ICOT Technical Report: TR- 055

TR-055

SIMPOS: An Operating System for a Personal Prolog Machine PSI

by

Takashi Hattori, Junichiro Tsuji, and Toshio Yokoi

April, 1984

ICOT

Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan

(03) 456-3191 ~ 5
Telex ICOT J32964

Institute for New Generation Computer Technology

SIMPOS: An Operating System for a Personal Prolog Machine PSI

Takashi HATTORI, Junichiro TSUJI, and Toshio YOKOI

Institute for New Generation Computer Technology

Mita Kokusai Building 21F

1-4-28, Mita, Minato-ku, Tokyo 108, JAPAN

Abstract

SIMPOS (Sequential Inference Machine Programming and Operating System) for a Prolog based super personal computer PSI (Personal Sequential Inference Machine), which will be a workstation for Japan's Fifth Generation Computer project, is discussed, being focused around basic concepts of its operating system part.

The operating system has a three layer structure consisting of hardware resource management, basic constructs management, and i/o medium management. Hardware resource management is the kernel of SIMPOS, which manages a processor, memory, and i/o devices. Basic constructs management supports the supervisor facilities to provide a program execution environment by system defined classes, such as pool, stream, process, and world. I/o medium management has three subsystems -- file, window, and network subsystems -- to interface with the outside world.

A brief overview of both the hardware configuration of the PSI and the programming system part is also given.

Table of Contents

1. Backgrounds

SIMPOS (Sequential Inference Machine Programming and Operating System) is an operating system for the first version of SIM, which is named PSI (Personal Sequential Inference Machine). PSI will be a workstation for researchers of Japan's Fifth Generation Computer project, and is currently under development at ICOT (Institute for New Generation Computer Technology).

The project has been organized to develop Knowledge Information Processing Systems (KIPS) within 10 years. KIPS will be the system which combines technologies in inference and problem solving, knowledge base systems, and intelligent man-machine interfaces. From this point of view, PSI is also considered as a KIPS pilot machine and it is expected that many application programs will be implemented on PSIs.

2. Hardware Overview

PSI is a Prolog oriented personal computer [1]. We have already defined its machine level language, named Kernel Language version 0 (shortly KL0) [2], which is essentially an extended Prolog. PSI will execute directly the internal code of KL0. Except for Prolog's execution mechanism, that is, a predicate call which is executed by the unification and resolution of Horn clauses, KL0 specifies mainly the built-in predicates (instructions) which manipulate hardware specific features.

The hardware configuration of PSI will be as follows:

o a stack-based micro-programmed processor with cache memory

o main memory of 2 mega words, each word being 40 bits (32 bits for data and 8 bits for tag)

o a large bit-mapped display with a keyboard and a mouse (a pointing device)

o hard disks and floppy disks

o a local area network interface

o a console microprocessor to support the main processor in initial loading and diagnostics

The processor has a logical-to-physical address translation mechanism, though it does not support a virtual address space. The main memory is divided into 256 areas of 16 mega words. Each area is used as a heap or a stack. The execution of KL0 programs requires four stacks (a control stack, a local stack, a global stack, and a trail stack). Programs may occupy several heap areas. Therefore, PSI can have a maximum of 63 contexts executing at the same time. To reduce the overhead of context switching, the processor has 63 sets of the internal registers for process control. Interrupts and traps cause direct context switching at the hardware level, but process dispatching must be controlled by software.

3. Software Overview

SIMPOS is a programming and operating system designed for researchers who can also work as programmers. Our design principle of SIMPOS is simplicity both in concept and implementation, which leads to versatility of the system. Such features are desirable for systems like ours which require rapid prototyping, frequent modifications, and implementations of various application programs.

Another design criteria is to construct our system based on logic programming.

We have not had any experience in writing an operating system in a logic programming language. Therefore, our first and most important job was to decide what kinds of basic constructs are to be chosen to fulfil this criteria.

