 (watching-command-process)
 92:        O: O2 (watching-command-process)
 93:        O: O2 (watching-command-process)
 94:        O: O2 (watching-command-process)
 95:        O: O2 (watching-command-process)
 96:        O: O2 (watching-command-process)
 97:        O: O2 (watching-command-process)
 98:        O: O2 (watching-command-process)
 99:        O: O2 (watching-command-process)
100:        O: O2 (watching-command-process)
101:        O: O2 (watching-command-process)
102:        O: O2 (watching-command-process)
103:        O: O2 (watching-command-process)
104:        O: O2 (watching-command-process)
105:        O: O2 (watching-command-process)
106:        O: O2 (watching-command-process)
107:        O: O2 (watching-command-process)
108:        O: O2 (watching-command-process)
109:        O: O2 (watching-command-process)
110:        O: O2 (watching-command-process)
111:        O: O2 (watching-command-process)
112:        O: O2 (watching-command-process)
113:        O: O2 (watching-command-process)
114:        O: O2 (watching-command-process)
115:        O: O2 (watching-command-process)
116:        O: O2 (watching-command-process)
117:        O: O2 (watching-command-process)
118:        O: O2 (watching-command-process)
119:        O: O2 (watching-command-process)
120:        O: O2 (watching-command-process)
121:        O: O2 (watching-command-process)
122:        O: O2 (watching-command-process)
123:        O: O2 (watching-command-process)
124:        O: O2 (watching-command-process)
125:        O: O2 (watching-command-process)
126:        O: O2 (watching-command-process)
127:        O: O2 (watching-command-process)
128:        O: O2 (watching-command-process)
129:        O: O2 (watching-command-process)
130:        O: O2 (watching-command-process)
131:        O: O2 (watching-command-process)
132:        O: O2 (watching-command-process)
133:        O: O2 (watching-command-process)
134:        O: O2 (watching-command-process)
135:        O: O2 (watching-command-process)
136:        O: O2 (watching-command-process)
137:        O: O2 (watching-command-process)
138:        O: O2 (watching-command-process)
139:        O: O2 (watching-command-process)
140:        O: O2 (watching-command-process)
141:        O: O2 (watching-command-process)
142:        O: O2 (watching-command-process)
143:        O: O2 (watching-command-process)
144:        O: O2 (watching-command-process)
```

```
    145:        O: O2 (watching-command-process)
    146:        O: O2 (watching-command-process)
    147:        O: O2 (watching-command-process)
    148:        O: O2 (watching-command-process)
    149:        O: O2 (watching-command-process)
    150:        O: O2 (watching-command-process)
    151:        O: O2 (watching-command-process)
    152:        O: O2 (watching-command-process)
    153:        O: O2 (watching-command-process)
    154:        O: O2 (watching-command-process)
    155:        O: O2 (watching-command-process)
    156:        O: O2 (watching-command-process)
    157:        O: O2 (watching-command-process)
    158:        O: O2 (watching-command-process)
    159:        O: O2 (watching-command-process)
    160:        O: O2 (watching-command-process)
    161:        O: O2 (watching-command-process)
    162:        O: O2 (watching-command-process)
    163:        O: O2 (watching-command-process)
    164:        O: O2 (watching-command-process)
    165:        O: O2 (watching-command-process)
    166:        O: O2 (watching-command-process)
    167:        O: O2 (watching-command-process)
    168:        O: O2 (watching-command-process)
    169:        O: O2 (watching-command-process)
    170:        O: O2 (watching-command-process)
    171:        O: O2 (watching-command-process)
    172:        O: O2 (watching-command-process)
    173:        O: O2 (watching-command-process)
    174:        O: O2 (watching-command-process)
    175:        O: O2 (watching-command-process)
    176:    O: O7 (lower-altitude)
  Flight number pr301 has reached First approach fix.
  Descending to 5000 meters.
write successful socket 4

    177:    ==>G: G3 (operator no-change)
    178:        P: P1 (atc)
    179:        S: S1
    180:        O: O10 (watching-command-process)
    181:        O: O10 (watching-command-process)
    182:        O: O10 (watching-command-process)
    183:        O: O10 (watching-command-process)
    184:        O: O10 (watching-command-process)
    185:        O: O10 (watching-command-process)
    186:        O: O10 (watching-command-process)
    187:        O: O10 (watching-command-process)
    188:        O: O10 (watching-command-process)
    189:        O: O10 (watching-command-process)
    190:        O: O10 (watching-command-process)
    191:        O: O10 (watching-command-process)
    192:        O: O10 (watching-command-process)
    193:        O: O10 (watching-command-process)
    194:        O: O10 (watching-command-process)
    195:        O: O10 (watching-command-process)
    196:    O: O9 (look-scope)
write successful socket 4

    197:    ==>G: G4 (operator no-change)
    198:        P: P1 (atc)
    199:        S: S1
    200:        O: O14 (watching-command-process)
    201:        O: O14 (watching-command-process)
    202:        O: O14 (watching-command-process)
    203:        O: O14 (watching-command-process)
    204:        O: O14 (watching-command-process)
    205:        O: O14 (watching-command-process)
    206:        O: O14 (watching-command-process)
    207:        O: O14 (watching-command-process)
    208:        O: O14 (watching-command-process)
    209:        O: O14 (watching-command-process)
    210:        O: O14 (watching-command-process)
    211:        O: O14 (watching-command-process)
    212:        O: O14 (watching-command-process)
    213:        O: O14 (watching-command-process)
    214:        O: O14 (watching-command-process)
    215:        O: O14 (watching-command-process)
    216:        O: O14 (watching-command-process)
    217:        O: O14 (watching-command-process)
    218:        O: O14 (watching-command-process)
    219:    O: O19 (there-are-planes)
  There are planes to land.
    220:    O: O22 (look-scope)
write successful socket 4

    221:    ==>G: G5 (operator no-change)
    222:        P: P1 (atc)
    223:        S: S1
    224:        O: O25 (watching-command-process)
    225:        O: O25 (watching-command-process)
    226:        O: O25 (watching-command-process)
    227:        O: O25 (watching-command-process)
    228:        O: O25 (watching-command-process)
    229:        O: O25 (watching-command-process)
    230:        O: O25 (watching-command-process)
    231:        O: O25 (watching-command-process)
    232:        O: O25 (watching-command-process)
    233:        O: O25 (watching-command-process)
    234:        O: O25 (watching-command-process)
    235:        O: O25 (watching-command-process)
    236:        O: O25 (watching-command-process)
    237:        O: O25 (watching-command-process)
    238:        O: O25 (watching-command-process)
    239:        O: O25 (watching-command-process)
    240:        O: O25 (watching-command-process)
    241:    O: O29 (there-are-planes)
  There are planes to land.
    242:    O: O32 (look-scope)
write successful socket 4

    243:    ==>G: G6 (operator no-change)
    244:        P: P1 (atc)
    245:        S: S1
    246:        O: O35 (watching-command-process)
    247:        O: O35 (watching-command-process)
    248:        O: O35 (watching-command-process)
    249:        O: O35 (watching-command-process)
    250:        O: O35 (watching-command-process)
    251:        O: O35 (watching-command-process)
    252:        O: O35 (watching-command-process)
    253:        O: O35 (watching-command-process)
    254:        O: O35 (watching-command-process)
    255:         O: O35 (watching-command-process)
    256:         O: O35 (watching-command-process)
    257:         O: O35 (watching-command-process)
    258:         O: O35 (watching-command-process)
    259:         O: O35 (watching-command-process)
    260:         O: O35 (watching-command-process)
    261:         O: O35 (watching-command-process)
    262:         O: O35 (watching-command-process)
    263:      O: O39 (there-are-planes)
   There are planes to land.
    264:      O: O42 (look-scope)
 write successful socket 4

    265:      ==>G: G7 (operator no-change)
    266:         P: P1 (atc)
    267:         S: S1
    268:         O: O45 (watching-command-process)
    269:         O: O45 (watching-command-process)
    270:         O: O45 (watching-command-process)
    271:         O: O45 (watching-command-process)
    272:         O: O45 (watching-command-process)
    273:         O: O45 (watching-command-process)
    274:         O: O45 (watching-command-process)
    275:         O: O45 (watching-command-process)
    276:         O: O45 (watching-command-process)
    277:         O: O45 (watching-command-process)
    278:         O: O45 (watching-command-process)
    279:         O: O45 (watching-command-process)
    280:         O: O45 (watching-command-process)
    281:         O: O45 (watching-command-process)
    282:         O: O45 (watching-command-process)
    283:         O: O45 (watching-command-process)
    284:         O: O45 (watching-command-process)
    285:      O: O49 (there-are-planes)
   There are planes to land.
    286:      O: O52 (look-scope)
 write successful socket 4

    287:      ==>G: G8 (operator no-change)
    288:         P: P1 (atc)
    289:         S: S1
    290:         O: O55 (watching-command-process)
    291:         O: O55 (watching-command-process)
    292:         O: O55 (watching-command-process)
    293:         O: O55 (watching-command-process)
    294:         O: O55 (watching-command-process)
    295:         O: O55 (watching-command-process)
    296:         O: O55 (watching-command-process)
    297:         O: O55 (watching-command-process)
    298:         O: O55 (watching-command-process)
    299:         O: O55 (watching-command-process)
    300:         O: O55 (watching-command-process)
    301:         O: O55 (watching-command-process)
    302:         O: O55 (watching-command-process)
    303:         O: O55 (watching-command-process)
    304:         O: O55 (watching-command-process)
    305:         O: O55 (watching-command-process)
    306:      O: O62 (lower-altitude)
   Flight number cx901 has reached First approach fix.
   Descending to 5000 meters.
 write successful socket 4
```

```
    307:     ==>G: G9 (operator no-change)
                P: P1 (atc)
                S: S1
                O: O65 (watching-command-process)
                O: O65 (watching-command-process)
                O: O65 (watching-command-process)
                O: O65 (watching-command-process)
                O: O65 (watching-command-process)
                O: O65 (watching-command-process)
                O: O65 (watching-command-process)
                O: O65 (watching-command-process)
                O: O65 (watching-command-process)
                O: O65 (watching-command-process)
                O: O65 (watching-command-process)
                O: O65 (watching-command-process)
                O: O65 (watching-command-process)
                O: O65 (watching-command-process)
                O: O65 (watching-command-process)
                O: O65 (watching-command-process)
                O: O65 (watching-command-process)
             O: O64 (look-scope)
 write successful socket 4

    328:     ==>G: G10 (operator no-change)
                P: P1 (atc)
                S: S1
                O: O69 (watching-command-process)
                O: O69 (watching-command-process)
                O: O69 (watching-command-process)
                O: O69 (watching-command-process)
                O: O69 (watching-command-process)
                O: O69 (watching-command-process)
                O: O69 (watching-command-process)
                O: O69 (watching-command-process)
                O: O69 (watching-command-process)
                O: O69 (watching-command-process)
                O: O69 (watching-command-process)
                O: O69 (watching-command-process)
                O: O69 (watching-command-process)
                O: O69 (watching-command-process)
                O: O69 (watching-command-process)
                O: O69 (watching-command-process)
                O: O69 (watching-command-process)
                O: O69 (watching-command-process)
             O: O74 (there-are-planes)
   There are planes to land.
    350:     O: O77 (look-scope)
 write successful socket 4

    351:     ==>G: G11 (operator no-change)
                P: P1 (atc)
                S: S1
                O: O80 (watching-command-process)
                O: O80 (watching-command-process)
                O: O80 (watching-command-process)
                O: O80 (watching-command-process)
                O: O80 (watching-command-process)
                O: O80 (watching-command-process)
                O: O80 (watching-command-process)
                O: O80 (watching-command-process)
                O: O80 (watching-command-process)
    363:        O: O80 (watching-command-process)
    364:        O: O80 (watching-command-process)
    365:        O: O80 (watching-command-process)
    366:        O: O80 (watching-command-process)
    367:        O: O80 (watching-command-process)
    368:        O: O80 (watching-command-process)
    369:        O: O80 (watching-command-process)
    370:        O: O80 (watching-command-process)
    371:    O: O84 (there-are-planes)
   There are planes to land.
    372:    O: O87 (look-scope)
 write successful socket 4

    373:    ==>G: G12 (operator no-change)
    374:        P: P1 (atc)
    375:        S: S1
    376:        O: O90 (watching-command-process)
    377:        O: O90 (watching-command-process)
    378:        O: O90 (watching-command-process)
    379:        O: O90 (watching-command-process)
    380:        O: O90 (watching-command-process)
    381:        O: O90 (watching-command-process)
    382:        O: O90 (watching-command-process)
    383:        O: O90 (watching-command-process)
    384:        O: O90 (watching-command-process)
    385:        O: O90 (watching-command-process)
    386:        O: O90 (watching-command-process)
    387:        O: O90 (watching-command-process)
    388:        O: O90 (watching-command-process)
    389:        O: O90 (watching-command-process)
    390:        O: O90 (watching-command-process)
    391:        O: O90 (watching-command-process)
    392:        O: O90 (watching-command-process)
    393:    O: O94 (there-are-planes)
   There are planes to land.
    394:    O: O97 (look-scope)
 write successful socket 4

    395:    ==>G: G13 (operator no-change)
    396:        P: P1 (atc)
    397:        S: S1
    398:        O: O100 (watching-command-process)
    399:        O: O100 (watching-command-process)
    400:        O: O100 (watching-command-process)
    401:        O: O100 (watching-command-process)
    402:        O: O100 (watching-command-process)
    403:        O: O100 (watching-command-process)
    404:        O: O100 (watching-command-process)
    405:        O: O100 (watching-command-process)
    406:        O: O100 (watching-command-process)
    407:        O: O100 (watching-command-process)
    408:        O: O100 (watching-command-process)
    409:        O: O100 (watching-command-process)
    410:        O: O100 (watching-command-process)
    411:        O: O100 (watching-command-process)
    412:        O: O100 (watching-command-process)
    413:        O: O100 (watching-command-process)
    414:        O: O100 (watching-command-process)
    415:    O: O104 (there-are-planes)
   There are planes to land.
   416:     O: O107 (look-scope)
 write successful socket 4

   417:     ==>G: G14 (operator no-change)
   418:        P: P1 (atc)
   419:        S: S1
   420:        O: O110 (watching-command-process)
   421:        O: O110 (watching-command-process)
   422:        O: O110 (watching-command-process)
   423:        O: O110 (watching-command-process)
   424:        O: O110 (watching-command-process)
   425:        O: O110 (watching-command-process)
   426:        O: O110 (watching-command-process)
   427:        O: O110 (watching-command-process)
   428:        O: O110 (watching-command-process)
   429:        O: O110 (watching-command-process)
   430:        O: O110 (watching-command-process)
   431:        O: O110 (watching-command-process)
   432:        O: O110 (watching-command-process)
   433:        O: O110 (watching-command-process)
   434:        O: O110 (watching-command-process)
   435:        O: O110 (watching-command-process)
   436:     O: O114 (there-are-planes)
  There are planes to land.
   437:     O: O117 (look-scope)
 write successful socket 4

   438:     ==>G: G15 (operator no-change)
   439:        P: P1 (atc)
   440:        S: S1
   441:        O: O120 (watching-command-process)
   442:        O: O120 (watching-command-process)
   443:        O: O120 (watching-command-process)
   444:        O: O120 (watching-command-process)
   445:        O: O120 (watching-command-process)
   446:        O: O120 (watching-command-process)
   447:        O: O120 (watching-command-process)
   448:        O: O120 (watching-command-process)
   449:        O: O120 (watching-command-process)
   450:        O: O120 (watching-command-process)
   451:        O: O120 (watching-command-process)
   452:        O: O120 (watching-command-process)
   453:        O: O120 (watching-command-process)
   454:        O: O120 (watching-command-process)
   455:        O: O120 (watching-command-process)
   456:        O: O120 (watching-command-process)
   457:        O: O120 (watching-command-process)
   458:     O: O124 (there-are-planes)
  There are planes to land.
   459:     O: O127 (look-scope)
 write successful socket 4

   460:     ==>G: G16 (operator no-change)
   461:        P: P1 (atc)
   462:        S: S1
   463:        O: O130 (watching-command-process)
   464:        O: O130 (watching-command-process)
   465:        O: O130 (watching-command-process)
   466:        O: O130 (watching-command-process)
   467:        O: O130 (watching-command-process)
   468:        O: O130 (watching-command-process)
   469:        O: O130 (watching-command-process)
   470:        O: O130 (watching-command-process)
   471:        O: O130 (watching-command-process)
   472:        O: O130 (watching-command-process)
   473:        O: O130 (watching-command-process)
   474:        O: O130 (watching-command-process)
   475:        O: O130 (watching-command-process)
   476:        O: O130 (watching-command-process)
   477:        O: O130 (watching-command-process)
   478:        O: O130 (watching-command-process)
   479:     O: O134 (there-are-planes)
  There are planes to land.
   480:     O: O137 (look-scope)
write successful socket 4

