TM-0009

Basic Constructs of the SIM Operating System

by
Takeshi Hattori
and Toshio Yokoi

June, 1983

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

Basic Constructs of the SIM Operating System

Takashi HATTORI and Toshio YOKOI
Institute for New Generation Computer Technology (ICOT)
1-4-28, Mita, Minato-ku, Tokyo 108

Abstract

The basic constructs of SIMPOS (Sequential Inference Machine Programming and Operating System) are explained. SIMPOS is an operating system for a super-personal computer (SIM), based on a logic programming language (a modified Prolog, called KL0). Our design principle is the simplicity both in concepts and the structure. The entire system will be constructed using these basic constructs.

1. Objects and Relations

1.1 Objects

The elements of the system (also called the universe) are objects, which are classified into elementary objects, compound objects, and variable objects.

An elementary object is an atomic object without any structure, such as a symbolic atom (e.g. abc), an integer atom (e.g. 12), or a real number atom (e.g. 4.5), which is an elementary term in Prolog.

A compound object is constructed with its subobjects, which are objects themselves, and is described as a compound term in Prolog:

<class-name>(<subobject1>,...,<subobjectN>),

where <class-name> is a symbolic atom.

A variable object, which is represented by a logical variable, is an object whose value is initially undefined and can be bound later. The variable objects have two basic operations, a unification and an (destructive) assignment.

1.2 Relations

Relations are defined among objects. A relation is a set of object-tuples and is represented as a predicate

<relation-name>(<object1>,...,<objectN>).

Each relation is defined with the clauses as facts and/or rules.

fact: <relation-name>(<object1>,...,<objectN>).

rule: <relation-name>(<object1>,...,<objectN>) :-
<predicate1>,...,<predicateM>.

A clause is considered as a meta-object whose class name is ':-'.

fact: :-(<head>,true)

rule: :-(<head>,body(<goal1>,...,<goalN>))

where <head> and <goalI> are predicates.

2. Classes

The class concept is introduced as a means of data abstraction. A class is a set of the objects which have the same features. These features are given by the data structure of those objects (an instance template of the class) and the relations they should have.

As found in the Flavor system of the Lisp machine 1), multiple inheritance mechanism is supported in our class system. In a relation-based system, inheritance is not so simple as in a functional system, though. The classes which a new class should inherit are said to be its component classes. It means that a new class can be built from many other classes as its components.

An instance template is used to create an object of the class and to access the subobjects of the object. It has the following form:

<class-name>(<subobject1>,...,<subobjectN>),

where <subobjectI> is a variable object.

An instance template definition is given as

new(<class-name>:class,
<subobject-names>,
<component-class-names>,
<required-subobject-names>,
<required-component-class-names>),

where <subobject-names> is a list of the (slot) names of the

subobjects and <component-class-names> is a list of the names of the component classes which this class should inherit. <Required-subobject-names> and <required-components-class-names> are what are used in this class, but must be defined elsewhere.

A relation among the objects describes the action which is taken when the corresponding predicate is called on those objects. A relation which the instance has to have is defined by the clauses of an extended form:

<relation-name>(<variable1>:<class-name1>,...,
<variableM>:<class-nameM>) :-
<predicate1>,...<predicateL>.

Here, <class-nameI> specifies the class of the argument object <variableI>. When a relation is defined on some class, a new class which inherits it can have that relation implicitly. Of course, the new class can have its own relations, which may or may not use the inherited relations.

3. Pools and Streams

3.1 Pools

A pool is a container which can hold many objects. Some operations on a pool, which is an instance of the class 'pool', are

* to insert an object into a pool,
* to extract an object out of a pool,
* to find an object in a pool.

We define a tap which is plugged in a pool and is used when sequential access is necessary to the objects in a pool.

3.2 Streams

A stream is a pipe through which objects flow. Some

operations on a stream, which is an instance of the class 'stream', are

* to put an object into a stream,
* to get an object out of a stream.