3.1 Operating System

The operating system part of SIMPOS consists of three layers:

- o hardware resource management
- o basic constructs management
- o i/o medium management

The hardware resource management, which may be called the kernel, fills a gap between the PSI hardware and the basic constructs management, which corresponds to the supervisor in conventional operating systems. It covers up hardware dependency and provides software-defined resources. It includes:

- o processor management which realizes a multiple processing environment
- o memory management which allocates and deallocates memory space, including garbage collection
- o input/output device management which controls the i/o devices

Basic constructs management provides the basic execution environment of programs, based on a class system enhancement made on KL0. When designing these basic constructs, we can take advantages of the fact that SIMPOS is a single language system, which means that we do not have to have various constructs applicable to each of many programming languages as a usual system does. We have selected several basic conceptual constructs, such as:

- o pools and streams

- o processes and worlds
- o channels and ports

These will be explained in detail later.

I/o medium management takes care of the interfaces to the outside world. This includes:

- o a file subsystem which manages the file system on disk devices
- o a window subsystem which interacts with a user, using a display, a keyboard, and a mouse
- o a network subsystem which communicates with other computers via local networks

These will be briefly explained later.

3.2 Programming System

The programming system part is organized from a collection of several programming expert processes, each of which helps programmers to develop and maintain their programs. They are called experts, in that they should know what programmers want to do at each stage of programming. These experts include:

- o listeners (command interpreters)
- o editors
- o compilers
- o debugging interpreters
- o librarians
- o analyzers

Coordination among these experts is necessary to provide an integrated programming system, where all the experts must cooperate. It is managed by another expert, a coordinator. What each expert does in the programming system is fairly clear, but how it does with each other's cooperations is not trivial. Though the programming system part is another big issue in designing SIMPOS, we will not discuss it further.

4. Basic Constructs

The basic constructs of the SIMPOS operating system will be discussed. A portion of the SIMPOS design may be thought as language design in the conventional system. However, since SIMPOS is a single language system, there is not a clear distinction between what a programming language should provide as its facilities and what an operating system should do. Those concepts which will be introduced in this section should be supported as an integrated system of the both.

4.1 Objects and Classes

Hardware resource management defines processes and areas whose use is not structured. A common practice in software to enhance productivity is to structure programs and data. Some currently available techniques are procedure oriented programming with abstract data types and object oriented programming [4], [5]. Although they are quite different in underlying concept of programming, what programmers get in programming seems similar in the sense that each data is defined as its structure and the operations allowed on it. So, two choices are open to us in extending Prolog as a system programming language.

Prolog is a procedure oriented, type-less programming language which interprets declarative Horn clauses as procedural programs. Hence, we may take the former approach as a natural choice. However, Prolog has a problem to be considered.

Since Prolog does not allow destructive assignment to a logical variable, we cannot describe state changes as changes in the value of a varible in the ordinary sense. On way to solve this problem is to create a new variable whenever the state has been changed. This is what we generally do when writing a program in Prolog. But as far as constructing an operating system where state change is essential to manage the resources in a system, we do not think this solution as effective because it requires much memory and it obscures identification of the resources. Instead we will introduce the concept of objects and classes into Prolog, which solves the problems of structured programming extension and state changes at the same time.

(1) Objects

An object represents and simulates something from the real world. It has a certain state which is changed as a result of some operations on it. An object in programming languages is defined by specifying its data structure and the operations available to it. The general mechanism for defining objects and executing the operations on them should be given by a system, but it is left to the programmers what should be objects and what operations should be defined for them.

In SIMPOS, each object is represented by a vector defined in KL0, and is identified by its location which is called an object pointer. Values which describe the state of an object are stored in the elements of the vector and can be changed by destructive assignments. An object pointer is allocated when the object is created and remains as a constant. This means that a logical variable of Prolog can hold an object pointer just as an integer or a symbol. In this way, we can introduce objects which have internal states, yet which can be manipulated in Prolog.

(2) Classes

The definition of an object requires specifying its data structure and the operations on it. However, there often exist many objects of the same kind, and it is unwise to define each of such objects. Classes are introduced as a convenient way to define these objects as a whole. After defining a class, every object of that class is created as an instance of the class.

In SIMPOS, the data structure of an instance is given as a template vector which may be represented as

```
{<code>,<attribute1>,<attribute2>,...},
```

where <code> is the code for the instance predicates and <attributeN> is a slot to hold a value of the internal state. Each attribute has its identifier which is used as the name of an accessing predicate. For example, if 'x' is the name of an x coordinate of a point, the predicate

```
X = Point!x
```

will give an x coordinate of Point to X.

