ISE-TR-87-63

A LOOSELY COUPLED MULTIPROCESSOR SYSTEM : ADMS
——SOFTWARE ENVIRONMENT——

by

Sanae Amada

Masamitsu Baba

Norio Ohashi

May 26, 1987

INSTITUTE
OF
INFORMATION SCIENCES AND ELECTRONICS

UNIVERSITY OF TSUKUBA

A LOOSELY COUPLED MULTIPROCESSOR SYSTEM : ADMS

----- SOFTWARE ENVIRONMENT -----

by

Sanae AMADA*, Masamitsu BABA**, and Norio OHASHI***

* Institute of Information Sciences & Electronics,
University of Tsukuba.

** The Matsushita Communication Industry Co. Ltd..

*** The Nippon Electric Co. Ltd..

Abstract

The specification of the system implementation language, the outline of the intermediate language, and the software tools for ADMS are described in this paper. Namely, this paper is a part of serial reports on ADMS.

1. INTRODUCTION

Concerning with the system implementation language for ADMS (SIL-ADMS), the main target of the development of the language is the description of OS for ADMS. However, we tried that the language will appropriate to describe many application programs.

While, we prepare an intermediate language for the easy development and portability of software resources.

We prepare many functions needed as commands of OS by a set of libraries. A file system is now available in a level of the specification, and needed tools are listed up.

In this paper, we'll report on above mentioned problems.

2. SIL-ADMS

We prepare following objects which the system can be recognized.

* System management objects. (Processor, Process, Storage)
* Execution management objects. (Instruction, Data, Domain, Context)
* Inter-process communication objects. (Message, Port)
* Access management objects. (Type, Template)

2.1 The Object

The object is a set of related information and is defined by a set of operations applicable on itself, and internal construction of it is hidden and protected. In ADMS, programs and hardware resources are classified into objects and managed by the system.

a. Declaration

All sorts of objects must be declared with an identification name of the sort. These objects can be declared as types. This is not needed to apply on only one object.

```
ex. type stack=object . . .
    type cardreaders=process(     );
         cardreader_1:cardreaders;
```

b. Creation

By the declaration all objects are created. The object with type is created by naming concretely.

c. Destruction

Executive, or active objects are destroyd when the last statement has been executed and the control has moved to the system. Non-executive, or passive objects as abstract data objects must be destroyd with some instructions. The space occupied with the concerned object is re-used after the destruction by the function of the system.

2.2 The Process

The process is a concurrently executable object. It is needed to describe the concurrency when processes are executed asynchronously in parallel just like OS. In our language, the process must be defined and declared explicitly.

a. Declaration

The process is an active object, and must be declared just like described in 2.1.a. If the process has an inner data structure, and is protected and managed as the abstract data in the form of the object, we must express concerning points in the program.

The scope clarifies the right of data managed by the process.

```
ex.  cardreader=process(      )
         var
         buffer=object
         end
     end
```

b. Creation

Same to 2.1.b.

c. Activation

By the cobegin statement which will be written later, the process is activated in the main routine of the parent process or the main program.

d. Termination

The executing process comes to terminate when the control has arrived to its end. After then, the control moves to the proper point of the program.

e. Synchronization/Communication

This function is realized by the inter-process communication procedure given later.

f. Nesting Construction of the Process

We admit the nesting construction to the process. As a result, we have a parent process and a child process. The parent process can terminate after the termination of all children processes. The creation of the child process occurs after the creation of the parent process.

g. Declaration of the Entry

To use for the inter-process communication, the entry used in the process must be declared as the interface part of the process with the reserved word "entry".

```
ex.  entry  out (      ),
            in  (      );


     process  X  X  (      );
        entry  X  X  X ;
        var    X  X  X  X  X ;
                .   .   .
     end
```

2.3 The Designation of the Non-Determinacy

We use a reformed select statement to express the producer/consumer problem, etc.. Two forms are prepared.

a. Waiting for the Selection

A non-determinate branch is expressed, and some functions as the election of one in plural branches and the waiting by an input statement in the branch is feasible.

```
ex.  select
        when  FLAG           ->  .  .  .
        or when receive (      )
                             ->  .  .  .

        else  .  .  .
     end select
```