   481:     ==>G: G17 (operator no-change)
   482:        P: P1 (atc)
   483:        S: S1
   484:        O: O140 (watching-command-process)
   485:        O: O140 (watching-command-process)
   486:        O: O140 (watching-command-process)
   487:        O: O140 (watching-command-process)
   488:        O: O140 (watching-command-process)
   489:        O: O140 (watching-command-process)
   490:        O: O140 (watching-command-process)
   491:        O: O140 (watching-command-process)
   492:        O: O140 (watching-command-process)
   493:        O: O140 (watching-command-process)
   494:        O: O140 (watching-command-process)
   495:        O: O140 (watching-command-process)
   496:        O: O140 (watching-command-process)
   497:        O: O140 (watching-command-process)
   498:        O: O140 (watching-command-process)
   499:        O: O140 (watching-command-process)
   500:        O: O140 (watching-command-process)
   501:     O: O144 (there-are-planes)
  There are planes to land.
   502:     O: O147 (look-scope)
write successful socket 4

   503:     ==>G: G18 (operator no-change)
   504:        P: P1 (atc)
   505:        S: S1
   506:        O: O150 (watching-command-process)
   507:        O: O150 (watching-command-process)
   508:        O: O150 (watching-command-process)
   509:        O: O150 (watching-command-process)
   510:        O: O150 (watching-command-process)
   511:        O: O150 (watching-command-process)
   512:        O: O150 (watching-command-process)
   513:        O: O150 (watching-command-process)
   514:        O: O150 (watching-command-process)
   515:        O: O150 (watching-command-process)
   516:        O: O150 (watching-command-process)
   517:        O: O150 (watching-command-process)
   518:        O: O150 (watching-command-process)
   519:        O: O150 (watching-command-process)
   520:        O: O150 (watching-command-process)
   521:        O: O150 (watching-command-process)
   522:     O: O154 (there-are-planes)
  There are planes to land.
```

```
    523:    O: O157 (look-scope)
 write successful socket 4

    524:    ==>G: G19 (operator no-change)
    525:        P: P1 (atc)
    526:        S: S1
    527:        O: O160 (watching-command-process)
    528:        O: O160 (watching-command-process)
    529:        O: O160 (watching-command-process)
    530:        O: O160 (watching-command-process)
    531:        O: O160 (watching-command-process)
    532:        O: O160 (watching-command-process)
    533:        O: O160 (watching-command-process)
    534:        O: O160 (watching-command-process)
    535:        O: O160 (watching-command-process)
    536:        O: O160 (watching-command-process)
    537:        O: O160 (watching-command-process)
    538:        O: O160 (watching-command-process)
    539:        O: O160 (watching-command-process)
    540:        O: O160 (watching-command-process)
    541:        O: O160 (watching-command-process)
    542:        O: O160 (watching-command-process)
    543:        O: O160 (watching-command-process)
    544:    O: O164 (there-are-planes)
  There are planes to land.
    545:    O: O167 (look-scope)
 write successful socket 4

    546:    ==>G: G20 (operator no-change)
    547:        P: P1 (atc)
    548:        S: S1
    549:        O: O170 (watching-command-process)
    550:        O: O170 (watching-command-process)
    551:        O: O170 (watching-command-process)
    552:        O: O170 (watching-command-process)
    553:        O: O170 (watching-command-process)
    554:        O: O170 (watching-command-process)
    555:        O: O170 (watching-command-process)
    556:        O: O170 (watching-command-process)
    557:        O: O170 (watching-command-process)
    558:        O: O170 (watching-command-process)
    559:        O: O170 (watching-command-process)
    560:        O: O170 (watching-command-process)
    561:        O: O170 (watching-command-process)
    562:        O: O170 (watching-command-process)
    563:        O: O170 (watching-command-process)
    564:        O: O170 (watching-command-process)
    565:    O: O174 (there-are-planes)
  There are planes to land.
    566:    O: O177 (look-scope)
 write successful socket 4

    567:    ==>G: G21 (operator no-change)
    568:        P: P1 (atc)
    569:        S: S1
    570:        O: O180 (watching-command-process)
    571:        O: O180 (watching-command-process)
    572:        O: O180 (watching-command-process)
    573:        O: O180 (watching-command-process)
    574:        O: O180 (watching-command-process)
    575:        O: O180 (watching-command-process)
    576:        O: O180 (watching-command-process)
    577:        O: O180 (watching-command-process)
    578:        O: O180 (watching-command-process)
    579:        O: O180 (watching-command-process)
    580:        O: O180 (watching-command-process)
    581:        O: O180 (watching-command-process)
    582:        O: O180 (watching-command-process)
    583:        O: O180 (watching-command-process)
    584:        O: O180 (watching-command-process)
    585:        O: O180 (watching-command-process)
    586:        O: O180 (watching-command-process)
    587:    O: O184 (there-are-planes)
   There are planes to land.
    588:    O: O187 (look-scope)
 write successful socket 4

    589:    ==>G: G22 (operator no-change)
    590:        P: P1 (atc)
    591:        S: S1
    592:        O: O190 (watching-command-process)
    593:        O: O190 (watching-command-process)
    594:        O: O190 (watching-command-process)
    595:        O: O190 (watching-command-process)
    596:        O: O190 (watching-command-process)
    597:        O: O190 (watching-command-process)
    598:        O: O190 (watching-command-process)
    599:        O: O190 (watching-command-process)
    600:        O: O190 (watching-command-process)
    601:        O: O190 (watching-command-process)
    602:        O: O190 (watching-command-process)
    603:        O: O190 (watching-command-process)
    604:        O: O190 (watching-command-process)
    605:        O: O190 (watching-command-process)
    606:        O: O190 (watching-command-process)
    607:        O: O190 (watching-command-process)
    608:    O: O194 (there-are-planes)
   There are planes to land.
    609:    O: O197 (look-scope)
 write successful socket 4

    610:    ==>G: G23 (operator no-change)
    611:        P: P1 (atc)
    612:        S: S1
    613:        O: O200 (watching-command-process)
    614:        O: O200 (watching-command-process)
    615:        O: O200 (watching-command-process)
    616:        O: O200 (watching-command-process)
    617:        O: O200 (watching-command-process)
    618:        O: O200 (watching-command-process)
    619:        O: O200 (watching-command-process)
    620:        O: O200 (watching-command-process)
    621:        O: O200 (watching-command-process)
    622:        O: O200 (watching-command-process)
    623:        O: O200 (watching-command-process)
    624:        O: O200 (watching-command-process)
    625:        O: O200 (watching-command-process)
    626:        O: O200 (watching-command-process)
    627:        O: O200 (watching-command-process)
    628:        O: O200 (watching-command-process)
    629:        O: O200 (watching-command-process)
   630:    O: O204 (there-are-planes)
  There are planes to land.
   631:    O: O207 (look-scope)
write successful socket 4

   632:    ==>G: G24 (operator no-change)
   633:       P: P1 (atc)
   634:       S: S1
   635:       O: O210 (watching-command-process)
   636:       O: O210 (watching-command-process)
   637:       O: O210 (watching-command-process)
   638:       O: O210 (watching-command-process)
   639:       O: O210 (watching-command-process)
   640:       O: O210 (watching-command-process)
   641:       O: O210 (watching-command-process)
   642:       O: O210 (watching-command-process)
   643:       O: O210 (watching-command-process)
   644:       O: O210 (watching-command-process)
   645:       O: O210 (watching-command-process)
   646:       O: O210 (watching-command-process)
   647:       O: O210 (watching-command-process)
   648:       O: O210 (watching-command-process)
   649:       O: O210 (watching-command-process)
   650:       O: O210 (watching-command-process)
   651:    O: O214 (there-are-planes)
  There are planes to land.
   652:    O: O217 (look-scope)
write successful socket 4

   653:    ==>G: G25 (operator no-change)
   654:       P: P1 (atc)
   655:       S: S1
   656:       O: O220 (watching-command-process)
   657:       O: O220 (watching-command-process)
   658:       O: O220 (watching-command-process)
   659:       O: O220 (watching-command-process)
   660:       O: O220 (watching-command-process)
   661:       O: O220 (watching-command-process)
   662:       O: O220 (watching-command-process)
   663:       O: O220 (watching-command-process)
   664:       O: O220 (watching-command-process)
   665:       O: O220 (watching-command-process)
   666:       O: O220 (watching-command-process)
   667:       O: O220 (watching-command-process)
   668:       O: O220 (watching-command-process)
   669:       O: O220 (watching-command-process)
   670:       O: O220 (watching-command-process)
   671:       O: O220 (watching-command-process)
   672:       O: O220 (watching-command-process)
   673:    O: O224 (there-are-planes)
  There are planes to land.
   674:    O: O227 (look-scope)
write successful socket 4

   675:    ==>G: G26 (operator no-change)
   676:       P: P1 (atc)
   677:       S: S1
   678:       O: O230 (watching-command-process)
   679:       O: O230 (watching-command-process)
   680:       O: O230 (watching-command-process)
   681:       O: O230 (watching-command-process)
```

```
    682:        O: O230 (watching-command-process)
    683:        O: O230 (watching-command-process)
    684:        O: O230 (watching-command-process)
    685:        O: O230 (watching-command-process)
    686:        O: O230 (watching-command-process)
    687:        O: O230 (watching-command-process)
    688:        O: O230 (watching-command-process)
    689:        O: O230 (watching-command-process)
    690:        O: O230 (watching-command-process)
    691:        O: O230 (watching-command-process)
    692:        O: O230 (watching-command-process)
    693:        O: O230 (watching-command-process)
    694:     O: O234 (there-are-planes)
  There are planes to land.
    695:     O: O237 (look-scope)
write successful socket 4

    696:     ==>G: G27 (operator no-change)
    697:        P: P1 (atc)
    698:        S: S1
    699:        O: O240 (watching-command-process)
    700:        O: O240 (watching-command-process)
    701:        O: O240 (watching-command-process)
    702:        O: O240 (watching-command-process)
    703:        O: O240 (watching-command-process)
    704:        O: O240 (watching-command-process)
    705:        O: O240 (watching-command-process)
    706:        O: O240 (watching-command-process)
    707:        O: O240 (watching-command-process)
    708:        O: O240 (watching-command-process)
    709:        O: O240 (watching-command-process)
    710:        O: O240 (watching-command-process)
    711:        O: O240 (watching-command-process)
    712:        O: O240 (watching-command-process)
    713:        O: O240 (watching-command-process)
    714:        O: O240 (watching-command-process)
    715:        O: O240 (watching-command-process)
    716:        O: O240 (watching-command-process)
    717:        O: O240 (watching-command-process)
    718:        O: O240 (watching-command-process)
    719:        O: O240 (watching-command-process)
    720:        O: O240 (watching-command-process)
    721:        O: O240 (watching-command-process)
    722:        O: O240 (watching-command-process)
    723:        O: O240 (watching-command-process)
    724:        O: O240 (watching-command-process)
    725:        O: O240 (watching-command-process)
    726:        O: O240 (watching-command-process)
    727:        O: O240 (watching-command-process)
    728:        O: O240 (watching-command-process)
    729:        O: O240 (watching-command-process)
    730:        O: O240 (watching-command-process)
    731:        O: O240 (watching-command-process)
    732:        O: O240 (watching-command-process)
    733:        O: O240 (watching-command-process)
    734:        O: O240 (watching-command-process)
    735:        O: O240 (watching-command-process)
    736:        O: O240 (watching-command-process)
    737:        O: O240 (watching-command-process)
    738:        O: O240 (watching-command-process)
    739:        O: O240 (watching-command-process)
    740:          O: O240 (watching-command-process)
    741:          O: O240 (watching-command-process)
    742:          O: O240 (watching-command-process)
    743:          O: O240 (watching-command-process)
    744:          O: O240 (watching-command-process)
    745:          O: O240 (watching-command-process)
    746:          O: O240 (watching-command-process)
    747:          O: O240 (watching-command-process)
    748:          O: O240 (watching-command-process)
    749:          O: O240 (watching-command-process)
    750:          O: O240 (watching-command-process)
    751:          O: O240 (watching-command-process)
    752:          O: O240 (watching-command-process)
    753:          O: O240 (watching-command-process)
    754:          O: O240 (watching-command-process)
    755:          O: O240 (watching-command-process)
    756:          O: O240 (watching-command-process)
    757:          O: O240 (watching-command-process)
    758:          O: O240 (watching-command-process)
    759:          O: O240 (watching-command-process)
    760:          O: O240 (watching-command-process)
    761:          O: O240 (watching-command-process)
    762:          O: O240 (watching-command-process)
    763:          O: O240 (watching-command-process)
    764:          O: O240 (watching-command-process)
    765:          O: O240 (watching-command-process)
    766:          O: O240 (watching-command-process)
    767:          O: O240 (watching-command-process)
    768:          O: O240 (watching-command-process)
    769:          O: O240 (watching-command-process)
    770:          O: O240 (watching-command-process)
    771:          O: O240 (watching-command-process)
    772:          O: O240 (watching-command-process)
    773:          O: O240 (watching-command-process)
    774:          O: O240 (watching-command-process)
    775:          O: O240 (watching-command-process)
    776:          O: O240 (watching-command-process)
    777:          O: O240 (watching-command-process)
    778:      O: O244 (there-are-planes)
  There are planes to land.
    779:      O: O247 (look-scope)
write successful socket 4

    780:      ==>G: G28 (operator no-change)
    781:          P: P1 (atc)
    782:          S: S1
    783:          O: O250 (watching-command-process)
    784:          O: O250 (watching-command-process)
    785:          O: O250 (watching-command-process)
    786:          O: O250 (watching-command-process)
    787:          O: O250 (watching-command-process)
    788:          O: O250 (watching-command-process)
    789:          O: O250 (watching-command-process)
    790:          O: O250 (watching-command-process)
    791:          O: O250 (watching-command-process)
    792:          O: O250 (watching-command-process)
    793:          O: O250 (watching-command-process)
    794:          O: O250 (watching-command-process)
    795:          O: O250 (watching-command-process)
    796:          O: O250 (watching-command-process)
    797:          O: O250 (watching-command-process)
```

```
   798:        O: O250 (watching-command-process)
   799:        O: O250 (watching-command-process)
   800:     O: O254 (there-are-planes)
  There are planes to land.
   801:     O: O257 (look-scope)
write successful socket 4

   802:     ==>G: G29 (operator no-change)
   803:        P: P1 (atc)
   804:        S: S1
   805:        O: O260 (watching-command-process)
   806:        O: O260 (watching-command-process)
   807:        O: O260 (watching-command-process)
   808:        O: O260 (watching-command-process)
   809:        O: O260 (watching-command-process)
   810:        O: O260 (watching-command-process)
   811:        O: O260 (watching-command-process)
   812:        O: O260 (watching-command-process)
   813:        O: O260 (watching-command-process)
   814:        O: O260 (watching-command-process)
   815:        O: O260 (watching-command-process)
   816:        O: O260 (watching-command-process)
   817:        O: O260 (watching-command-process)
   818:        O: O260 (watching-command-process)
   819:        O: O260 (watching-command-process)
   820:        O: O260 (watching-command-process)
   821:     O: O264 (there-are-planes)
  There are planes to land.
   822:     O: O267 (look-scope)
write successful socket 4

   823:     ==>G: G30 (operator no-change)
   824:        P: P1 (atc)
   825:        S: S1
   826:        O: O270 (watching-command-process)
   827:        O: O270 (watching-command-process)
   828:        O: O270 (watching-command-process)
   829:        O: O270 (watching-command-process)
   830:        O: O270 (watching-command-process)
   831:        O: O270 (watching-command-process)
   832:        O: O270 (watching-command-process)
   833:        O: O270 (watching-command-process)
   834:        O: O270 (watching-command-process)
   835:        O: O270 (watching-command-process)
   836:        O: O270 (watching-command-process)
   837:        O: O270 (watching-command-process)
   838:        O: O270 (watching-command-process)
   839:        O: O270 (watching-command-process)
   840:        O: O270 (watching-command-process)
   841:        O: O270 (watching-command-process)
   842:        O: O270 (watching-command-process)
   843:     O: O274 (there-are-planes)
  There are planes to land.
   844:     O: O277 (look-scope)
write successful socket 4

   845:     ==>G: G31 (operator no-change)
   846:        P: P1 (atc)
   847:        S: S1
   848:        O: O280 (watching-command-process)
   849:        O: O280 (watching-command-process)
```

```
     850:        O: O280 (watching-command-process)
     851:        O: O280 (watching-command-process)
     852:        O: O280 (watching-command-process)
     853:        O: O280 (watching-command-process)
     854:        O: O280 (watching-command-process)
     855:        O: O280 (watching-command-process)
     856:        O: O280 (watching-command-process)
     857:        O: O280 (watching-command-process)
     858:        O: O280 (watching-command-process)
     859:        O: O280 (watching-command-process)
     860:        O: O280 (watching-command-process)
     861:        O: O280 (watching-command-process)
     862:        O: O280 (watching-command-process)
     863:        O: O280 (watching-command-process)
     864:     O: O284 (there-are-planes)
   There are planes to land.
     865:     O: O287 (look-scope)
write successful socket 4

     866:     ==>G: G32 (operator no-change)
     867:        P: P1 (atc)
     868:        S: S1
     869:        O: O290 (watching-command-process)
     870:        O: O290 (watching-command-process)
     871:        O: O290 (watching-command-process)
     872:        O: O290 (watching-command-process)
     873:        O: O290 (watching-command-process)
     874:        O: O290 (watching-command-process)
     875:        O: O290 (watching-command-process)
     876:        O: O290 (watching-command-process)
     877:        O: O290 (watching-command-process)
     878:        O: O290 (watching-command-process)
     879:        O: O290 (watching-command-process)
     880:        O: O290 (watching-command-process)
     881:        O: O290 (watching-command-process)
     882:        O: O290 (watching-command-process)
     883:        O: O290 (watching-command-process)
     884:        O: O290 (watching-command-process)
     885:        O: O290 (watching-command-process)
     886:     O: O294 (there-are-planes)
   There are planes to land.
     887:     O: O297 (look-scope)
write successful socket 4