An operation is defined as a set of Horn clauses. The form of a head is

```
:<action_name>(<object1>,<object2>,...),
```

where <action_name> is a predicate name which selects what operation should be taken on those objects <objectN>. Since an action name is generic, which clauses should be executed is determined conceptually by those classes of all the argument objects, but actually by the class of the first argument object. The form of a goal in a predicate body is

```
        :<action_name>(<object1>,<object2>,...),
```

where <action_name> and <objectN> are as defined above.

(3) Inheritance

Inheritance is a mechanism useful with class systems to enhance program modularity and extensionability. With multiple inheritance of classes, a programmer is allowed to define a new class from existing classes, so that it has the attributes and the operations defined in those classes, and if necessary, to add and/or change its specification as suitable.

If an instance of the new class needs additional attributes together with those of the inherited classes, it is sufficient to specify only these attributes in its definition.

Without redefining the inherited operations, an instance of the new class can accept them. When some new operations are necessary, they can be defined as usual. When an inherited operation is to be changed, a new definition of the operation can be given in this class, yet preserving the overridden definition. All the clauses which are defined in this class and which are inherited from its super classes are OR'ed.

The programming language ESP [3] which is a system description language for

SIMPOS will be specified on KL0, incorporating these class features.

4.2 Pools

We have introduced objects and classes. Now we will introduce the concept of places to store these objects. Objects may be stored in a place which we call a pool. A pool is thought of as a container for objects and objects of any class can be kept in a pool. Many pools can coexist in the system.

A pool itself is an instance of the class 'pool'. Operations allowed on pools are:

- o to insert an object into a pool
- o to extract an object from a pool
- o to find an object in a pool

4.3 Streams

Sometimes it is necessary to move an object from one place to the other. Streams provide a means to transport objects. A stream can be thought as a pipe through which objects flow. At one end of a stream is an input head and at the other is an output head. Objects which are put into a stream at the input head are to be retrieved at the output head in first-in first-out order. When there is no object in the stream, retrieve operations will be suspended until some object is put into the stream. In this way, streams can be used as a synchronization mechanism. SIMPOS uses streams as the only basic construct for synchronization and communication between processes.

A stream is an instance of the class 'stream' which defines the operations including:

- o to insert (put) an object into a stream
- o to extract (get) an object from a stream

An instance of the class 'tap' makes a stream out of a pool. Using a tap, it is possible to retrieve objects sequentially from a pool.

4.4 Processes

So far, we have not explained what an active entity in SIMPOS is. Object-oriented systems assume that each object is active in the sense that it has an internal state and responds to an incoming message by executing a method. This formalism has the feature that each object is uniformly treated of as an entity. But if we allow each object to act independently, which is necessary to some extent to realize concurrent execution, the overhead of managing these objects is prohibitive in conventional sequential machines. Hence, our approach is to introduce processes as only the active entities, and to treat objects as passive entities.

A process in SIMPOS is an entity to solve a question with a given program (rules and facts) and data (objects). A question is given as a predicate call. A process is created as an instance of the class 'process' and is manipulated by the operations which includes:

- o to suspend a process
- o to resume a process
- o to terminate a process

4.5 Worlds

The scope of global object references which a process can make is called a world. There may exist many worlds in a system (sometimes called a universe) and different processes may exist in different worlds. A world is a sequence of directories which are constructed as a pool of associations of objects and their names. A world can be created and changed dynamically, in order for a process to be able to acquire some new knowledge or to exchange a collection of knowledge with another.

A world can be shared by many processes. In that case, if a process changes the contents of a world, such changes are observable from another process which resides in the same world.

As a special kind of directory, we define permanent directories. A permanent directory is a directory which has a directory file (explained later in the file subsystem) as permanent storage of the included objects. Once an object is included, it is stored in a directory file at the same time. Therefore, this object can be retrieved, even after the system is rebooted.

4.6 Channels and Ports

Streams are the very basic means to implement inter-process communication. On top of streams, SIMPOS defines a higher-level communication construct, namely a channel. A channel allows message communication between two processes, even in different machines, by transporting an object packaged in a message of a certain protocol.