The imput statement in the guard comes to the true when it can execute immediately. After all elements in the guard come to the true and statements following to the guard are selected, the input statement is executed, and then, these statements are executed. There are some differences from the select statement in Ada.

b. Immediate Call

This procedure realizes the control flow in which the output statement can execute only in the case of the communication is allowed at once.

```
ex.  select
         send (      );  . . . .
         else   . . .
     end select
```

2.4 The Designation of the Parallelism

In our language, the parallelism can be designated explicitly, and this function is useful to catch the flow of the control. The function has a role to activate the process. We use a cobegin statement for this purpose.

```
ex.  cobegin Pi(  ); Pj(  ); . . .  . Pn(  ) end
```

In the example, Pi~Pn shows the name of each process. The cobegin block comes to the termination after the termination of all processes in the block. The order of description of the process is optional, but the nesting of the block can not be allowed. In case of the nesting of the process, the parent process can designate the activation of the child process. If the parent process has only one child process, we describe the program in the form of sub-program, because the child process can not be executed at a time with the parent process. If the parent process has children processes and we need the concurrency on them, we can designate the activation of each child process by the cobegin statement.

2.5 Inter-Process Communication

a. Entry of the Process

All distinctionable entries of the process must be declared for their names and types. On the standard input or output media, the application of the rule is not needed.

b. Link of Entries

The entry declared in the process must be linked to the target for the communication. The link is realized by the connection of both entries, and this procedure is described in an initialization part in a main program.

ex. connect sender. out > receiver, in

As basic instructions for the inter-process communication we prepare following three pairs.

* send/receive (for the synchronous communication)

A process which has executed these instructions, comes to the block status and waits the finish of execution of the corresponded instruction in the target process. After the matching of both instructions, it comes the transfer of data, and then both processes are re-activated.

* put/get (for the asynchronous communication)

A process which has executed these instructions, is kept in its activated status.

* ecall/accept (for the communication with reply)

The sender process comes to the block status until the receiver process sends out the reply.

2.6 The Mode of the Inter-Process Communication

By the use of above mentioned instructions, we can have following communication modes.

* Block mode same to CSP. [Hoa78]
* Conditional mode.
* Non-block mode.
* Reply mode.
* Non-block mode with the interrupt.
* Conditional reply mode.
* Condition check mode (to use for the realizing of select statement).

3. INTERMEDIATE LANGUAGE

3.1 Character of the Language

Our intermediate language is based on the model of the Actor theory. [Hew77] [Yon79] Because, the model seems to be fitted to the image of distributed processing, and can apply to describe the parallelism. We put following limitation on it.

* We set the acquaintance for each sort of message, not for the Actor. This rule has the advantage as to limit the flow of message and to inhibit the sending of unjust message to the Actor.
* We are on the premise that the every communication needs the reply. When the sender process needs no reply as the

result of the execution in the receiver process, the sender will need the information of end of the execution in the receiver.

* We don't allow any dynamic creation or destruction of the Actor. Of course, the addition of some new Actors is allowed.

We call the Actor which is limited in its functions as mentioned above as the Capsule. The acquaintance of the Capsule has the name of the target Capsule and the name of type of the message which the target Capsule can receive.

3.2 The Outline of Instructions

The execution in the Capsule is carried out by a stack-machine. SIL-ADMS is similar to Pascal and congenial to a stack-machine. The machine has a script part, or a code segment which contains the instruction corresponded to the executing statement of the program, and an inner status, or a data segment which contains data and an acquaintance.

The instruction set is same to the P-code of Pascal except described in the next paragraph and 3.4.

3.3 The Instruction for the Communication

Next three instructions are added.

* To send the message and to require the reply.
* To receive the message.
* To answer to the requirement.

A message is able to have data and an acquaintance. The first instruction contains two types. One designates the target directly, and another designates the target with the import of the top of stack. The former has an acquaintance and a type of reply, and by the execution the message is put on the top of stack. The later has a type of reply, and by the execution the message and the acquaintance of the target are pushed on the stack. In both cases, the message is popped and the reply is pushed after the execution.

The second instruction contains two types, too. One designates the pattern of message directly, and another designates it with the import of the top of stack.