     888:     ==>G: G33 (operator no-change)
     889:        P: P1 (atc)
     890:        S: S1
     891:        O: O300 (watching-command-process)
     892:        O: O300 (watching-command-process)
     893:        O: O300 (watching-command-process)
     894:        O: O300 (watching-command-process)
     895:        O: O300 (watching-command-process)
     896:        O: O300 (watching-command-process)
     897:        O: O300 (watching-command-process)
     898:        O: O300 (watching-command-process)
     899:        O: O300 (watching-command-process)
     900:        O: O300 (watching-command-process)
     901:        O: O300 (watching-command-process)
     902:        O: O300 (watching-command-process)
     903:        O: O300 (watching-command-process)
     904:        O: O300 (watching-command-process)
     905:         O: O300 (watching-command-process)
     906:         O: O300 (watching-command-process)
     907:     O: O304 (there-are-planes)
    There are planes to land.
     908:     O: O307 (look-scope)
  write successful socket 4

     909:     ==>G: G34 (operator no-change)
     910:         P: P1 (atc)
     911:         S: S1
     912:         O: O310 (watching-command-process)
     913:         O: O310 (watching-command-process)
     914:         O: O310 (watching-command-process)
     915:         O: O310 (watching-command-process)
     916:         O: O310 (watching-command-process)
     917:         O: O310 (watching-command-process)
     918:         O: O310 (watching-command-process)
     919:         O: O310 (watching-command-process)
     920:         O: O310 (watching-command-process)
     921:         O: O310 (watching-command-process)
     922:         O: O310 (watching-command-process)
     923:         O: O310 (watching-command-process)
     924:         O: O310 (watching-command-process)
     925:         O: O310 (watching-command-process)
     926:         O: O310 (watching-command-process)
     927:         O: O310 (watching-command-process)
     928:         O: O310 (watching-command-process)
     929:     O: O317 (land-plane)
    Flight number pr301 has reached Final approach fix.
    Handing to Approach controller.
  write successful socket 4

     930:     ==>G: G35 (operator no-change)
     931:         P: P1 (atc)
     932:         S: S1
     933:         O: O320 (watching-command-process)
     934:         O: O320 (watching-command-process)
     935:         O: O320 (watching-command-process)
     936:         O: O320 (watching-command-process)
     937:         O: O320 (watching-command-process)
     938:         O: O320 (watching-command-process)
     939:         O: O320 (watching-command-process)
     940:         O: O320 (watching-command-process)
     941:         O: O320 (watching-command-process)
     942:         O: O320 (watching-command-process)
     943:         O: O320 (watching-command-process)
     944:         O: O320 (watching-command-process)
     945:         O: O320 (watching-command-process)
     946:         O: O320 (watching-command-process)
     947:         O: O320 (watching-command-process)
     948:         O: O320 (watching-command-process)
     949:     O: O319 (look-scope)
  write successful socket 4

     950:     ==>G: G36 (operator no-change)
     951:         P: P1 (atc)
     952:         S: S1
     953:         O: O324 (watching-command-process)
     954:         O: O324 (watching-command-process)
     955:         O: O324 (watching-command-process)
     956:         O: O324 (watching-command-process)
     957:         O: O324 (watching-command-process)
     958:         O: O324 (watching-command-process)
     959:         O: O324 (watching-command-process)
     960:         O: O324 (watching-command-process)
     961:         O: O324 (watching-command-process)
     962:         O: O324 (watching-command-process)
     963:         O: O324 (watching-command-process)
     964:         O: O324 (watching-command-process)
     965:         O: O324 (watching-command-process)
     966:         O: O324 (watching-command-process)
     967:         O: O324 (watching-command-process)
     968:         O: O324 (watching-command-process)
     969:         O: O324 (watching-command-process)
     970:         O: O324 (watching-command-process)
     971:      O: O328 (there-are-planes)
   There are planes to land.
     972:      O: O332 (look-scope)
write successful socket 4

     973:      ==>G: G37 (operator no-change)
     974:         P: P1 (atc)
     975:         S: S1
     976:         O: O335 (watching-command-process)
     977:         O: O335 (watching-command-process)
     978:         O: O335 (watching-command-process)
     979:         O: O335 (watching-command-process)
     980:         O: O335 (watching-command-process)
     981:         O: O335 (watching-command-process)
     982:         O: O335 (watching-command-process)
     983:         O: O335 (watching-command-process)
     984:         O: O335 (watching-command-process)
     985:         O: O335 (watching-command-process)
     986:         O: O335 (watching-command-process)
     987:         O: O335 (watching-command-process)
     988:         O: O335 (watching-command-process)
     989:         O: O335 (watching-command-process)
     990:         O: O335 (watching-command-process)
     991:         O: O335 (watching-command-process)
     992:      O: O338 (there-are-planes)
   There are planes to land.
     993:      O: O342 (look-scope)
write successful socket 4

     994:      ==>G: G38 (operator no-change)
     995:         P: P1 (atc)
     996:         S: S1
     997:         O: O345 (watching-command-process)
     998:         O: O345 (watching-command-process)
     999:         O: O345 (watching-command-process)
    1000:         O: O345 (watching-command-process)
    1001:         O: O345 (watching-command-process)
    1002:         O: O345 (watching-command-process)
    1003:         O: O345 (watching-command-process)
    1004:         O: O345 (watching-command-process)
    1005:         O: O345 (watching-command-process)
    1006:         O: O345 (watching-command-process)
    1007:         O: O345 (watching-command-process)
    1008:         O: O345 (watching-command-process)
    1009:         O: O345 (watching-command-process)
    1010:         O: O345 (watching-command-process)
    1011:         O: O345 (watching-command-process)
   1012:        O: O345 (watching-command-process)
   1013:        O: O345 (watching-command-process)
   1014:    O: O352 (land-plane)
   Flight number cx901 has reached Final approach fix.
   Handing to Approach controller.
 write successful socket 4

   1015:    ==>G: G39 (operator no-change)
   1016:        P: P1 (atc)
   1017:        S: S1
   1018:        O: O355 (watching-command-process)
   1019:        O: O355 (watching-command-process)
   1020:        O: O355 (watching-command-process)
   1021:        O: O355 (watching-command-process)
   1022:        O: O355 (watching-command-process)
   1023:        O: O355 (watching-command-process)
   1024:        O: O355 (watching-command-process)
   1025:        O: O355 (watching-command-process)
   1026:        O: O355 (watching-command-process)
   1027:        O: O355 (watching-command-process)
   1028:        O: O355 (watching-command-process)
   1029:        O: O355 (watching-command-process)
   1030:        O: O355 (watching-command-process)
   1031:        O: O355 (watching-command-process)
   1032:        O: O355 (watching-command-process)
   1033:        O: O355 (watching-command-process)
   1034:    O: O354 (look-scope)
 write successful socket 4

   1035:    ==>G: G40 (operator no-change)
   1036:        P: P1 (atc)
   1037:        S: S1
   1038:        O: O359 (watching-command-process)
   1039:        O: O359 (watching-command-process)
   1040:        O: O359 (watching-command-process)
   1041:        O: O359 (watching-command-process)
   1042:        O: O359 (watching-command-process)
   1043:        O: O359 (watching-command-process)
   1044:        O: O359 (watching-command-process)
   1045:        O: O359 (watching-command-process)
   1046:        O: O359 (watching-command-process)
   1047:        O: O359 (watching-command-process)
   1048:        O: O359 (watching-command-process)
   1049:        O: O359 (watching-command-process)
   1050:        O: O359 (watching-command-process)
   1051:        O: O359 (watching-command-process)
   1052:        O: O359 (watching-command-process)
   1053:        O: O359 (watching-command-process)
   1054:        O: O359 (watching-command-process)
   1055:        O: O359 (watching-command-process)
   1056:    O: O364 (no-more-planes-to-land)
   There are no more planes to land.
 goal G1 achieved
 goal atc achieved
   Goal atc succeeded.
System halted.
```

Appendix E

Air Traffic Control (ATC) Simulation Package

E.1 ATC-Garnet.lisp

```
;;;; -*- Mode: LISP; Syntax: Common-Lisp; Package: USER; Base: 10 -*-
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;         The Garnet User Interface Development Environment.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;; This code was written as part of the MSc Thesis project at
;;;; Nottingham University, and has been placed in the public
;;;; domain.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;
;;;; File              : ATC-Garnet.lisp
;;;; Author            : Roberto L. Ong <itxrlo@unicorn.nott.ac.uk>
;;;; Created On        : Fri Aug 19 15:36:00 1994
;;;; Last Modified By: Roberto L. Ong <itxrlo@unicorn.nott.ac.uk>
;;;; Last Modified On: Tue Sep 13 13:13:17 1994
;;;; Update Count      : 5
;;;;
;;;; PURPOSE
;;;;
;;;; This file provides an entire lisp package that runs the Air Traffic
;;;; Control (ATC) Simulation.
;;;;
;;;; This is part of a MSc Thesis project in Information Technology.
;;;;
;;;; N.B. See the "ATC-README" file on instructions on how to the load
;;;;      and run the entire ATC code.
;;;;
;;;;
;;;; TABLE OF CONTENTS
;;;;
;;;;          I.       Load Garnet Gadgets
;;;;          II.      Global Variables
;;;;          III.     Prototype Objects
;;;;          IV.      Garnet Functions
;;;;          V.       Utlity Functions
;;;;          VI.      Main Function
;;;;
;;;; (C) Copyright 1994, Roberto L. Ong <itxrlo@unicorn.nott.ac.uk>
;;;;     University of Nottingham, all rights reserved.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;
;;;; Status
;;;; HISTORY
;;;;
;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;
```

```
;;; load this file
(format t "~%loading ATC-Garnet.lisp .........~%")

;;; load file into default package - "USER"
(in-package "USER")
(use-package '("KR" "LISP"))

;;;
;;;  I.    Load Garnet Gadgets
;;;

(load "/psyc/lang/garnet/2.2/bin/gadgets/menubar-functions.sbin")
(load "/psyc/lang/garnet/2.2/bin/gadgets/menubar.sbin")


;;;
;;;  II.    Global Variables
;;;

(defconstant pi/18 (/ pi 18))
;;; change pathname with the appropriate path in the form:
;;; <garnet-directory>/beacon.db
;;; <garnet-directory>/plane.db
(defconstant beacon-file-name "~/garnet/beacon.db")
(defconstant plane-file-name "~/garnet/plane.db")
(defparameter atc-win nil)
(defparameter atc-agg nil)
(defparameter radar-agg nil)
(defparameter *LIST-OF-BEACONS* nil)
(defparameter *LIST-OF-PLANES* nil)


;;;
;;;  III.   Prototype Objects
;;;

;;; create outer scope
(create-instance 'my-scope opal:circle
   (:box '(0 15 560 560))
   (:left (o-formula (+ 0 25)))
   (:top (o-formula (+ 15 25)))
   (:width (o-formula (- 560 50)))
   (:height (o-formula (- 560 50)))
   (:line-style opal:white-line)
   (:filling-style opal:black-fill))

;;; create prototype beacon
(create-instance 'beacon opal:aggregadget
   (:box '(0 0))
   (:center-x (o-formula (first (gvl :box))))
   (:center-y (o-formula (second (gvl :box))))
   (:width 10)
   (:radius/2 (o-formula (/ (gvl :width) 2)))
   (:parts
    `((:first-line ,opal:line
      (:x1 ,(o-formula (- (gvl :parent :center-x)
                          (gvl :parent :radius/2))))
      (:y1 ,(o-formula (gvl :parent :center-y)))
      (:x2 ,(o-formula (+ (gvl :parent :center-x)
                          (gvl :parent :radius/2))))
      (:y2 ,(o-formula (gvl :parent :center-y)))
```

```
          (:line-style ,opal:white-line))
          (:second-line ,opal:line
          (:x1 ,(o-formula (gvl :parent :center-x)))
          (:y1 ,(o-formula (- (gvl :parent :center-y)
                              (gvl :parent :radius/2))))
          (:x2 ,(o-formula (gvl :parent :center-x)))
          (:y2 ,(o-formula (+ (gvl :parent :center-y)
                              (gvl :parent :radius/2))))
          (:line-style ,opal:white-line)))))

 ;;; create prototype airplane
 (create-instance 'plane-proto opal:aggregadget
    (:width 10) (:height 10)
    (:flight-no nil)
    (:x-position 0)
    (:y-position 0)
    (:altitude 0)
    (:velocity 0)
    (:distance 0)
    (:status nil)
    (:control nil)
    (:radius/3 (o-formula (round (gvl :width) 3)))
    (:radius/2 (o-formula (round (gvl :width) 2)))
    (:std-font (opal:get-standard-font :fixed :roman :small))
    (:parts
     `((:white-circle1 ,opal:circle
        (:left ,(o-formula (gvl :parent :x-position)))
        (:top ,(o-formula (gvl :parent :y-position)))
        (:width ,(o-formula (gvl :parent :width)))
        (:height ,(o-formula (gvl :width)))
        (:line-style ,opal:white-line))
       (:white-circle2 ,opal:circle
        (:left ,(o-formula (+ (gvl :parent :x-position)
                              (gvl :parent :radius/3))))
        (:top ,(o-formula (+ (gvl :parent :y-position)
                             (gvl :parent :radius/3))))
        (:width ,(o-formula (- (gvl :parent :width)
                               (* 2 (gvl :parent :radius/3)))))
        (:height ,(o-formula (gvl :width)))
        (:line-style ,opal:white-line))
       (:flight-text ,opal:text
        (:left ,(o-formula (- (gvl :parent :x-position)
                              (gvl :parent :radius/2))))
        (:top ,(o-formula (+ (gvl :parent :y-position)
                             (gvl :parent :height))))
        (:font ,(o-formula (gvl :parent :std-font)))
        (:string ,(o-formula (format nil "~s" (gvl :parent :flight-no))))
        (:line-style ,opal:white-line)))))


;;;
;;;  IV.     Garnet Functions
;;;

;;;*************************************************************
;;;Procedures to do the work
;;;*************************************************************

(defun create-atc-menu (agg)
  "Creates a menu bar for the simulation."
  (let (atc-menu)
    ;; create a menu bar
    (setf atc-menu (create-instance NIL gg:menubar
                    (:items
                     '(("File" Nil
                       (("Load-File" read-plane-db-file)("Quit" Doquit)))
                      ("RadarScope" Nil
                       (("Show Beacon" show-beacon)
                        ("Show weather" show-weather)))
                      ("Command" Nil
                       (("Speed")("Climb")("Handoff")))
                      ("Information" Nil
                       (("Flightplan")))))))

    (opal:add-components agg atc-menu)) )

(defun create-radar-screen (agg win)
  "Creates a radar screen."
  (let (radar-screen outer-scope)
    ;; create a radar screen
    (setf radar-screen (create-instance NIL opal:rectangle
                        (:left 0) (:top 15) (:width 560) (:height 560)
                        (:filling-style opal:white-fill)))

    ;; create outer scope
    (setf outer-scope (create-instance NIL my-scope
                       (:left (o-formula (+ (gv radar-screen :left) 25)))
                       (:top (o-formula (+ (gv radar-screen :top) 25)))
                       (:width (o-formula (- (gv radar-screen :width) 50)))
                       (:height (o-formula (gvl :width)))))

    (opal:add-components agg radar-screen outer-scope)

    ;; create inner scope
    (dotimes (i 6)
      (let ((interval (* i 40)))
       (opal:add-components
        agg
        ;; create outline of radar
        (create-instance nil opal:circle
          (:line-style opal:white-line)
          (:filling-style NIL)
          (:left (formula `(+ (gv ,outer-scope :left) 5 ,interval)))
          (:top (formula `(+ (gv ,outer-scope :top) 5 ,interval)))
           (:width (formula `(- (gv ,outer-scope :width) 10 (* 2 ,interval))))
          (:height (o-formula (gvl :width)))))))

   ;; add degree values and tick marks in scope
   (dotimes (j 36)
     (let ((cos10*j (cos (* pi/18 j)))
        (sin10*j (sin (* pi/18 j)))
        (offset-x (cond ((= j 0) -3) ((< j 10) -5) (t -7)))
        (offset-y (cond ((= j 0) -1) ((< j 10) -3) (t -5))))
      (opal:add-components
       agg
       (create-instance nil opal:text
          (:font (create-instance nil opal:font (:size :small)))
          (:string (formula `(int-to-string ,j)))
          (:left (formula `(+ (opal:gv-center-x ,outer-scope)
                              ,offset-x
                              (floor (* 13/25 (gv ,outer-scope :width)
                                     ,sin10*j)))))
          (:top (formula `(+ (opal:gv-center-y ,outer-scope)
                             ,offset-y
                             (floor (* -13/25 (gv ,outer-scope :width)
                                    ,cos10*j)))))))))