A port is where messages are put to send and are taken out to receive. Two channels connect two ports, so that a port can be used as a two-way communication message box. A port can be connected to many ports. In such a case, messages which are sent through the port will arrive at the ports connected to it, and messages sent from these ports will arrive at this port.

A special kind of port is called an i/o port. An i/o port is opened through i/o channels to the outer world, where a user, another machine, or a file is thought to exist. After creating an i/o port, a process can communicate with the outer world by sending or receiving messages, just as it does with another process through an ordinary port.

5. I/O Medium Systems

I/O medium systems take care of the interfaces with the outer world. We have three subsystems, which are file, window, and network subsystems. From the user program's point of view, they provide i/o ports with which user processes can communicate in order to input and output data. What each subsystem does for i/o operations is not relevant to these user processes. In some cases, an i/o subsystem does not perform any i/o operations, but simulates them in main memory. For example, a pool can be accessed as if it were an i/o device. Such detail realizations are concealed from user processes.

Each subsystem has a main process, which is called a manager. It holds a manager port which is connected to an i/o port held by a user process. The manager receives a user program i/o request and returns a reply through the manager port. What and how to do in response to i/o requests depends on each subsystem.

5.1 File Subsystem

The file subsystem provide permanent storages for data and objects.

(1) Files and Regions

A file is a container of records in a disk volume. The internal structure of a file depends on what kind of records are stored in it. We will support three types of files:

- o a binary file which does not assume any internal structure and can be used to store text or program code
- o a table file which stores fixed-length records
- o a heap file which stores variable-length records and is thought as a permanent heap area which resides on a disk volume

A region is an area in a volume, which physically stores a file. A region consists of disk pages which are allocated to it as it expands. A disk page has a physical address in a volume and a logical address which is identified by the region number to which it is allocated and an offset within the region. The address translation between these two addresses is done by using a page map associated with each region.

(2) Accessing a File

There are several file accessing modes:

- o random (or direct) accessing mode
- o sequential accessing mode
- o indexed accessing mode

In random accessing mode, each access is requested with a file marker which indicates the record position within a file. To access a file in sequential mode, a file tap is defined so that it should keep a file marker indicating the current record position. In indexed accessing mode, a record is identified by its associated key, rather than its position.

(3) Directory Files

Each file has its identifier, which a user specifies when finding it in a volume. If only one name space is allowed for file identifiers, name conflicts occur. To avoid these conflicts, a multi-level directory system is provided, where a file is identified with a pathname. A pathname is represented as, for example:

">dir1>dir2>file1"

A directory file is a file which associates files with their names. For example, the file specified above is identified by first finding a directory file with the name "dir1" in the root directory file, then finding a directory file with the name "dir2" in the directory file "dir1", and finally finding a file with the name "file1".

(4) Instance Files

An instance file is a table file to store objects. It is assumed that each class knows how to store and restore its instances in this file. A so-called VTOC (volume table of contents) is built as an instance file for file objects.

A directory file can also associate a record of this file with its name. The object "object" in the directory_file ">dir1>dir2" is specified by

">dir1>dir2>object"

just as a file is.

5.2 Window Subsystem

A man-machine interface based on multiple windows seems to be useful. It is now becoming prevalent in many new machines and we follow this approach in SIMPOS. A window is thought as a virtual terminal which has a screen, a keyboard, and a mouse. Each window represents an interactive session between a user and a process in the system. This means that multiple windows allow multiple concurrent sessions.

The window subsystem of SIMPOS will construct multiple window mechanism on its class system, as the LISP Machine does on its Flavor system [5]. The window subsystem provides both basic window classes and other classes which give a window some additional features, such as scrolling, labelling, and framing. A user can arbitrarily define a new kind of window using these classes.

(1) States of Windows

Logically there can be as many windows on a display device as necessary, but physically we cannot have these windows independent from each other because of the hardware limitation of the device. Therefore, to control and manage multiple windows, we define the states of a window:

o A shown window is fully exposed on the screen. It allows output and input from a mouse at anytime, but input from a keyboard may not always be allowed. Only one of the shown windows is called a selected window, which accepts input from the keyboard.

o A de-exposed window is a window which is partially exposed or fully covered by other windows. It cannot accept any input, but does allow output with certain restrictions.

o A de-activated window is not currently managed by the window system, so it is not shown on the screen.