In the third instruction, the message of the reply and the acquaintance are pushed before the execution.

3.4 Other Modifications

The standard function and procedure on the file, and an instruction "End of File" are struck off, because we have no concept of the file in our intermediate language. Following instructions are added; shift right logical, shift left logical, bit and, bit inclusive or, bit exclusive or. The properness of the instruction set is under inspection in present.

4. SOFTWARE TOOLS

As software resources for ADMS, we must have many tools.

4.1 File System

We are planning to have a file system similar to that in UNIX*. It ahs a tree configuration, and for the designation of each file we use a pass name and a file name. Input and output devices can be registered as files.

The communication between users is realized by the transfer of the content of communication to the home directo-

* UNIX is a software developed by AT&T.

ry of the receiver. The output from the home directory to the higher ranked one is regarded as the output to all directories ranked lower than that. The rule has effects as;

* The concept of input/output is woven into the directory.
* The communication from one to many is realized with ease.

To the protection of the file, we use in combination with a method in which an access right is established and a method by a mode. The mode means a supervisor mode, a programmer mode, and a user mode. The system assigns the user number and the group number to the user.

4.2 Tools

We will need following tools.

Text editor.

Pretty printer.

Compiler.

Linker/Loader.

Analyzer.

Debugger.

Command interpreter.

The management tool for the file system.

The mode management tool.

The library management tool.

We are planning to write these tools with SIL-ADMS. Still more, many libraries are needed for ADMS, and we have listed up them.

5. CONCLUSION

In this paper, we wrote on the system implementation language, the intermediate language, and tools for ADMS.

This is one of our continued reports [Ama86a][Ama86b], and we are scheduling to write on the hardware in near future. And we are planning to evaluate the performance of ADMS as the next work.

ACKNOWLEDGEMANT

We thank to Mr. Jun-ichiro OHGAKI, who analyzed about the file system and the needed tools and libraries. And we thank to Mr. Toshinori SONEHARA, Mr. Masahiko SAWABE, and Mr. Masashi SHINOHARA as cooperative investigators of our work.

References

Ama86a: S.Amada, M.Tsuchida, Y.Sato; A Loosely Coupled Multi-processor System : ADMS ---Basic Design---, Tech. Rep. of Inst. of Inf. & Electronics, Univ. of Tsukuba, ISE-TR-86-56, pp.1-16, (June 1986).

Ama86b: S.Amada, Y.Sato, S.Suzuki; A Loosely Coupled Multi-processor System: ADMS ---Logical Configuration & Operating System---, Tech. Rep, of Inst. of Inf. & Electronics, Univ. of Tsukuba, ISE-TR-86-59, pp.1-27, (Oct.1986).

Hew77: C.Hewitt; Viewing Control Structures as Patterns of Passing Messages, J. of Art. Intelligence, Vol.8, pp.323-364, (1977).

Hoa78: C.A.R.Hoare; Communicating Sequential Processes, Comm. of ACM, Vol.21, No.8, pp.666-677, (Aug. 1978).

Yon79: A Tutorial on ACTOR Theory, J. IPS Japan, Vol.20, No. 7, pp.580-589, (1979)(Japanese).

INSTITUTE OF INFORMATION SCIENCES AND ELECTRONICS
UNIVERSITY OF TSUKUBA
SAKURA-MURA, NIIHARI-GUN, IBARAKI 305 JAPAN

REPORT DOCUMENTATION PAGE	REPORT NUMBER ISE-TR-87-63
TITLE A Loosely Coupled Multiprocessor System : ADMS ----- Software Environment -----	
AUTHOR(S) Sanae Amada Masamitsu Baba Norio Ohashi	
REPORT DATE May 26, 1987	NUMBER OF PAGES 11
MAIN CATEGORY Multiprocessor System	CR CATEGORIES C.1.2, C.1.3, D.1.3, D.3.2,
KEY WORDS Distributed Processing, Multiprocessor, System Implementation Language, Intermediate Language.	

ABSTRACT

The specification of the system implementation language, the outline of the intermediate language, and the software toos for ADMS are described in this paper. Namely, this paper is a part of serial reports on ADMS.

SUPPLEMENTARY NOTES