    ;; create radar sweep
    (setf sweep (create-instance NIL opal:line
                  (:angle 0.0)
                  (:length (o-formula
                            (floor (- (gv outer-scope :width) 10) 2)))
                  (:x1 (o-formula (opal:gv-center-x outer-scope)))
                  (:y1 (o-formula (opal:gv-center-y outer-scope)))
                  (:x2 (o-formula
                        (floor (- (gvl :x1) (* (gvl :length)
                                               (sin (gvl :angle)))))))
                  (:y2 (o-formula
                        (floor (- (gvl :y1) (* (gvl :length)
                                               (cos (gvl :angle)))))))
                 (:line-style opal:white-line)))

    (opal:add-components agg sweep)

    ;; This interactor advances the radar sweep 1 degree every .01 sec
    (create-instance NIL inter:animator-interactor
       (:window win)
       (:start-event t)
       (:stop-event t)
       (:timer-repeat-wait 0.02)
       (:timer-handler
     #'(lambda (inter)
         (s-value sweep :angle
                (if (= (g-value sweep :angle) -359)
                    0
                  (- (g-value sweep :angle) (/ pi/18 10)))))))
   ))


(defun create-info-screen (agg win)
  "Creates a window where info about a plane can be displayed."
  (opal:add-components
   agg
   ;; creates an information screen
   (create-instance NIL opal:rectangle
      (:left 560) (:top 15) (:width 140) (:height 560))

   (create-instance NIL opal:rectangle
      (:left 560) (:top 15) (:width 140) (:height 20)
      (:filling-style opal:black-fill))

   (create-instance NIL opal:text
      (:left 560) (:top 15)
      (:justification :center)
      (:string "Plane Information")
      (:line-style opal:white-line))))


;;; not implemented at the moment -RLO (02/09/94)
(defun display-info (agg)
  "displays plane information on the screen."
  (let (info )
    ;; create plane information
    (setf info (create-instance nil opal:multifont-text
                  (:left 565)
                  (:top 35)
                  (:initial-text
                   '((" flight: ")
                     ("heading: ")
                     (" height: ")
                     (" others: ")))))

    (opal:add-components agg info)))

(defun show-weather (&rest arg-list)
  "Creates weather cells on the radar screen."
  (let (weather)
    ;; creates weather cells
    (setq weather
        (create-instance NIL opal:polyline
           (:point-list '(320 180 400 120 500 200 520 300 470 350 350 370 300
300))
           (:filling-style opal:light-gray-fill)
           (:line-style nil)
           (:draw-function :xor)))

    (opal:add-component radar-agg weather)))

(defun show-beacon (&rest arg-list)
  "Displays radar beacons across the radar."
  (if *LIST-OF-BEACONS*
      (turn-beacons-off *LIST-OF-BEACONS*)
      (with-open-file (beacon-stream beacon-file-name :direction :input)
      (do ((beacon-num (read beacon-stream nil nil))
           (count 1) (new-beacon nil))
          ((> count beacon-num) nil)
        (setq new-beacon (create-instance NIL beacon))
        (s-value new-beacon :center-x (read beacon-stream nil nil))
        (s-value new-beacon :center-y (read beacon-stream nil nil))
        (s-value new-beacon :visible t)
        (incf count)
        (opal:add-components radar-agg new-beacon)
        (setq *LIST-OF-BEACONS* (cons new-beacon *LIST-OF-BEACONS*))))))

(defun turn-beacons-off (alist-of-beacons)
  "Remove beacons from radar scope."
  (if (null alist-of-beacons)
      (setq *LIST-OF-BEACONS* nil)
    (let ((beacon (car alist-of-beacons)))
      (s-value beacon :visible nil)
      (turn-beacons-off (cdr alist-of-beacons)))))


(defun read-plane-db-file (&rest arg-list)
  "Reads plane database from a file."
  (if *LIST-OF-PLANES*
      nil
    (with-open-file (plane-stream plane-file-name :direction :input)
     (do ((plane-num (read plane-stream nil nil))
          (count 1))
         ((> count plane-num) nil)
       (let ((plane (gensym)))
         (setq plane (create-instance NIL plane-proto))
```

```
          (s-value plane :flight-no (read plane-stream nil nil))
          (s-value plane :x-position (read plane-stream nil nil))
          (s-value plane :y-position (read plane-stream nil nil))
          (s-value plane :altitude (read plane-stream nil nil))
          (s-value plane :distance
                (calculate-distance-from-airport
                   (g-value plane :x-position)
                   (g-value plane :y-position)))
          (s-value plane :velocity (read plane-stream nil nil))
          (s-value plane :control (read plane-stream nil nil))
          (s-value plane :status (read plane-stream nil nil))
          (opal:add-components radar-agg plane)
          (incf count)
          (setf *LIST-OF-PLANES* (cons plane *LIST-OF-PLANES*))
          ;; this interactor advance the plane a distance
          (create-instance nil inter:animator-interactor
             (:window atc-win)
             (:start-event t)
             (:start-where t)
             (:timer-repeat-wait 0.5)
             (:timer-handler
             #'(lambda (inter)
                  (if (<= (gv plane :distance) 20)
                      (progn ; (inter:stop-animator plane-inter)
                             ; (opal:remove-components radar-agg plane))
                         (setq *LIST-OF-PLANES*
                         (clean-up-list-of-planes *LIST-OF-PLANES*))
                         (s-value plane :visible nil))
                    (progn
                      (s-value plane :x-position
                           (calculate-new-x-position
                             (gv plane :x-position)
                             (gv plane :velocity)
                             (gv plane :distance)))
                    (s-value plane :y-position
                           (calculate-new-y-position
                             (gv plane :y-position)
                             (gv plane :velocity)
                             (gv plane :distance)))
                    (s-value plane :distance
                           (calculate-distance-from-airport
                             (gv plane :x-position)
                             (gv plane :y-position))))))))
     ))))


(defun doquit (&rest arg-list)
 "Exits the ATC simulation."
  (opal:destroy atc-win))


;;;
;;;  V.      Utility functions
;;;

(defun calculate-new-x-position (x-position velocity distance)
  "Calculate the new x-position based on where the plane is in the scope."
  (let* ((scope (create-instance nil my-scope))
         (center-x (+ (g-value scope :left)
                      (/ (g-value scope :width) 2)))
         (x-distance (abs (- center-x x-position))))
```

```
    (cond ((< x-position center-x)
           (round (+ x-position (/ (* x-distance velocity)
                                   distance))))
          ((> x-position center-x)
           (round (- x-position (/ (* x-distance velocity)
                                   distance))))
          ((= x-position center-x)
           center-x))))


(defun calculate-new-y-position (y-position velocity distance)
  "Calculate the new y-position based on where the plane is in the scope."
  (let* ((scope (create-instance nil my-scope))
         (center-y (+ (g-value scope :top) (/ (g-value scope :height) 2)))
         (y-distance (abs (- center-y y-position))))
    (cond ((< y-position center-y)
           (round (+ y-position (/ (* y-distance velocity)
                                   distance))))
          ((> y-position center-y)
           (round (- y-position (/ (* y-distance velocity)
                                   distance))))
          ((= y-position center-y)
           center-y))))

(defun calculate-distance-from-airport (x-position y-position)
  "Calculates the distance of a plane from the airport."
  (let* ((scope (create-instance nil my-scope))
         (center-x (+ (g-value scope :left)
                      (/ (g-value scope :width) 2)))
         (center-y (+ (g-value scope :top)
                      (/ (g-value scope :height) 2)))
         (distance 0))
    (setq distance
      (round (sqrt (+ (square (- center-x x-position))
                      (square (- center-y y-position))))))))

(defun clean-up-list-of-planes (alist-of-planes)
  "Clean up the *LIST-OF-PLANES* for handed planes to be removed."
  (if (null alist-of-planes)
      nil
    (let ((plane-struct (car alist-of-planes)))
      (if (eq (g-value plane-struct :control) 'handed)
          (cdr alist-of-planes)
        (cons plane-struct
          (clean-up-list-of-planes (cdr alist-of-planes)))))))

(defun int-to-string (int)
  "Converts a given integer to a string."
  (do ((x 0 (+ x 1)))
      ((= x int) (format nil "~s" (* int 10)))))

(defun square (num)
  "Calculates the square of a number."
  (* num num))


;;;
;;;  VI.     Main Function
;;;
```

```
;;;*************************************************************
;;; main procedure
;;;*************************************************************

(defun do-go (&key dont-enter-main-event-loop double-buffered-p)
  "Main routine for executing air traffic control (atc) simulation."
  (let (info-agg test)
    ;; create a viewport (top-level window)
    (setf atc-win (create-instance nil inter:interactor-window
                   (:left 300) (:top 10) (:width 700) (:height 575)
                   (:title "ATC Simulation") (:icon-title "ATC")
                   (:double-buffered-p double-buffered-p)))

    ;; create the top level aggregate
    (setf atc-agg (create-instance nil opal:aggregate
                   (:left 0) (:top 0)
                   (:width (o-formula (gv atc-win :width)))
                   (:height (o-formula (gv atc-win :height)))))

    (setf radar-agg (create-instance nil opal:aggregate))
    (setf info-agg (create-instance nil opal:aggregate))

    ;; create air traffic control (atc) simulation menu
    (create-atc-menu atc-agg)

    ;; create a radar screen
    (create-radar-screen radar-agg atc-win)

    ;; create information screen
    (create-info-screen atc-agg atc-win)

    ;; add the aggregates to the window and update
    (s-value atc-win :aggregate atc-agg)
    (opal:add-component atc-agg radar-agg info-agg)
    (opal:update atc-win)

    ;; print description
    (format t "~%ATC-Simulation:~%
              This simulation is provided as a tool for routinely tying a
              Soar model to a simulation. It is a bsimulation of a grossly
              simplified Air Traffic Control (ATC) task and therefore it
              does not contain all the features you could find in a real
              ATC. But this simulation is sufficient enough for its purpose.

              The simulation creates a radar screen with menu bars for which
              to display things like beacons, weather fragments and planes.
              It also displays a radar sweep which is constantly sweeping
              the radar scope.")

   ;; if not cmu commonlisp, then start the main event loop to look for
      events
   (unless dont-enter-main-event-loop #-cmu (inter:main-event-loop))

   ;; return atc window
   atc-win))
```

E.2 ATC-fn.lisp

```
;;;; -*- Mode: Lisp -*-
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;
;;;;
;;;; File             : ATC-fn.lisp
;;;; Author           : Roberto L.Ong <itxrlo@unicorn.nott.ac.uk>
;;;; Created On       : Fri Aug 19 15:36:00 1994
;;;; Last Modified By: Roberto L. Ong <itxrlo@unicorn.nott.ac.uk>
;;;; Last Modified On: Tue Sep 13 13:26:17 1994
;;;; Update Count     : 4
;;;;
;;;; PURPOSE
;;;;
;;;; This file provides an entire lisp package that supports ATC simulation
;;;; socket to Soar. It runs the Lisp socket code as a background process and
;;;; the Air Traffic Control (ATC) simulation in the foreground.
;;;;
;;;; This is part of a MSc Thesis project in Information Technology.
;;;;
;;;; N.B. See the "ATC-README" file on instructions on how to the load
;;;;      and run the entire ATC code.
;;;;
;;;;
;;;; TABLE OF CONTENTS
;;;;
;;;;            I.      Global Variables
;;;;            II.     Utility Functions
;;;;            III.    Soar-read-loop Functions
;;;;            IV.     ATC Functions
;;;;            V.      Main Function
;;;;
;;;; (C) Copyright 1994, Roberto L. Ong <itxrlo@unicorn.nott.ac.uk>
;;;;     University of Nottingham, all rights reserved.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;
;;;; Status
;;;; HISTORY
;;;;
;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;


;;; load this file
(format t "~%loading ATC-fn.lisp .............~%")

;;; load file into default package - "USER"
(in-package "USER")


;;;
;;;  I.     Global Variables
;;;

(defparameter *soar-read-loop-process* nil)
```

```
;;;
;;;  II.    Utility Functions
;;;

(defun plane-struct-to-list (plane)
  "Creates a list out of a plane structure."
  `(plane (flight-no ,(g-value plane :flight-no))
          (x-position ,(g-value plane :x-position))
          (y-position ,(g-value plane :y-position))
          (altitude ,(g-value plane :altitude))
          (distance ,(g-value plane :distance))
          (velocity ,(g-value plane :velocity))
          (control ,(g-value plane :control))
          (status ,(g-value plane :status))))

(defun send-to-soar (alist)
  (declare (special *socket-stream*))
  (write-message (cons 'input-link alist)))



;;;
;;;  III.    Soar-read-loop Functions
;;;

(defun launch-soar-read-loop-process ()
  "Spawn a process which is doing Soar interaction all the time
   RETURN the process."
  ;; If there was already a process running, kill it.
  (declare (special *socket-stream*))
  (when (lcl:processp *soar-read-loop-process*)
    (progn (lcl:kill-process *soar-read-loop-process*)
           (setq *soar-read-loop-process* nil)))
  (unless (connected-with-soar-p)
    (set-up-socket-as-server))
  (setf *soar-read-loop-process*
      (lcl:make-process :name "Soar-read-loop-process"
                        :priority  500
                        :stack-size 1000
                        :function 'soar-read-loop-process)))

(defun soar-read-loop-process ()
  "Creates the body of the Soar read loop process."
  (lcl:handler-bind
   ((lcl::error #'lcl:invoke-debugger))
   (loop
;     (setq *soar-read-loop-process* lucid::*current-process*)
    (when (not (connected-with-soar-p))
      (accept-client-and-make-socket-stream nil))
    (when (and (connected-with-soar-p)
               (listen *socket-stream*))
      (read-command-and-args (read-message-stream))))))

(defun connected-with-soar-p ()
  "test quick and dirty whether a connection exists"
  ;; should later be done with *features*
  (declare (special *client-id*))
  (and
   *client-id*
   (boundp '*client-id*)
   (not (eq -1 *client-id*))))
```

```
;;;
;;;  IV.     ATC Functions
;;;

(defun read-command-and-args (expression)
  "Reads a command and optional arguments, then processes it."
  ;; at the moment the commands are arriving as
  ;; (socketout-link (id <id>) (name <name>) [(args <args>*]).
  ;; note that the lisp-function has to sort out the right arguments
  (let ((command (second (assoc 'name (cdr expression))))
        (id (second (assoc 'id (cdr expression))))
        (args (mapcar
               #'(lambda (arg-el)
                   (second arg-el))
                   (remove-if-not
                     #'(lambda (el) (eq (car el) 'args))
                     (cdr expression)))))
    (if command
     (progn
       (format t "I have read the command ~d.~%" command)
       (cond ((eq command 'send-scope-info)
              (send-scope-info))
             ((eq command 'lower-plane-altitude)
              (lower-plane-altitude args *LIST-OF-PLANES*))
             ((eq command 'land-the-plane)
              (land-the-plane args *LIST-OF-PLANES*)))
       (perception-system-operate id)))))

(defun perception-system-operate (id)
  ;; send message to soar to remove latest-msg
  ;; write new message, saving it for later removal
  ;; (setq latest-msg id)
  (format t "I have done the command. ~%")
  (write-message
   `(done-external-operator ,id)))

(defun send-scope-info ()
  "Look for any planes in the scope and return their attributes."
  (if *LIST-OF-PLANES*
      (send-to-soar
       (mapcar #'plane-struct-to-list *LIST-OF-PLANES*))
    (send-to-soar '((plane (number 0))))))

(defun lower-plane-altitude (plane-attributes alist-of-planes)
  "Change the altitude of a particular plane."
  (let ((plane-struct (car alist-of-planes))
        (flight-num (car plane-attributes)))
    (if (eq (g-value plane-struct :flight-no) flight-num)
        (progn
          (s-value plane-struct :altitude 5000)
          (s-value plane-struct :control 'descend))
      (lower-plane-altitude plane-attributes (cdr alist-of-planes)))))

(defun land-the-plane (plane-attributes alist-of-planes)
  "Land a plane by handing it over to the approach controller."
  (let ((plane-struct (car alist-of-planes))
        (flight-num (car plane-attributes)))
    (if (eq (g-value plane-struct :flight-no) flight-num)
        (s-value plane-struct :control 'handed)
      (land-the-plane plane-attributes (cdr alist-of-planes)))))
```

```
;;;
;;;  V.     Main Function
;;;

(defun main-routine ()
  "This is the main routine."
  ;; initialize global variables
  (setq atc-win nil)
  (setq atc-agg nil)
  (setq radar-agg nil)
  (setq *LIST-OF-BEACONS* nil)
  (setq *LIST-OF-PLANES* nil)
  (setq *soar-read-loop-process* nil)
  (let (test)
    (setq test (launch-soar-read-loop-process))
    (format t "~s~%" test)
    (do-go)))
```

E.3 ATC-files.load

```
;;;; -*- Mode: LISP; Syntax: Common-Lisp; Package: USER; Base: 10 -*-
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;          The Garnet User Interface Development Environment.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;; This code was written as part of the MSc Thesis project at
;;;; Nottingham University, and has been placed in the public
;;;; domain.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;
;;;; File              : load-ATC-files.lisp
;;;; Author            : Roberto L. Ong <itxrlo@unicorn.nott.ac.uk>
;;;; Created On        : FRI Sep 02 12:25:17 1994
;;;; Last Modified By: Roberto L. Ong <itxrlo@unicorn.nott.ac.uk>
;;;; Last Modified On: Sat Sep 03 12:25:17 1994
;;;; Update Count      : 2
;;;;
;;;; PURPOSE
;;;;
;;;; This file provides the necessary the necessary files to be loaded
;;;; in order to run the ATC code.
;;;;
;;;; This is part of a MSc Thesis project in Information Technology.
;;;;
;;;; N.B. See the "ATC-README" file on instructions on how to the load
;;;;      and run the entire ATC code.
;;;;
;;;;
;;;; TABLE OF CONTENTS
;;;;      < see contents of this module >
;;;;
;;;; (C) Copyright 1994, Roberto L. Ong <itxrlo@unicorn.nott.ac.uk>
;;;;     University of Nottingham, all rights reserved.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;
;;;; Status
;;;; HISTORY
;;;;
;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;