The state transitions among these are controlled by user or program request.

(2) Subwindows

A process can create several subwindows within a window, each of which may show a different aspect of program execution information and states. For example, an editor may have subwindows, for text, command input, a command menu, and information about the current editing state. Note that since a window is regarded as a full screen to the subwindow, it cannot outgrow the parent window.

(3) Temporary Windows

A temporary window appears on demand, and disappears when it is no longer needed. It is used to show a message or menu. For example, a message has only to appear when a process wants to deliver it to a user, and it had better disappear when the user is done with it.

(4) Menu Windows

A menu provides a means for input by a mouse. A menu window shows a list of items, from which a user selects one item with a mouse. In general, a menu, if temporary one, is shown on a click of a specific button on the mouse, and when a user moves the mouse cursor to an item to be selected and clicks a button, the selected item is input through the window to the process. Then the menu disappears. If the user moves the mouse cursor out of the menu window, it disappears without any input.

5.3 Network Subsystem

Although super personal computers are powerful tools, it is difficult to share resources among researchers if these computers stand alone. A computer network is a solution to achieve a cooperative research environments with these computers. PSI will have a local network interface for this purpose.

The network subsystem provides the interface to communicate with other machines. Three levels of communication will be supported.

(1) Inter-machine Communication

Inter-machine communication is supported to connect a PSI with another PSI or a machine of another type. The network subsystem defines the following objects:

- o a node which represents each site in network
- o a socket which represents a connection point in a node
- o a cable which connects two sockets
- o a plug which is an access point to a cable

For inter-machine communication, a user program first creates a plug on an existing socket and connects it to another socket, usually on a different site, then reads or writes data to the plug.

(2) Inter-process Communication

Inter-process communication allows two processes on different nodes to communicate with each other as if they existed on the same node. The network subsystem defines remote channels for this purpose. A remote channel represents a channel on a remote node. A process can send a message to this remote channel, and another process on the remote node can receive it from the channel represented by this remote channel.

(3) Remote Object Operations

We believe that an operating system, which supports a computer network and object-oriented framework in programming, should provide a means to deal with an object in the system of a remote site. A solution which SIMPOS takes is an introduction of remote objects.

A remote object is an object which represents an object in a remote node. The operations on a remote object have the same interface as the operations on an ordinary object in a self node, but the implementation of them are quite different. An operation on a remote object is encoded into a message, the network subsystem in a self node sends it to the one in a remote node, and the message is decoded into an operation on the specified object.

Currently we assume that each class knows how to encode and decode the operations and the messages. The network subsystem uses it to transfer and perform the operations on a remote object.

6. Conclusions

So far, we have described a brief overview of SIMPOS and the design approaches we have taken. There may be still many concepts left to be clearly specified, and we do not expect the current design and implementation to be final. Rather we will refine it further and develop other system and application programs, so that we can achieve a pilot system of KIPS with PSI and SIMPOS.

We have finished the phase of functional design of SIMPOS development and have started coding and debugging SIMPOS in ESP on a development system. As the first prototype version of PSIs is now available, we will transfer the programs developed on the development system to PSIs and release the first version by the end of the 1984 fiscal year.

Acknowledgements

We would like to thank all the members of ICOT who have been working with us to design the operating system part of SIMPOS. Some of them are T.Kurokawa, K.Sakai, T.Chikayama, S.Takagi, and A.Taguchi.

References

[1] H.Nishikawa, et al., "The Personal Inference Machine (PSI): Its design philosophy and machine architecture", ICOT TR-013 (June 1983).

[2] T.Chikayama, et al., "Fifth generation kernel language version 0", ICOT internal document (June 1983).

[3] T.Chikayama, "ESP Reference Manual", ICOT TR-044 (Feb. 1984).

[4] A.Goldberg, D.Robson, Smalltalk-80: The language and its implementation, Addison-Wesley (1983).

[5] D.Weinreb, D.Moon, "Flavors: Message passing in the lisp machine", MIT A.I.Memo No.602 (November 1980).