When a stream is empty, a get operation is suspended until a put operation is performed.

4. Worlds

A world is a scope of the knowledge which can be used to solve a given question, and many worlds can coexist in the universe. A world is created dynamically from many pools, each of which contains some data or programs. We can also include other worlds to be elements of the new world, which we call a combined world. This kind of mechanism is useful for creating a hypothetical world on the existing world.

A world is a meta-object, an instance of the class 'world'. The pools which are included in a world are listed in the pool list of the world. An object in the world is searched in the order of the pool list.

5. Processes

A process is an active entity which solves a given question in a specified world. A process can be created dynamically, and terminates when it cannot solve the question, or it finds an answer which is the only answer or the last one. It is suspended when it finds an answer, but there may exist other answers. In this case, the process has two options which it can take later, that is, it may try to find another answer, or it may terminate.

5.1 Process Creation

SIMPOS supports two useful ways of process creation, a fork

operation and a remote call operation.

A fork operation is described as, for example,

....,<process-name1>::<question1> !!

<process-name2>::<question2>,...

as found in Relational Language 2) or Concurrent Prolog 3). The above operation creates two child processes. The process with <process-name1> as its identifier tries to solve <question1> and the process with <process-name2> tries to solve <question2>. After forked, the parent process waits until all its child processes terminate. At that time, a join operation is performed, and the parent process resumes its execution. However, if the parent process does not have any other goals left to solve, it can terminate when forked.

A remote call is a predicate which asks a question about another world. In order to solve such a question, a process has to be created in that world. SIMPOS creates a surrogate process implicitly to implement a remote call, although a user may not think he/she creates a new process. A remote call is represented as

...,<question> @ <world>,...

which means solving <question> in <world>. Note that the caller will be suspended until an answer is returned from the callee. The call can be backtracked in the sense that we can request alternative answers.

A process, after created, can communicate with other processes through streams, and can cooperate with each other to solve a question. A stream is the only basic mechanism for inter-process communication.

5.2 Process Operations

A process is considered as a meta-object, and is created as an instance of the class 'process'. It accepts such operations as

* process initialization
* process termination
* process suspension
* process resumption

6. Ports and Channels

A stream is suitable to implement a one-way communication, a semaphore, or a synchronization. However, it is often necessary to provide a means of two-way communications between two processes. A channel, which is created by connecting two ports, is such a mechanism to allow a process to communicate with another process in two directions. A process holds a port which works as an input/output gate, and sends or receives an object through the port.

A port is allowed be connected to many ports. In this case, the object which is sent through the port arrives at every port which is connected to it, and the object sent from these ports arrives at this port.

We define a special kind of a port, that is, a source/sink port. It works as a hole into the outer world, through which a process can access the objects there. Conceptually there is a pseudo-process in the outer world, for example, a user at the terminal, another machine connected through computer networks, and so on. The process sets up the open-ended channel with such a pseudo-process. The i/o subsystems, which are not explained

here, take care of the actual i/o operations.

7. Conclusions

We have here defined several basic concepts, which we hope are sufficient to build the entire operating system (SIMPOS). We are currently at the functional design stage of the development, and are writing down its specification in a semi-formal notation which looks like a logic programming language and can be converted easily into the executable codes. These basic concepts may be modified to incorporate better ideas and implementation details, though. The first version of SIMPOS will be hopefully finished in the fall of 1984.

Acknowledgements

We would like to thank each member of 3rd Lab. of ICOT for their participations in the conceptual design of SIMPOS.

References

1) D.Weinreb and D.Moon: Flavors: Message Passing in the Lisp Machine, MIT A.I. Memo No.602 (November 1980).

2) K.L.Clark and S.Gregory: A Relational Language for Parallel Programming, Imperial College, Department of Computing, Research Report DOC 81/16 (July 1981).

3) E.Y.Shapiro: A Subset of Concurrent Prolog and Its Interpreter, ICOT Technical Report TR-003 (January 1983).