;;; load this file
(format t "~%loading load-ATC-files.lisp ..........~%")

;;; load file into default package - "USER"
(in-package "USER")

(load "/psyc/teaching/UG/otherug/itxro/socket/socket/stdio.lisp")
(load "/psyc/teaching/UG/otherug/itxro/socket/socket/socket.lisp")
(load "/psyc/teaching/UG/otherug/itxro/garnet/ATC-Garnet.lisp")
(load "/psyc/teaching/UG/otherug/itxro/garnet/ATC-fn.lisp")
```

Appendix F

The ATC Soar Model

```
;;; -*- Mode: SDE -*-
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;
;;;
;;; File                 : ATC.soar
;;; Original author(s): Roberto L. Ong <itxrlo@unicorn@nott.ac.uk>
;;; Organization         : /psyc/teaching/UG/otherug/itxro/.organization
;;; Created on           : 19 Aug 1994, 13:07:35
;;; Last Modified By     : Roberto Ong <itxro@nott.ac.uk>
;;; Last Modified On     : 15 Sep 1994, 20:01:45
;;; Soar Version         : 6.2.3
;;;
;;;
;;; ABSTRACT. These Soar productions implement the Air Traffic Control (ATC)
;;;           task. The task is to find a plane and land it.
;;;
;;;
;;; Description:  This is a model of an individual doing a simple "Air Traffic
;;;               Control" task. The model interacts with the outside via UNIX
;;;               sockets. The socket stuff was provided by Joseph Mertz and
;;;               Gary Pelton, and was modified, adapted and generalized by
;;;               Roberto L. Ong and Josef Nerb (hence MONGSU [Mertz-Ong-Nerb-
;;;               Gary Socket Utility]).
;;;
;;;               It does basic ATC task such as looking at the scope and
;;;               landing planes. It also checks to see if there are no planes
;;;               in the scope. The goal is to land planes which has landing
;;;               status. If all planes (i.e., planes with status land) has
;;;               landed then the system terminates, otherwise it continues
;;;               to look at the scope.
;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;


;;; turn learning off
learn off


;;;
;;; TOP GOAL:
;;; ATC
;;;

(sp top-goal*elaborate*goal*ATC
    (goal <g> ^object nil)
    -->
    (<g> ^name ATC))
```

```
;;;
;;; TOP GOAL PROBLEM SPACE:
;;; ATC
;;;

(sp ATC*propose*space*ATC
    (goal <g> ^name ATC ^object nil)
    -->
    (<g> ^problem-space <p>)
    (<p> ^name ATC))


;;;
;;; ATC PROBLEM SPACE:
;;; DESIRED STATE
;;;

(sp ATC*elaborate*goal*desired-state
    (goal <g> ^problem-space <p> ^object nil)
    (<p> ^name ATC)
    -->
    (<g> ^desired <d>)
    (<d> ^planes-to-land none))


;;;
;;; ATC PROBLEM SPACE:
;;; INITIAL STATE
;;;

(sp ATC*propose*state*initial-state
    (goal <g> ^name ATC
              ^problem-space <p>
              ^desired <d>)
    (<p> ^name ATC)
    (<d> ^planes-to-land none)
    -->
    (<g> ^state <s>)
    (<s> ^planes-to-land yes ^first-look no ^looked no)
    (<s> ^current-planes none))


;;;
;;; ATC PROBLEM SPACE OPERATORS:
;;; look-scope -> propose
;;;

(sp ATC*propose*operator*look-scope
    (goal <g> ^problem-space <p> ^state <s> ^object nil)
    (<p> ^name ATC)
    (<s> ^planes-to-land yes ^looked no)
    ;; there should be one condition which makes sure that the rule
    ;; fires not during you are monitoring <or>
    ;; you could say that all the monitoring operators are better than
    ;; look-scope
    -->
    (<g> ^operator <o> + >)
    (<o> ^name look-scope
         ^external yes))
```

```
;;;
;;; ATC PROBLEM SPACE:
;;; look-scope*note-first-op -> implement
;;;
(sp ATC*implement*look-scope*note-first-op
    (goal <g> ^state <s> ^operator <o>)
    (<s> ^planes-to-land yes ^first-look no ^looked no)
    (<s> ^current-planes none)
    (<o> ^name look-scope)
    -->
    (<s> ^first-look no - yes +))

;;;
;;; ATC PROBLEM SPACE:
;;; look-scope -> implement
;;;
(sp ATC*implement*operator*look-scope
    (goal <g> ^state <s> ^operator <o>)
    (<s> ^planes-to-land yes ^first-look yes ^looked no)
    (<o> ^name look-scope
         ^external yes)
    -->
    (<s> ^looked no - yes +)
    (<o> ^issued <command>)
    (<s> ^socketout-link <command> + &)
    (<command> ^name send-scope-info)
    (<command> ^id (make-constant-symbol)))


;;;
;;; ATC PROBLEM SPACE:
;;; look-scope*receive*and*mark*input -> implement
;;;
(sp ATC*implement*look-scope*receive*and*mark*input
    (goal <g> ^state <s> ^operator <o>)
    (<s> ^planes-to-land yes)
    (<o> ^name look-scope)
    (<s> ^input-link <planes>)
    (<s> -^read <planes>)
    (<s> ^current-planes <old-planes>)
    -->
    (<s> ^current-planes <old-planes> - <planes> + &)
    (<s> ^read <planes> + &))

;;;
;;; ATC PROBLEM SPACE:
;;; look-scope -> terminate
;;;
(sp terminate*operator*look-scope
    (goal <g> ^problem-space <p> ^state <s> ^operator <o>)
    (<o> ^name look-scope
         ^issued <command>
         ^external yes)
    (<p> ^name ATC)
    (<s> ^current-planes <planes> { <> none })
    (<s> ^socketout-link <command>)
    (<s> ^input-link <planes> ^read <planes>)
    (<command> ^id <constant-symbol>)
    (<s> ^done-external-operator <constant-symbol>)
    -->
    (<g> ^operator <o> @))
```

```
;;;
;;; ATC PROBLEM SPACE OPERATORS:
;;; lower-altitude -> propose
;;;

;; plane has reached first approach fix
;; issue command to descend
(sp ATC*propose*operator*lower-altitude
    (goal <g> ^problem-space <p> ^state <s>)
    (<p> ^name ATC)
    (<s> ^looked yes ^current-planes <cp>)
    (<cp> ^plane <plane>)
    (<plane> ^status land ^control under ^distance {<= 200} <d>)
    -->
    (<g> ^operator <o> + >)
    (<o> ^name lower-altitude
         ^plane <plane>
         ^external yes))


;;;
;;; ATC PROBLEM SPACE:
;;; lower-altitude -> implement
;;;

(sp ATC*implement*operator*lower-altitude
    (goal <g> ^problem-space <p> ^state <s> ^operator <o>)
    (<p> ^name ATC)
    (<o> ^name lower-altitude
         ^plane <plane>
         ^external yes)
    (<plane> ^flight-no <arg>)
    -->
    (<s> ^looked yes - no +)
    (write
      (crlf) |  Flight number | <arg> | has reached First approach fix. |
      (crlf) |  Descending to 5000 meters. |)
    (<o> ^issued <command>)
    (<s> ^socketout-link <command> + &)
    (<command> ^name lower-plane-altitude)
    (<command> ^id (make-constant-symbol))
    (<command> ^args <arg>))


;;;
;;; ATC PROBLEM SPACE:
;;; lower-altitude -> terminate
;;;

(sp ATC*terminate*operator*lower-altitude
    (goal <g> ^problem-space <p> ^state <s> ^operator <o>)
    (<o> ^name lower-altitude
         ^issued <command>
         ^external yes)
    (<p> ^name ATC)
    (<s> ^socketout-link <command>)
    (<command> ^id <constant-symbol>)
    (<s> ^done-external-operator <constant-symbol>)
    -->
    (<g> ^operator <o> @ ))
```

```
;;;
;;; ATC PROBLEM SPACE OPERATORS:
;;; land-plane -> propose
;;;

(sp ATC*propose*operator*land-plane
    (goal <g> ^problem-space <p> ^state <s>)
    (<p> ^name ATC)
    (<s> ^looked yes ^current-planes <cp>)
    (<cp> ^plane <plane>)
    (<plane> ^status land ^control descend ^distance {<= 80} <d>)
    -->
    (<g> ^operator <o> + >)
    (<o> ^name land-plane
         ^plane <plane>
         ^external yes))


;;;
;;; ATC PROBLEM SPACE:
;;; land-plane -> implement
;;;

(sp ATC*implement*operator*land-plane
    (goal <g> ^problem-space <p> ^state <s> ^operator <o>)
    (<p> ^name ATC)
    (<o> ^name land-plane
         ^plane <plane>
         ^external yes)
    (<plane> ^flight-no <arg>)
    -->
    (<s> ^looked yes - no +)
    (write
      (crlf) |  Flight number | <arg> | has reached Final approach fix. |
      (crlf) |  Handing to Approach controller. |)
    (<o> ^issued <command>)
    (<s> ^socketout-link <command> + &)
    (<command> ^name land-the-plane)
    (<command> ^id (make-constant-symbol))
    (<command> ^args <arg>))


;;;
;;; ATC PROBLEM SPACE:
;;; land-plane -> terminate
;;;

(sp ATC*terminate*operator*land-plane
    (goal <g> ^problem-space <p> ^state <s> ^operator <o>)
    (<o> ^name land-plane
         ^issued <command>
         ^external yes)
    (<p> ^name ATC)
    (<s> ^socketout-link <command>)
    (<command> ^id <constant-symbol>)
    (<s> ^done-external-operator <constant-symbol>)
    -->
    (<g> ^operator <o> @))
```

```
;;;
;;; ATC PROBLEM SPACE OPERATORS:
;;; there-are-planes -> propose
;;;

;; plane has landing status but is in intermediate distance
(sp ATC*propose*operator*there-are-planes
    (goal <g> ^state <s> ^problem-space <p>)
    (<p> ^name ATC)
    (<s> ^looked yes ^planes-to-land yes ^current-planes <cp>)
    (<cp> ^plane <plane>)
    (<plane> ^status land ^control <cntl> { <> handed })
    -->
    (<g> ^operator <o> + =)
    (<o> ^name there-are-planes))


;;;
;;; ATC PROBLEM SPACE:
;;; there-are-planes -> implement
;;;

(sp ATC*implement*operator*there-are-planes
    (goal <g> ^problem-space <p> ^state <s> ^operator <o>)
    (<p> ^name ATC)
    (<o> ^name there-are-planes)
    (<s> ^planes-to-land yes)
    -->
    (<s> ^looked yes - no +)
    (write (crlf)
           |  There are planes to land. |))

;;;
;;; ATC PROBLEM SPACE:
;;; there-are-planes -> terminate
;;;

(sp ATC*terminate*operator*there-are-planes
    (goal <g> ^state <s> ^operator <o>)
    (<o> ^name there-are-planes)
    (<s> ^looked no)
    -->
    (<g> ^operator <o> @))


;;;
;;; ATC PROBLEM SPACE:
;;; no-more-planes-to-land1 -> propose
;;;

;; there are no more planes with landing status
;; there are only overflights
(sp ATC*propose*operator*no-more-planes-to-land1
    (goal <g> ^problem-space <p> ^state <s>)
    (<p> ^name ATC)
    (<s> ^planes-to-land yes ^looked yes ^current-planes <cp>)
    (<cp> ^plane <plane>)
    (<plane> ^control under ^status over)
    -->
    (<g> ^operator <o> + =)
    (<o> ^name no-more-planes-to-land))
```

```
;;;
;;; ATC PROBLEM SPACE:
;;; no-more-planes-to-land2 -> propose
;;;

;; all planes with landing status have already been
;; handed over to the approach controller
(sp ATC*propose*operator*no-more-planes-to-land2
    (goal <g> ^problem-space <p> ^state <s>)
    (<p> ^name ATC)
    (<s> ^planes-to-land yes ^looked yes ^current-planes <cp>)
    (<cp> ^plane <plane>)
    (<plane> ^control handed ^status land)
    -->
    (<g> ^operator <o> + =)
    (<o> ^name no-more-planes-to-land))


;;;
;;; ATC PROBLEM SPACE:
;;; no-more-planes-to-land3 -> propose
;;;

;; there are no more planes in the radar scope
(sp ATC*propose*operator*no-more-planes-to-land3
    (goal <g> ^state <s> ^problem-space <p>)
    (<p> ^name ATC)
    (<s> ^planes-to-land yes ^looked yes ^current-planes <cp>)
    (<cp> ^plane <plane>)
    (<plane> ^number 0)
    -->
    (<g> ^operator <o> +)
    (<o> ^name no-more-planes-to-land))


;;;
;;; ATC PROBLEM SPACE:
;;; no-more-planes-to-land -> implement
;;;

(sp ATC*implement*operator*no-more-planes-to-land
    (goal <g> ^problem-space <p> ^state <s> ^operator <o>)
    (<p> ^name ATC)
    (<o> ^name no-more-planes-to-land)
    (<s> ^planes-to-land yes)
    -->
    (<s> ^planes-to-land yes - none +)
    (write (crlf)
          |  There are no more planes to land. |))

;;;
;;; ATC PROBLEM SPACE:
;;; no-more-planes-to-land -> terminate
;;;

(sp ATC*terminate*no-more-planes-to-land
    (goal <g> ^state <s> ^operator <o>)
    (<o> ^name no-more-planes-to-land)
    (<s> ^planes-to-land none)
    -->
    (<g> ^operator <o> @))
```

```
;;;
;;; ATC: SEARCH CONTROL KNOWLEDGE
;;;
;;;

;; compare distances of the planes that has reached the first approach fix
;; and issue command to the plane that is nearest to the airport
(sp ATC*compare*operator*monitor-distance*and*lower-altitude
    (goal <g> ^problem-space <p> ^state <s>
              ^operator <o1> + { <> <o1> <o2> } +)
    (<o1> ^name lower-altitude ^plane <plane1>)
    (<o2> ^name lower-altitude ^plane <plane2>)
    (<plane1> ^distance <distance-1>)
    (<plane2> ^distance { > <distance-1> } <distance-2>)
    -->
    (<g> ^operator <o1> > <o2>))


;; compare distances of the planes that has reached the final approach fix
;; and issue command to the plane that is nearest to the airport
(sp ATC*compare*operator*monitor-distance*and*land-plane
    (goal <g> ^problem-space <p> ^state <s>
              ^operator <o1> + { <> <o1> <o2> } +)
    (<o1> ^name land-plane ^plane <plane1>)
    (<o2> ^name land-plane ^plane <plane2>)
    (<plane1> ^distance <distance-1>)
    (<plane2> ^distance { > <distance-1> } <distance-2>)
    -->
    (<g> ^operator <o1> > <o2>))

;; land-plane is always better than lower-altitude
(sp ATC*compare*operator*land-plane*better*lower-altitude
    (goal <g> ^problem-space <p> ^state <s>
              ^operator <o1> + { <> <o1> <o2> } +)
    (<o1> ^name lower-altitude)
    (<o2> ^name land-plane)
    -->
    (<g> ^operator <o2> > <o1>))


;; lower-altitude and land-plane is always better than
;; there-are-planes
(sp ATC*compare*operator*there-are-planes*always*worst
    (goal <g> ^problem-space <p> ^state <s>
              ^operator <o1> + { <> <o1> <o2> } +)
    (<o1> ^name << lower-altitude land-plane >>)
    (<o2> ^name there-are-planes)
    -->
    (<g> ^operator <o2> < <o1>))

;; watching-command-process is always better than
;; any operator proposed simultaneously
;; -> bug fix for operator tie impasse
;; -RLO (15/09/94)
(sp ATC*compare*operator*watch*always*best
    (goal <g> ^problem-space <p> ^state <s>
              ^operator <o1> + { <> <o1> <o2> } +)
    (<o1> ^name << lower-altitude land-plane >>)
    (<o2> ^name watching-command-process)
    -->
    (<g> ^operator <o2> > <o1>))
```

```
;; if there are still planes to land, don't quit
;; look-again
(sp ATC*compare*operator*there-are-planes*no-more-planes
    (goal <g> ^problem-space <p> ^state <s>
              ^operator <o1> + { <> <o1> <o2> } +)
    (<o1> ^name there-are-planes)
    (<o2> ^name no-more-planes-to-land)
    -->
    (<g> ^operator <o1> > <o2>))

;;;
;;; ATC: Watch command processor
;;;

(sp look-at-scope*impasse-watching-command-process
    (goal <g> ^impasse no-change ^attribute operator ^object <g1>)
    (goal <g1> ^operator <o>)
    (<o> ^external yes)
    -->
    (<g> ^operator <ow> <ow> >)
    (<ow> ^name watching-command-process))

(sp terminate*operator*watching-command-process
  "Terminate watching-motor-process as soon as it is selected."
  (goal <g> ^state <s> ^operator <o>)
  (<o> ^name watching-command-process)
  -->
  (<g> ^operator <o> @))

;;;
;;; ATC PROBLEM SPACE:
;;; STATE EVALUATION
;;;

;;;
;;; EVALUATION: STATE SUCCESS/GOAL TEST
;;;

;; success means there are no more planes to land
;; i.e., all planes that have to be landed already landed

(sp ATC*evaluate*state*success
    (goal <g> ^problem-space <p>
              ^state <s>
              ^desired <d>
              ^object nil)
    (<p> ^name ATC)
    (<d> ^planes-to-land none)
    (<s> ^planes-to-land none)
    -->
    (<s> ^success <d> + &))

(sp default*top-goal*halt-success*state*success
    :default
    (goal <g> ^state <s> ^desired <d> ^object nil)
    (<s> ^success <d>)
    -->
    (write (crlf) | goal | <g> | achieved |)
    (halt))

;;; eof of ATC.soar
```

Appendix G

Mertz-Ong-Nerb-Gary Socket Utility (MONGSU)

G.1 std-soar-socket.c

```
/*  -*- Mode: C -*-  */
/****************************************************************************/
/*
/*  File              : std-soar-socket.c
/*  Author            : Ralph Morelli
/*  Created On        : Sat Mar 14 14:32:14 1992
/*  Last Modified By: Roberto L. Ong <itxrlo@unicorn.nott.ac.uk>
/*  Last Modified On: Tue Sep 13 12:49:20 1994
/*  Update Count      : 19
/*
/*  PURPOSE
/*      This file provides socket support for sockets from LISP. It will
/*      interface with Allegro, Lucid or CMU Common Lisp. It was originally
/*      written for use with cT.lisp, which defines the functions given here
/*      as foreign functions. It is compiled to std-soar-socket.o and loaded
/*      into LISP. See these programs and the Makefile for details.
/*
/*      The following foreign (integer) functions are defined.
/*         sock = create_socket ()
/*         port = bind_socket (sock)
/*         client_file_descriptor = listen_socket (sock)
/*         accept_socket (sock)
/*         close_socket (sock)
/*         shutdown_socket (sock, how)
/*         connect_socket (sock, server_system_name, server_port_number)
/*
/* TO COMPILE: (Use the Makefile)
/*
/*  TABLE OF CONTENTS
/*      |>Contents of this module<|
/*
/*  Sections:
/*  (C) Copyright 1991, Carnegie Mellon University, all rights reserved.
/*  (C) Other sections Copyright 1994, Roberto L. Ong and Frank E. Ritter.
/****************************************************************************/
/*  Status           : Unknown, Use with caution!
/*  HISTORY
/*
/*   Blake Ward developed first version of this program which defined three
/*   functions: create-socket, connect-socket, destroy-socket
/*   Modification: Garret Pelton modified the functions to work with cT
/*   Modification: 3/9/92: RM modified documentation
/*   Modification: 3/11/92: RM added DEBUG and PERROR code
/*   Modification: 3/11/92: RM modified destroy_socket
/*   Modification: 3/14/92: RM redefined the foreign functions to bring them
/*   into a 1-1 relationship with C-language system calls, thereby giving
/*   more fine-grained control to LISP.
/*   Modification: 12/7/92: JM added connect_socket
/*   Modification: 19/8/94: RLO changed variable names and added global
/*                           variables to make the code more general,
/*                           understandable and clear.
/*                           Made error messages more constructive and clear.
/*
/*************************************************************************/


#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>
#include <stdio.h>
#include <errno.h>

#define DEBUG 1                      /* Set to 1 for ON, 0 for OFF */
#define DEFAULT_PROTOCOL 0           /* Set to standard protocol   */
                                     /* -RLO (19/08/94) */
#define DEFAULT_QUEUE_LENGTH 1       /* Set to number of connectrions allowed */
                                     /* -RLO (19/08/94) */

static struct sockaddr_in serverINETaddress; /* Full internet name of */
                                             /* Lisp side of socket   */
                                             /* -RLO (19/08/94)       */

/* Create a socket */
create_socket ()
{
  int sock;

  /* Create a socket on which to send. */
  sock = socket(AF_INET, SOCK_STREAM, DEFAULT_PROTOCOL);
  if (sock == -1) {
    /* modified error message -RLO (31/08/94) */
    fprintf(stderr,
          "Establish Socket: Unable to create socket in create_socket.\n");
    perror ("ERROR: CREATING STREAM SOCKET");
    return(-1);
  }
  if (DEBUG)
    fprintf (stderr, "Created Socket with id %d\n", sock);
  return (sock);
} /* end create */


/* Bind the socket to a port */
bind_socket (sock)
int sock;
{
  int namelen;

  serverINETaddress.sin_family = AF_INET;
  serverINETaddress.sin_addr.s_addr = INADDR_ANY;
  serverINETaddress.sin_port = htons(0);    /* Uses a wild card port number */
                                             /* assigned by the system */
    if (bind(sock, &serverINETaddress, sizeof(serverINETaddress)) == -1) {
      /* modified error message -RLO (31/08/94) */
      fprintf(stderr,
          "Establish Socket: Unable to bind to socket %d (in bind_socket).\n",
sock);
      perror ("BIND ERROR: Closing Socket");
      return(-1);
    }
    namelen = sizeof(serverINETaddress);
    if (getsockname (sock, &serverINETaddress, &namelen)) {
      /* modified error message -RLO (31/08/94) */
      perror("ERROR: Unable to get socket name.");
      return(-1);
    }

    return (ntohs(serverINETaddress.sin_port));
  } /* end bind */

  /* Put socket in "listen" state (one connection) */
  listen_socket (sock)
  int sock;
  {
    int rtncode;

    rtncode = listen(sock,DEFAULT_QUEUE_LENGTH);
    if (rtncode == -1) {
       fprintf(stderr, "Establish Socket: Unable to listen to socket %d.\n",
  sock);
      perror ("LISTEN ERROR: Closing Socket");
      return(-1);
    }
    if (DEBUG)
      fprintf (stderr, "Listening at socket with id %d\n", sock);

    return(rtncode);
  }  /* end listen */

  /* This function takes a socket, waits for someone to connect to it and
     then returns a unix file descriptor corresponding to the socket. */
  accept_socket (sock)
  int sock;
  {
    int client_fd;        /* fd after client has connected */
    struct sockaddr_in clientINETaddr;
    int client_addr_len;
    struct hostent *host;

    /* Wait for a connection */
    client_addr_len = sizeof(clientINETaddr);

    client_fd= accept(sock, &clientINETaddr, &client_addr_len);
    if (client_fd == -1) {
      /* commented for use with launching a process in Lisp.          */
      /* Because it keeps on printing error messages during initial   */
      /* launching of a process as there is no connected process yet */
      /* -RLO (19/08/94) */
      /* fprintf(stderr, "Accept: Unable to accept connection at socket %d\n",
 sock); */
      /* perror ("ACCEPT ERROR: Closing Socket"); */
      return(-1);
    }
    host = gethostbyaddr((char*) &clientINETaddr.sin_addr.s_addr,4,AF_INET);

    if (DEBUG)
      fprintf(stderr, "Accepted connection from %s\n",host->h_name);

    return(client_fd);
  } /* end accept */


  /* connect_socket: This function takes the server name and port number
   * of a socket from a server process, and a socket server_fd and connects
   * to it.  The process returns the result of the connect system call.
   */

  int connect_socket(server_fd, server_name, port_no)
  int server_fd, port_no;
  char *server_name;
  {
    struct sockaddr_in serverINETaddress;
    struct hostent *host;
    int rtncode;

    host = gethostbyname(server_name);

    if (host == 0) {
      fprintf(stderr, "Connect: Unknown host %s.\n", host);
      perror("CONNECT ERROR");
      return(-1);
    }
    bzero((char *)&serverINETaddress, sizeof(serverINETaddress));
    serverINETaddress.sin_family = host->h_addrtype;
    bcopy(host->h_addr, (char *)&serverINETaddress.sin_addr, host->h_length);
    serverINETaddress.sin_port = htons(port_no);

    rtncode = connect(server_fd, &serverINETaddress, sizeof(serverINETaddress));

    if (rtncode == -1) {
      fprintf(stderr, "Test:  Unable to connect (errno=%d).\n",errno);
      perror ("ERROR MESSAGE");
      return(-1);
    }

    if (DEBUG)
      fprintf("Connected to server with fd %d\n",server_fd);
    return(rtncode);
  }



  /* Close the socket */
  close_socket (sock)
  int sock;
  {
    int rtncode;

    rtncode = close(sock);
    if (rtncode == -1) {
      fprintf (stderr, "close: Unable to close socket %d.\n", sock);
      perror ("CLOSE ERROR");
      return(-1);
    }
  if (DEBUG)
    fprintf (stderr,"Closed socket with id %d\n", sock);

  return(rtncode);
}  /* end close */


/*    Shutdown causes a socket's connection to be shutdown. If how is 0, then
      further receives will be disallowed.  If how is 1, then further sends
      will be disallowed.  If how is 2, then further sends and receives will
      be disallowed.          */

shutdown_socket (sock, how)
int sock, how;
{
  int rtncode;

  rtncode = shutdown(sock,how);
  if (rtncode == -1) {
    fprintf (stderr, "close: Unable to shutdown socket %d.\n", sock);
    perror ("SHUTDOWN ERROR");
    return(-1);
  }
  return (rtncode);
}
```

G.2 soar-socket.c

```
/*  -*- Mode: C -*-  */
/************************************************************************
/*
/* File             : soar-socket.c
/* Author           : Joe Mertz
/* Created On       : December 1992
/* Last Modified By: Roberto L. Ong <itxrlo@unicorn.nott.ac.uk>
/* Last Modified On: Thu Sep 27 12:57:20 1994
/* Update Count     : 14
/* Associated modules : std-soar-socket.c
/*                    : ../src/*  (i.e., the rest of Soar6)
/*
/* Description: This file implements a simple socket I/O facility for
/*              Soar6.  The user has to be aware of the following new
/*              Soar interface interface commands
/*
/*              init-socket-io
/*              init-socket-server
/*              close-socket-io
/*              shutdown-socket-io
/*              socket-output-link
/*
/*    See "help" for each command in Soar6 for more information.
/*
/* Changes necessary in HOOKS.C:
/*    stdsocket_init() must be called from system_startup_hook()
/*    stdsocket_flush() must be called from before_init_soar_hook()
/*
/* Also, do not forget to add soar-socket.c and std-soar-socket.c to the
/* makefile.
/*
/* CAVEATS:
/* Some sizes to keep in mind:
/*    10 chars   Max length server name -RLO (19/08/94)
/*  1024 chars   Max length of single output wme list
/*  2048 chars   Max length of single input wme list
/*   128 chars   Max length of any input attribute or value constant
/*   512 tokens  Max number per single input list.
/*                   Token = '(' | ')' | attribute constant | value constant
/*
/* That should be all of them.  I think that these are reasonably
/* large, and making them dynamic was more work than I wanted to do
/* at this point.  But if you start blowing up, you might check
/* these hardcoded barriers first (there is no checking in the code).
/*
/* Error recovery from incoherent input lists is not very strong.
/*
/*  "When working with cT, and other systems where Soar is directing the
/*  action, Soar as a server works well. Some of these systems can not really
/*  work well as a server, but can interface to a server easily.
/*
/*  When working with multiple Soars or Soar and other agent-like systems and
/*  and all these agents talking to some common interface, Soar as a client
/*  works better."
/*  -GAP (September, 1994)
/*
/* Revision history:
/* ??Dec92 - started hacking
/* 01Feb93 - added this header
/* 16Jun93 - added ability to have (foo ()) which will translate into
/*           an attribute foo with an id as its value.
/* 03Feb93 - seriously bumped up the limits listed above to accomodate the
/*           posttest test.  Used respectively: 1024 30000 128 5000
/* 19Aug94 - added header files for Soar global variables
/* (RLO)     modified init_socket_io to act as a server and allow parameters
/*                    to be typed in the Soar command line
/*           modified variables to make it compatible to Soar6.2.3
/*           modified init-socket-io for Soar to act as client
/*           added init-socket-server for Soar to act as server
/*           modified functions to work in Soar 6.2.3
/*           added shutdown-socket-io
/*
/************************************************************************/

#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <strings.h>
#include <fcntl.h>
#include <ctype.h>
#include <errno.h>
/* provide full path for the following soar files -RLO */
#include "/psyc/teaching/UG/otherug/itxro/soar6.2.3/src/soar.h"
#include "/psyc/teaching/UG/otherug/itxro/soar6.2.3/src/global_vars.c"

/* The following external functions are defined in socket.c. */
extern int create_socket ();
extern int connect_socket (int, char*, int);
extern int bind_socket (int);
extern int listen_socket (int);
extern int accept_socket (int);
extern int close_socket (int);
extern int shutdown_socket (int, int);

/* Definition of constants */
#define MAX_SERVER_NAME_LENGTH 10 /* max length of server name */
                                  /* -RLO (19/08/94) */

int client_id = -1;    /* file descriptor for client process */
                       /* if client_id == -1, then the socket is not up, */
                       /* initialize for use with stdsocket_flush during */
                       /* init-soar        */
                       /* -RLO (31/08/94)  */

int socket_id;         /* file descriptor for server process    */
                       /* This is used if Soar acts as a server */
                       /* -RLO (31/08/94) */


/* OUTPUT STUFF */
char wme_list_buf[102]; /* where wme lists are built for output */


/* INPUT STUFF */
char in_buf[30000];    /* where whole input lists are read and staged */
int in_nest;           /* the level of nesting currently being input */
int in_count;       /* the character count in the current list being input */
char in_string[128]; /* where each input token is built */
```

```
int in_string_idx;   /* end of input token being built */
int in_integer, in_float, in_sym_constant;  /* use to decide token type */

typedef enum paren_type_enum {LEFTPAREN, RIGHTPAREN} paren_type;

typedef union in_token_union {
  Symbol *in_symbol;
  paren_type paren;
} in_token;

in_token in_tokens[5000];

typedef struct in_list_struct {
  Symbol *link_symbol;
  wme *link_wme;
  struct in_list_struct *next;
} in_list;

in_list *in_links;

int in_token_last;
int in_token_next;

/* Routines defined in this file */
void stdsocket_init (void);
void stdsocket_flush (void);
void socketout (int, io_wme*);
void parse_output (Symbol*, io_wme*);
bool init_socket_io (void);
bool init_socket_server (void);
bool close_socket_io (void);
bool shutdown_socket_io (void);
void socketin (int);
void parse_input (void);
void build_wmes (Symbol*);
void buildtoken (char);
void endtoken ();
void paren (paren_type);
in_token get_input_token ();
in_token get_input_token_paren ();
in_token get_input_token_symbol ();
bool is_in_token_paren (in_token);
bool is_in_token_symbol (in_token);
bool socket_output_link (void);


/***** stdsocket_flush: read any input on socket that may be pending *******/
void stdsocket_flush (void) {
  if (client_id == -1) return;  /* flush only if the socket is open */

  while(TRUE) { /* read until error or when input blocked */
    char next_char;
    if (read(client_id, &next_char, 1) == -1)
       if (errno == EWOULDBLOCK)  /* poll shows nothing to read on socket */
          return;
       else {
          perror ("ERROR message ");
          return;
       }
  }
}
```

```
/*********** socketout:  output routine called by soar **********/
void socketout (int mode, io_wme *outputs) {
  Symbol *socketout_link_id;

  /* currently only care about ADDED_OUTPUT_COMMAND */
  switch (mode) {
  case REMOVED_OUTPUT_COMMAND:
    printf ("This is remove_output_command\n");
    return;
  case MODIFIED_OUTPUT_COMMAND:
    printf ("This is modified_output_command\n");
    return;
  case ADDED_OUTPUT_COMMAND: break;
  }


/* The statement below converts something like: */
/* (X1 ^foo Z1)                                 */
/* (Z1 ^bar noo)                                */
/* (S1 ^socketout-link X1)                      */
/* and turns it into                            */
/* (foo (bar noo))                              */
/* If the above output is desired               */
/* (a) uncomment the statement below            */
/* (b) change arguments of parse_output to (socketout_link_id, outputs) */
/* -RLO (19-Aug-94)                             */
/* socketout_link_id = get_output_value (outputs, current_agent(top_state),
NIL);  */

  /* take something like:             */
  /* (X1 ^foo Z1)                     */
  /* (Z1 ^bar noo)                    */
  /* (S1 ^socketout-link X1)          */
  /* and turn it into                 */
  /* (socketout-link (foo (bar noo))) */

  /* reset the buffer "wme_list_buf" for building the nested wme list */
  wme_list_buf[0] = '\0';
  parse_output (current_agent(top_state), outputs);
  strcat (wme_list_buf, " ");

  /* put extra space on end for dumb lisp reader*/
  if (write (client_id, wme_list_buf, strlen(wme_list_buf) ) == -1 ) {
    printf("soar-socket.c: Unable to write to fd %d.\n", client_id);
  }
  printf("\nwrite successful socket %d\n", client_id);
}


/********** parse_output:  take output wme's and turn to nested list *****/
void parse_output (Symbol *id, io_wme *outputs)
{
  io_wme *next_wme;

  for (next_wme = outputs ; next_wme != NIL ; next_wme = next_wme->next) {

    if (next_wme->id == id) {

      /* start with attribute */
      strcat (wme_list_buf, "(");
      strcat (wme_list_buf, next_wme->attr->sc.name);
```

```
        /* add value and finally right paren */
        switch (next_wme->value->common.symbol_type) {
        case VARIABLE_SYMBOL_TYPE:
             print ("whoops! a variable as a value.");
             break;
        case IDENTIFIER_SYMBOL_TYPE:
             strcat (wme_list_buf, " ");
             parse_output (next_wme->value, outputs);
             break;
        case SYM_CONSTANT_SYMBOL_TYPE:
             strcat (wme_list_buf, " ");
             strcat (wme_list_buf, next_wme->value->sc.name);
             break;
        case INT_CONSTANT_SYMBOL_TYPE:
             sprintf (wme_list_buf+strlen(wme_list_buf), " %u",
             next_wme->value->ic.value);
             break;
        case FLOAT_CONSTANT_SYMBOL_TYPE:
             sprintf (wme_list_buf+strlen(wme_list_buf), " %f",
             next_wme->value->fc.value);
             break;
        }
        strcat(wme_list_buf, ")");
      }
    }
}

/********** init_socket_io:  interface command to initialize socket ***/
/* Syntax: init-socket-io <server-name> <port-no> -RLO (31/08/94) */
/* This allows Soar to act as a client */
bool init_socket_io (void)
{
  int result;
  int server_port_no;
  char server_name[MAX_SERVER_NAME_LENGTH];

  get_lexeme();  /* consume "init_socket_io", advance to arguments */

  /* create a raw socket */
  if ((client_id = create_socket())  == -1) return FALSE;

  /* Gets the server name argument as a symbol constant -RLO (19/08/94) */
  if (current_agent(lexeme).type != SYM_CONSTANT_LEXEME) {
    printf("Server name should be a symbol.\n");
    return FALSE;
  } else {
    /* copies the server name as a string -RLO (19/08/94) */
    strcpy(server_name, current_agent(lexeme).string);
    get_lexeme(); /* consume server name */
  }

  printf("Connecting to server with name %s\n", server_name);

  /* Gets the server port number argument */
  if (current_agent(lexeme).type != INT_CONSTANT_LEXEME) {
    printf("Server port number should be an integer.\n");
    return FALSE;
  } else {
    server_port_no = current_agent(lexeme).int_val;
    get_lexeme(); /* consume port number */
  }
```

```
    printf("Connecting to server with port #%d\n", server_port_no);
    fflush(stdout);

    /* connect to named server */
    if ((connect_socket(client_id, server_name, server_port_no)) == -1)
      return FALSE;

    printf("connected to socket %d", client_id);
    /* use non-blocking i/o to stream */
    if ((fcntl (client_id, F_SETFL, FNDELAY)) == -1) return FALSE;

    /* do the rest of initialization: */
    in_nest = 0;
    in_buf[0] = '\0';
    in_count = 0;
    in_links = NIL;
    return TRUE;
}


/********** init_socket_server:  interface command to initialize socket ***/
/* Syntax: init-socket-server  -RLO (31/08/94) */
/* Use this function to make Soar act as a server. */
bool init_socket_server (void) {
  int socket_port_no;

  get_lexeme();  /* consume command */

  if (client_id != -1) {  /* if the socket is already open, close first */
    print ("First close previous socket connection.\n");
    return FALSE;
  }

  if ((socket_id = create_socket())  == -1 ) return FALSE;
  if ((socket_port_no = bind_socket(socket_id)) == -1) return FALSE;

  print ("Please initialize I/O client with port #%d\n", socket_port_no);
  fflush(stdout);

  /* Put the socket in "listen" state (allow only one connection) */
  if (listen_socket(socket_id) == -1) return FALSE;
  print ("Waiting for connection...\n");
  fflush(stdout);

  if ((client_id = accept_socket (socket_id)) == -1 ) return FALSE;

  /* use non-blocking i/o to stream */
  if ((fcntl (client_id, F_SETFL, FNDELAY)) == -1) return FALSE;

  print ("Connection complete.\n");

  /* do the rest of initialization: */
  in_nest = 0;
  in_buf[0] = '\0';
  in_count = 0;
  in_links = NIL;
  return TRUE;
}
```

```
/********** close_socket_io:  interface command to close socket *****/
/* Syntax: close-socket-io */
bool close_socket_io (void)
{
  get_lexeme();  /* consume command */

  /* clean up */
  while (in_links) {
    in_list *x;
    remove_input_wme (in_links->link_wme);
    release_io_symbol (in_links->link_symbol);
    x = in_links;
    in_links = in_links->next;
    free_memory (x, MISCELLANEOUS_MEM_USAGE);
  }

  if (close_socket(client_id) == -1) return FALSE;
}

/********** shutdown_socket_io:  interface command to close socket *****/
/* Syntax: shutdown-socket-io */
/* - RLO (15/09/94) */
bool shutdown_socket_io (void)
{
  get_lexeme();  /* consume command */

  /* clean up */
  while (in_links) {
    in_list *x;
    remove_input_wme (in_links->link_wme);
    release_io_symbol (in_links->link_symbol);
    x = in_links;
    in_links = in_links->next;
    free_memory (x, MISCELLANEOUS_MEM_USAGE);
  }

  if (close_socket(socket_id) == -1) return FALSE;
  client_id = -1;
}

/********** socketin:  input routine called by soar ******/
void socketin (int mode)
{
  int rd_result;
  char next_char;

  if (client_id == -1) return;   /* if the socket is not initialized */

  switch (mode) {
  case TOP_STATE_JUST_CREATED:     /* Initialize and internalize objects */
       return;
  case TOP_STATE_JUST_REMOVED:     /* Clean up */
       while (in_links) {
         in_list *x;
       /* remove_input_wme is not done because it has already been removed */
         release_io_symbol (in_links->link_symbol);
         x = in_links;
         in_links = in_links->next;
         free_memory (x, MISCELLANEOUS_MEM_USAGE);
       }
       return;
    case NORMAL_INPUT_CYCLE:         /* Standard stuff */
         break;
    }

    while((rd_result = read(client_id, &next_char, 1)) > 0)
    {
      in_buf[in_count++] = next_char;
      switch (next_char) {
      case '(':
        in_nest++;
        break;
      case ')':
        if (in_nest-- == 1) { /* transitioning 1 to 0 means completed list */
      parse_input();           /* so parse it into wme's */
      in_buf[0] = '\0';
      in_count = 0;
        }
        break;
      }
    }

    /* return on error or when input blocked */
    if (rd_result == 0)
      return;
    else {
      if (errno == EWOULDBLOCK) { /* poll shows nothing to read on socket */
        return;
      } else {
        perror ("ERROR message ");
        return;
      }
    }
}

/********** parse_input:  take input and turn into wme's ***********/
void parse_input () {
  in_token link_token;
  in_list **link_p;
  int idx;

  /* first syntactically turn the string in_buf into a list in_token
   * of either symbol or parenthesis tokens
   */
  in_string_idx = in_integer = in_float = in_sym_constant = in_token_last =
0;
  for (idx = 0 ; in_buf[idx] != '\0' ; idx++) {
    switch (in_buf[idx]) {
    case '(':   endtoken(); paren(LEFTPAREN);  break;
    case ')':   endtoken(); paren(RIGHTPAREN); break;
    case ' ':   endtoken();                    break;
    default:   buildtoken(in_buf[idx]);        break;
    }
  }

  /* the link name should be the second token: "( eyes ..." */
  link_token = in_tokens[1];
  if (is_in_token_paren(link_token)) {
    printf("Socket input error, first item on list is not an atom\n");
    printf("Input string: %s\n", in_buf);
    return;
  }
  /* for now, throw out all old input link wmes matching the link symbol */
  link_p = &in_links;

  while (*link_p) {
    if ((*link_p)->link_symbol == link_token.in_symbol) {
      in_list *x;
      remove_input_wme ((*link_p)->link_wme);
      release_io_symbol ((*link_p)->link_symbol);
      x = *link_p;
      *link_p = (*link_p)->next;
      free_memory (x, MISCELLANEOUS_MEM_USAGE);
    } else
      link_p = &((*link_p)->next);
  }

  in_token_next = 0;  /* reset the index to the first token */
  get_input_token_paren();  /* always call build_wmes past opening paren */
  build_wmes(current_agent(top_state));
}


/********** build_wmes: recursively transverse the in_token's and make wmes
**/
void build_wmes(Symbol *id) {
  in_token attribute_token, value_token;
  wme *new_wme;
  Symbol *sub_value;

  sub_value = NIL;
  attribute_token = get_input_token_symbol();

  /* may be the case of (), which means just an unattached id */
  if (attribute_token.paren == RIGHTPAREN) return;

  for (value_token = get_input_token();
       value_token.paren != RIGHTPAREN;
       value_token = get_input_token()) {
    new_wme = NIL;
    if (value_token.paren == LEFTPAREN) {   /* we have substructure */
      if (!sub_value) {
    sub_value=get_new_io_identifier(attribute_token.in_symbol->sc.name[0]);
        new_wme = add_input_wme (id,
                                 attribute_token.in_symbol,
                                 sub_value);
      }
      build_wmes (sub_value);
    }
    else {  /* no substructure */
      new_wme = add_input_wme (id,
                               attribute_token.in_symbol,
                               value_token.in_symbol);
      release_io_symbol (value_token.in_symbol);
    }
    if (new_wme) {
      if (id == top_state) { /* save a record of these wme's */
        in_list *new;
        new = (in_list *) allocate_memory
                          (sizeof (in_list), MISCELLANEOUS_MEM_USAGE);
     /* need to get_io_sym_constant to keep the reference counts correct */
        new->link_symbol =
                  get_io_sym_constant(attribute_token.in_symbol->sc.name);
```

```
          new->link_wme = new_wme;
          new->next = in_links;
          in_links = new;
        }
      }
    }
    release_io_symbol (attribute_token.in_symbol);
    if (sub_value) release_io_symbol (sub_value);
  }


  /**********buildtoken(): add letters to a token *****************/
  void buildtoken (char nextchar) {
    in_string[in_string_idx++] = nextchar;
    in_string[in_string_idx] = '\0' ;
    if (isdigit(nextchar)) in_integer++;
    else if ((nextchar == '.') && (in_float == 0)) in_float++;
    else in_sym_constant++;
  }

  /********** endtoken:  turn string into token and add to list ******/
  void endtoken () {
    Symbol *tok_sym;

    /* the io constants gotten here are released as they are used after */
    /* becoming part of a wme.                                           */

    if ((in_sym_constant > 0) || ((in_float == 1) && (in_integer == 0))) {
      if (in_string[0] == '<') {  /* simple test that it is an identifier  */
                                  /*  ... it would be nice if I kept track */
                                  /* of these, then i could have <x>       */
                                  /* multiple places in the tree and make  */
                                  /* them the same soar id... but I don't  */
                                  /* need this immediately so i will take  */
                                  /* the short route                       */

        in_tokens[in_token_last++].in_symbol =
                                   get_new_io_identifier(in_string[1]);
      } else
        in_tokens[in_token_last++].in_symbol = get_io_sym_constant(in_string);
    }
    else if (in_float == 1) { /* and in_integer must be > 0 */
      in_tokens[in_token_last++].in_symbol =
                                   et_io_float_constant(atof(in_string));
    }
    else if (in_integer > 0) {
      in_tokens[in_token_last++].in_symbol =
                                   get_io_int_constant(atoi(in_string));
    }

    /* reset everything */
    in_string_idx = in_integer = in_float = in_sym_constant = 0;
  }


  /********** leftparen: add to input list that found left paren ******/
  void paren(paren_type paren) {
    in_tokens[in_token_last++].paren = paren;
  }
```

```
/********** get_input_token: return the next token in the input list ****/
in_token get_input_token () {
  return in_tokens[in_token_next++];
}

/********* get_input_token_paren: return next token & check if paren ***/
in_token get_input_token_paren () {
  in_token tok;
  tok = get_input_token();
  if (is_in_token_paren(tok))
    return tok;
  else {
    printf("Socket input error, expected a parenthesis\n");
    printf("Input string: %s\n", in_buf);
    return;
  }
}

/********* get_input_token_symbol: return next token & check if symbol ***/
in_token get_input_token_symbol () {
  in_token tok;
  tok = get_input_token();
  if (is_in_token_paren(tok)) {
    printf("Socket input error, first item on list is not an atom\n");
    printf("Input string: %s\n", in_buf);
    return;
  }
  else return tok;
}

/********** is_in_token_paren: is the token a paren? ***/
bool is_in_token_paren(in_token tok) {
  if ((tok.paren == LEFTPAREN) || (tok.paren == RIGHTPAREN))
    return TRUE;
  else
    return FALSE;
}

/********** is_in_token_paren: is the token a symbol (not paren)? ***/
bool is_in_token_symbol (in_token tok) {
  return (!is_in_token_paren(tok));
}

/********** socket_output_link:  interface command to set output link ***/
/* Syntax: socket-output-link <output-link-name> */
bool socket_output_link (void) {

  get_lexeme();  /* consume command */

  while (current_agent(lexeme).type != R_PAREN_LEXEME) {
    if (current_agent(lexeme).type == SYM_CONSTANT_LEXEME) {
      add_output_function (current_agent(lexeme).string, socketout);
      get_lexeme();
    } else {
      printf("Expected an output link name.\n");
      print_location_of_most_recent_lexeme();
      return FALSE;
    }
  }
  return TRUE;
}
```

```
/******************** HELP STRINGS ********************************/
/* modified to incorporate changes on the commands -RLO (31/08/94) */

char *help_on_init_socket_io[] = {
"Command: init-socket-io",
"",
"Syntax: init-socket-io server_name server_port_no",
"",
"This command creates a socket port and connects to a named server and",
"port number.",
"",
"Once a socket is initialized, then when any modification is made",
"to a socket-output-link augmentation on the top-state, the",
"augmentation and its substructure is sent as a nested list",
"out the socket port.",
"",
"A default socket-output-link is created called socketout-link.",
"Use socket-output-link to add other links.",
0 };


/* added this help for init-socket-servger */
/* -RLO (15/09/94) */
char *help_on_init_socket_server[] = {
"Command: init-socket-server",
"",
"Syntax: init-socket-server",
"",
"This command creates a socket port and displays its port number.",
"The command then waits for another process to connect to the socket",
"port, using this number.",
"",
"Once a socket is initialized, then when any modification is made",
"to a socket-output-link augmentation on the top-state, the",
"augmentation and its substructure is sent as a nested list",
"out the socket port.",
"",
"A default socket-output-link is created called socketout-link.",
"Use socket-output-link to add other links.",
0 };


char *help_on_close_socket_io[] = {
"Command:  close-socket-io",
"",
"Syntax: close-socket-io",
"",
"This command closes a previously initialized and opened client socket.",
0 };

/* added this help for shutdown-socket-io */
/* -RLO (15/09/94) */
char *help_on_shutdown_socket_io[] = {
"Command:  shutdown-socket-io",
"",
"Syntax: shutdown-socket-io",
"",
"This command closes a previously initialized and opened server socket.",
0 };
```

```
char *help_on_socket_output_link[] = {
"Command: socket-output-link",
"",
"Syntax: socket-output-link output-link-name",
"",
"The socket-output-link command adds socket output-links.  Once a",
"socket-output-link is named, when any modification to an augmentation",
"by that name on the top-state is made, the augmentation and its",
"substructure is sent as a nested list out the socket port.",
0 };


/********** stdsocket_init:  register socket routines with soar ********/
void stdsocket_init (void) {
  add_command ("init-socket-io", init_socket_io);
  add_help ("init-socket-io", help_on_init_socket_io);

  /* added this function for creating a server socket */
  /* -RLO (15/09/94) */
  add_command ("init-socket-server", init_socket_server);
  add_help ("init-socket-server", help_on_init_socket_server);

  add_command ("close-socket-io", close_socket_io);
  add_help ("close-socket-io", help_on_close_socket_io);

  /* added this function for shutting down a server socket */
  /* -RLO (15/09/94) */
  add_command ("shutdown-socket-io", shutdown_socket_io);
  add_help ("shutdown-socket-io", help_on_shutdown_socket_io);

  add_command ("socket-output-link", socket_output_link);
  add_help ("socket-output-link", help_on_socket_output_link);

  add_input_function (socketin);
  add_output_function ("socketout-link", socketout);
}
```

G.3 socket.lisp

```
;;;; -*- Mode: Lisp -*-
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;
;;;;
;;;; File             : socket.lisp
;;;; Author           : Blake Ward, Garret Pelton, Ralph Morelli, Joe Mertz
;;;; Created On       : Mon Mar 16 08:58:50 1992
;;;; Last Modified By: Roberto L. Ong <itxrlo@unicorn.nott.ac.uk>
;;;; Last Modified On: Thu Sep 15 13:10:33 1994
;;;; Update Count     : 58
;;;;
;;;; PURPOSE
;;;;
;;;; This file provides low-level socket support in Lucid, CMU or
;;;; Allegro Common Lisp. It is meant to be used in conjunction with the
;;;; object file "std-soar-socket.o".  The ".o" is a compilation of the
;;;; functions in "std-soar-socket.c" which are imported into LISP and
;;;; declared as foreign functions.
;;;;
;;;; There are two modes for which the socket support has been written.
;;;; The first mode is for use when the lisp process is a server and the
;;;; make-socket-stream is used to accept a client process (e.g., Soar).
;;;; The second mode is for use when the lisp process is a client and the
;;;; connect-socket-stream call is used to connect to a soar process.
;;;;
;;;; FOREIGN FUNCTIONS DECLARED -- These correspond to unix system calls
;;;;  (create_socket)          -- creates a UNIX socket
;;;;  (bind_socket sock port) -- binds soc to port in Unix address space
;;;;  (listen_socket sock)    -- wait at soc for external process to connect
;;;;  (accept_socket sock)     -- returns file descriptor of external socket
;;;;  (connect_socket sock name port) -- connects to server port on name
;;;;  (close_socket sock)      -- closes the socket
;;;;  (shutdown_socket sock how) -- discontinues communication at socket
;;;;
;;;; Based on the above foreign functions, the following functions provide
;;;; socket-support for SoarIO.
;;;;
;;;;     make-socket-stream
;;;;             creates a socket using a wildcard 'port', waits for someone
;;;;             to connect to it and then creates an input/output character
;;;;             stream that uses the socket and the socket. Uses global
;;;;             variables *socket-id*, *socket-stream*, *socket-port* and
;;;;             *client-id*.
;;;;     connect-socket-stream server_name server_port
;;;;             creates a socket and connects to server_port on system
;;;;             server_name
;;;;     shutdown-socket-stream
;;;;             Closes *socket-stream* and *socket-id* and resets the
;;;;             corresponding global variables.
;;;;
;;;; TABLE OF CONTENTS
;;;;  |>Contents of this module<|
;;;;
;;;; Sections:
;;;; (C) Copyright 1991, Carnegie Mellon University, all rights reserved.
;;;; (C) Other sections Copyright 1994, Roberto L. Ong and Frank E. Ritter
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;
;;;; Status           : Debbugged only the Allegro code. (RM) Ditto. (JM)
;;;; HISTORY
```

```
;;;;   1. First version of this interface was created by BW for ET-Soar.
;;;;   2. Second version was created by GAP for draw-Soar on cT.
;;;;   3. Current version by RM for Soar/ITS (also for cT-Soar).
;;;;   4. Added connect-socket-stream and *soar6-mode*.
;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;

(format t "~%loading socket.lisp ..............~%")

;;; load file into default package - "USER"
;;; -RLO (10/09/94)
(in-package "USER")

;;; The following variables are for keeping version of the code
;;; and not used for any other purpose -RLO
(defvar *soar-version* '6.2.3)
(defvar *socket-lisp-version* 1.0)

;;; define global variables
;;; replace the path for *socket-o-file* with the appropriate path in the
;;; form:
;;; "<soar-directory>/user/obj/<machine-directory>/std-soar-socket.o>"
;;; -RLO (09/09/94)
(defparameter *socket-o-file* "~/soar6.2.3/user/obj/sun4/std-soar-socket.o")
(defparameter *soar6-mode* t)
(defparameter *debug-on* nil)
(defparameter *sockets-on* t)
(defparameter *socket-connected* nil)
(defparameter *socket-id* nil)
(defparameter *socket-port* nil)
(defparameter *socket-stream* nil)
(defparameter *client-id* nil)

(if *soar6-mode* (setq *print-case* :downcase))

;;; gc before foreign function load -fer 14/9/90
#+:ALLEGRO
   (excl:gc t)

;;; commented because of garbage collection problems -RLO (29/08/94)
;;#-:ALLEGRO
;;   (gc t)

;;; Definitions of foreign functions only works for LUCID 4.1.
;;; If other versions of Lucid Lisp is going to be used, reference to
;;; the manual is strongly suggested.
;;; -RLO (09/09/94)
#+(and :LUCID :LCL4.1)
(if *sockets-on*
    (progn
      (def-foreign-function (create_socket (:name "_create_socket")))
      (def-foreign-function (bind_socket (:name "_bind_socket")) sock)
      (def-foreign-function (listen_socket (:name "_listen_socket")) sock)
      (def-foreign-function (accept_socket (:name "_accept_socket")) sock)
      (def-foreign-function (close_socket (:name "_close_socket")) sock)
      (def-foreign-function (shutdown_socket (:name "_shutdown_socket")) sock
how)
      (def-foreign-function (connect_socket (:name "_connect_socket"))
server-fd server-name port-no)
      (load-foreign-files *socket-o-file*)))
```

```
#-(and :LUCID :LCL4.1)
(if *sockets-on*
    (progn
      (define-foreign-function :c 'create_socket :integer)
      (define-foreign-function :c 'bind_socket   :integer)
      (define-foreign-function :c 'listen_socket :integer)
      (define-foreign-function :c 'accept_socket :integer)
      (define-foreign-function :c 'close_socket  :integer)
      (define-foreign-function :c 'shutdown_socket :integer)
      (define-foreigh-function :c 'connect_socket :integer)
      (load-foreign-files
       '(*socket-o-file*))
      ))

#+:CMU
(if *sockets-on*
    (progn
      (extensions:load-foreign "std-soar-socket.o")

      (extensions:def-c-routine "create_socket" (int) ())
      (extensions:def-c-routine "bind_socket" (int) (soc int))
      (extensions:def-c-routine "listen_socket" (int) (soc int))
      (extensions:def-c-routine "accept_socket" (int) (soc int))
      (extensions:def-c-routine "close_socket" (int) (soc int))
      (extensions:def-c-routine "shutdown_socket" (int) (soc int how int))
      (extensions:def-c-routine "connect_socket" (int) (soc int name string?
 port-no int))
      ))

#+:DEC3100
(if *sockets-on*
    (load *socket-o-file*))

#+:ALLEGRO
(if *sockets-on*
    (progn
      (ff:defforeign 'create_socket :arguments '()
                                    :return-type :integer)
      (ff:defforeign 'bind_socket :arguments '(fixnum)
                                  :return-type :integer)
      (ff:defforeign 'listen_socket :arguments '(fixnum)
                                    :return-type :integer)
      (ff:defforeign 'accept_socket :arguments '(fixnum)
                                    :return-type :integer)
      (ff:defforeign 'close_socket :arguments '(fixnum)
                                   :return-type :integer)
      (ff:defforeign 'shutdown_socket :arguments '(fixnum fixnum)
                                      :return-type :integer)
      (ff:defforeign 'connect_socket :arguments '(fixnum simple-string fixnum)
                                     :return-type :integer)
      ))


;;; Creates a server process by creating a socket,
;;; and listen for incoming requests for connection
(defun make-socket-stream (&optional (messages t))
  "Create a socket when the Lisp process is the server."
  ;; create a socket
  (setq *socket-id* (create_socket))
  (if (= *socket-id* -1)
      (error "Unable to create socket~%" ))
    (if messages (format t "Created socket ~d.~%" *socket-id*))

    ;; bind the socket
    (setq *socket-port* (bind_socket *socket-id*))
    (if (= *socket-port* -1)
        (error "Unable to bind socket ~d.~%" *socket-id*))
    (if messages (format t "Bound socket ~d to port ~d~%" *socket-id*
                  *socket-port*))
    (if messages (format t "Listening at socket ~d~%" *socket-id*))

    ;; wait for a connection
    (setq ret-code (listen_socket *socket-id*))
    (if (= ret-code -1)
        (progn (format t "Error code ~d~%" ret-code)
               (error "Unable to listen socket ~d.~%" *socket-id*))))

;;; creates a client process
;;; Syntax: (connect-socket-stream "server-name" server-port-no)
(defun connect-socket-stream (server-name server-port-no
                              &optional (messages t))
  "Create a socket when the Lisp process is a client."
  ;; create a raw socket
  (setq *client-id* (create_socket))
  (if (= *client-id* -1)
      (error "Unable to create socket~%" ))
  (if messages (format t "Created socket ~d.~%" *client-id*))

  ;; connect to a server
  (setq *socket-port* server-port-no)
  (setq retno (connect_socket *client-id* server-name server-port-no))
  (if (= retno -1)
      (error "Error in connecting to socket~%"))
  (if messages (format t "Connected to server, file descriptor is: ~d~%"
                *client-id*))

  (setq *socket-connected* t)

  ;; make a Lisp stream for socket
  #+:LUCID
  (setq *socket-stream*
      (make-lisp-stream :input-handle *client-id*
                        :output-handle *client-id* :stream-type :ephemeral))
  #+:CMU
  (setq *socket-stream*
      (lisp::make-terminal-stream *client-id* *client-id*))

  #+:ALLEGRO
  (setq *socket-stream*
        (excl::make-buffered-terminal-stream *client-id* *client-id* t t))
)

;;; shutdown-socket-stream performs an orderly shutdown of the socket
;;; connection. It can be used during development & debugging when
;;; shutdown-io fails to shutdown sockets.
(defun shutdown-socket-stream ()
  "Shutdown a socket stream."
  (close_socket *socket-id*)
  (format t "Socket ~d closed.~%" *socket-id*)
  (setq *socket-id* nil)
  (setq *client-id* nil)
  (setq *socket-port* nil)
```

```
    (close *socket-stream* :abort t)
    (setq *socket-stream* nil)
    (setq *sockets-on* nil)
    (format t "Channel interface is shutdown.~%"))

(eval-when (eval compile load)
  (if (not *soar6-mode*) (soarsyntax)))

;;; The following statements are commented since cT is not used
;;; in our system -RLO (19/08/94)
;;  (and (not (member :ct *features*))
;;  (pushnew :ct *features*))


;;; The following functions tests the socket code by setting up a simple
;;; "Lisp server".  It creates a socket, accepts lisp sexprs from it,
;;; evaluates them and returns the result. It does no error checking,
;;; so is not robust if the form should contain an error. This is use in
;;; conjunction with "clienttest.out" (complied version of "clienttest.c")
;;; -RLO (31-08-94)

(defun set-up-socket-as-server ()
  "Creates a server process."
  (setq *sockets-on* t)
  (setq *debug-on* nil)
  (make-socket-stream))


(defun accept-client-and-make-socket-stream (&optional (messages t))
  "Accepts a client."
  (setq *client-id* (accept_socket *socket-id*))
  (when (and messages (= *client-id* -1))
        (format t "Client has not yet connected.~%"))
  (when (and messages (not (= *client-id* -1)))
        (format t "Connected to client, file descriptor is: ~d~%"
         *client-id*))

  #+:LUCID
  (unless (= *client-id* -1)
    (setq *socket-stream*
        (make-lisp-stream :input-handle *client-id*
                          :output-handle *client-id*
                          :stream-type :ephemeral)))

  #+:CMU
  (unless (= *client-id* -1)
    (setq *socket-stream*
        (lisp::make-terminal-stream *client-id* *client-id*)))

  #+:ALLEGRO
  (unless (= *client-id* -1)
    (setq *socket-stream*
        (excl::make-buffered-terminal-stream *client-id* *client-id* t t)))

  (unless (= *client-id* -1)
    (setq *socket-connected* t)))


(defun do-stuff ()
  "Do-stuff when socket is already connected."
  (when (= *client-id* -1)
        (accept-client-and-make-socket-stream))
  (unless (= *client-id* -1)
          (setq input (read-message-stream))
          (format t "Received message: ~a~%" input)
          (format t "Evaluated message: ~a~%" (eval input))
          (shutdown-socket-stream)))

;;; The following functions tests the socket code by setting up a simple
;;; "Lisp client".  It connects to a named socket and port-number.
;;; A list  is then sent to the server and printed out. This is used in
;;; conjunction with "servertest.out" (compiled version of "servertest.c").
;;; -RLO (31-08-94)

;; Syntax: set-up-socket-as-client "<server-name>" port-number
(defun set-up-socket-as-client (server-name port-number)
  "Creates a server process."
  (connect-socket-io server-name port-number))

(defun test-write ()
  "Writes a message to the server and prints it."
  (if *socket-connected*
      (progn
        (write-message 'hello-there)
        (write-message 'this-is-a-test)
        (close-socket-io))))
```

G.4 stdio.lisp

```
;;;; -*- Mode: Lisp -*-
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;
;;;;
;;;; File            : stdio.lisp
;;;; Author          : Blake Ward, Garret Pelton, Ralph Morelli, Joe Mertz
;;;; Created On      : Mon Mar 16 08:58:50 1992
;;;; Last Modified By: Roberto L. Ong <itxrlo@unicorn.nott.ac.uk>
;;;; Last Modified On: Thu Sep 15 13:12:33 1994
;;;; Update Count    : 3
;;;;
;;;; PURPOSE
;;;;
;;;; This file provides general functions to support Lucid, Allegro
;;;; and Common Lisp. It is meant to be used in conjunction with the
;;;; file "socket.lisp" and provide utilities such as reading, writing
;;;; from socket streams
;;;;
;;;; The following functions provide socket-support for SoarIO.
;;;;
;;;;     read-message-stream
;;;;             reads a "message" from the *socket-stream*.
;;;;     write-message sexpr stream
;;;;             writes a "message" to the stream.
;;;;
;;;; TABLE OF CONTENTS
;;;;   |>Contents of this module<|
;;;;
;;;; Sections:
;;;; (C) Copyright 1991, Carnegie Mellon University, all rights reserved.
;;;; (C) Other sections Copyright 1994, Roberto L. Ong and Frank E. Ritter
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;
;;;; Status
;;;; HISTORY
;;;;   ????????? - Unknown
;;;;   19-Aug-94 - RLO put all Lisp-socket utilities here.
;;;;               Modified some existing utilities and added
;;;;               some more.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;

(format t "~%loading stdio.lisp ..............~%")

;;; load file into default package - "USER"
(in-package "USER")

(defun read-message-stream ()
  "Read a list from the socket stream."
  ;; read alist
  (let ((inp nil))
    (if (listen *socket-stream*)
        (setq inp (read *socket-stream*)))
    (if *debug-on*
        (format t "~%debug: read-message-stream ~s" inp))
    inp))
```

```
(defun write-message (alist)
  "Writes a message to *socket-stream* iff a socket link is already
   established."
  (if *debug-on*
      (format t "~%|<>| send to Soar ~A |<>|~%" alist))

  (if *socket-connected*
      (progn
        (write alist :stream *socket-stream*)
        (finish-output *socket-stream*))
    ;;else
    (format t "Socket not connected, use connect-socket-io~%")))

(defun do-world()
  "Do stuff if a socket connection is already established."
  (if *socket-connected*
      (do ()(nil)
        (let ((message (read-message-stream)))
          (if (listp message)
              (world message)
            ;;else
            (format t "Caught a non-list message~%"))))
    ;;else
    (format t "Socket not connected, use connect-socket-io~%")))

(defun connect-socket-io (server-name server-port)
  "Connects to a client process iff there is no open socket."
  (if *socket-connected*
      (progn
        (format t "Closing down current connection first~%")
        (close-socket-io))
    ;;else
    (progn
      (connect-socket-stream server-name server-port)
      (setq *socket-connected* t))))

;; Modified this function to work on closing a
;; client socket (if Lisp is a client).
;; -RLO (15/09/94)
(defun close-socket-io ()
  "Close an open client socket."
  (if *socket-connected*
      (progn (close_socket *client-id*)
             (setq *socket-id* nil)
             (setq *client-id* nil)
             (close *socket-stream* :abort t) ;; still cause some error
             (setq *socket-stream* nil)
             (setq *socket-connected* nil))
    ;; else
    (format t "Socket not connected~%")))


(defun debug (message-string)
  "Debugs a message string."
  (case message-string
    (on (setq *debug-stdio* t))
    (off (setq *debug-stdio* nil))
    (otherwise
      (when *debug-stdio*
        (format t message-string)))))
```

```
;; The following utility functions are for setting up a process
;; that would read incoming data from Soar if there are any, and
;; evaluate them depending on the user's application.
;;
;; N.B. These functions are specially built for Lucid Lisp 4.1
;;      and Soar 6
;; -RLO (31-08-94)

(defvar *soar-read-loop-process* nil)

(defun launch-a-soar-read-loop-process ()
  "Spawn a process which is doing Soar interaction all the time
   RETURN the process."
  ;; If there was already a process running, kill it.
  (declare (special *socket-stream*))
  (when (lcl:processp *soar-read-loop-process*)
    (progn (lcl:kill-process *soar-read-loop-process*)
           (setq *soar-read-loop-process* nil)))
  (unless (connected-with-soar-p)
          (set-up-socket-as-server))
  (setf *soar-read-loop-process*
      (lcl:make-process :name "Soar-read-loop-process"
                        :priority 10000 ;; the higher this value,
                                        ;; the better and faster
                                        ;; in terms of speed
                                        ;; -RLO (31/08/94)
                        :function 'soar-read-loop-process)))

(defun soar-read-loop-process ()
  "Creates the body of the Soar read loop process."
  (lcl:handler-bind
   ((lcl::error #'lcl:invoke-debugger))
   (loop
    (setq *soar-read-loop-process* lucid::*current-process*)
    (when (not (connected-with-soar-p))
      (accept-client-and-make-socket-stream))
    (when (and (connected-with-soar-p)
               (listen *socket-stream*))
      ;; this is where incoming data is read and evaluated,
      ;; you can therefore add your own functions.
      ;; -RLO (31/08/94)
      (setq input (read-message-stream))))))

(defun connected-with-soar-p ()
  "test quick and dirty whether a connection exists"
  ;; should later be done with *features*
  (declare (special *client-id*))
  (and
   *client-id*
   (boundp '*client-id*)
   (not (eq -1 *client-id*))))
```