INFORMATION TO USERS

This reproduction was made from a copy of a document sent to us for microfilming. While the most advanced technology has been used to photograph and reproduce this document, the quality of the reproduction is heavily dependent upon the quality of the material submitted.

The following explanation of techniques is provided to help clarify markings or notations which may appear on this reproduction.

1. The sign or "target" for pages apparently lacking from the document photographed is "Missing Page(s)". If it was possible to obtain the missing page(s) or section, they are spliced into the film along with adjacent pages. This may have necessitated cutting through an image and duplicating adjacent pages to assure complete continuity.

2. When an image on the film is obliterated with a round black mark, it is an indication of either blurred copy because of movement during exposure, duplicate copy, or copyrighted materials that should not have been filmed. For blurred pages, a good image of the page can be found in the adjacent frame. If copyrighted materials were deleted, a target note will appear listing the pages in the adjacent frame.

3. When a map, drawing or chart, etc., is part of the material being photographed, a definite method of "sectioning" the material has been followed. It is customary to begin filming at the upper left hand corner of a large sheet and to continue from left to right in equal sections with small overlaps. If necessary, sectioning is continued again—beginning below the first row and continuing on until complete.

4. For illustrations that cannot be satisfactorily reproduced by xerographic means, photographic prints can be purchased at additional cost and inserted into your xerographic copy. These prints are available upon request from the Dissertations Customer Services Department.

5. Some pages in any document may have indistinct print. In all cases the best available copy has been filmed.
Implementation of the model base concept in SIMSCRIPT II.5: Application to computer network design

Chen, Chun-Ting, M.S.
The University of Arizona, 1987
PLEASE NOTE:

In all cases this material has been filmed in the best possible way from the available copy. Problems encountered with this document have been identified here with a check mark ✓.

1. Glossy photographs or pages
2. Colored illustrations, paper or print
3. Photographs with dark background
4. Illustrations are poor copy
5. Pages with black marks, not original copy ✓
6. Print shows through as there is text on both sides of page
7. Indistinct, broken or small print on several pages
8. Print exceeds margin requirements
9. Tightly bound copy with print lost in spine
10. Computer printout pages with indistinct print
11. Page(s) lacking when material received, and not available from school or author.
12. Page(s) seem to be missing in numbering only as text follows.
13. Two pages numbered. Text follows.
14. Curling and wrinkled pages
15. Dissertation contains pages with print at a slant, filmed as received
16. Other

University Microfilms International
IMPLEMENTATION OF THE MODEL BASE CONCEPT
IN SIMSCRIPT II.5: APPLICATION TO
COMPUTER NETWORK DESIGN

by
Chun-Ting Chen

A Thesis Submitted to the Faculty of the
DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING
In Partial Fulfillment of Requirements
For the Degree of
MASTER OF SCIENCE
WITH A MAJOR IN COMPUTER ENGINEERING
In the Graduate College
THE UNIVERSITY OF ARIZONA

1987
STATEMENT BY AUTHOR

This thesis has been submitted in partial fulfillment of requirements for an advanced degree at The University of Arizona and is deposited in the University Library to be made available to borrowers under rules of the Library.

Brief quotations from this thesis are allowable without special permission, provided that accurate acknowledgment of the source is made. Requests for permission for extended quotation from or reproduction of this manuscript in whole or in part may be granted by the Dean of the Graduate College when in his or her judgment the proposed use of the material is in the interests of scholarship. In all other instances, however, permission must be obtained from the author.

SIGNED: Chuntai Chen

APPROVAL BY THESIS COMMITTEE

This thesis has been approved on the date shown below:

B. P. Zeigler

Professor of Electrical and Computer Engineering

July 25/87
ACKNOWLEDGMENT

I wish to express sincere gratitude to my advisor, Dr. B. P. Zeigler, for his patient guidance and encouragement throughout this project.

I also express my gratitude to Dr. R. Martinez and Dr. J. W. Rozenblit, members of my graduate committee for reading and constructively criticizing this thesis.

Thanks are extended to the faculty of the Department of Electrical and Computer Engineering and fellow graduate students for making this graduate program an educational and unforgettable experience.

Last, thanks are expressed to my parents and my wife for their encouragement and support.
TABLE OF CONTENTS

LIST OF ILLUSTRATIONS ...................... vi
ABSTRACT ........................................ viii

CHAPTER

1. INTRODUCTION ............... 1
   1.1 Model Base Concept ........... 1
   1.2 Application to Computer Network Design .................. 4

2. MODEL BASE FRAMEWORK ............. 7
   2.1 Model Base Entry: I/O Port Description ............... 8
   2.2 I/O Port Communication ........... 10
   2.3 Couple Scheme Algorithm .................. 14
   2.4 File Organization and Entity Structure ............... 17

3. FORMAL MODEL DESCRIPTION ........... 18
   3.1 CRT Module Description ........... 18
   3.2 Interface Module Description ........... 22
       3.2.1 Packet Communication Unit ........... 23
       3.2.2 Token Interface Module ........... 29
       3.2.3 Gateway Module ........... 30
   3.3 Host Computer Module ........... 32
   3.4 Printer Module ........... 35
   3.5 Transmission Medium Module ........... 37

4. SIMSCRIPT IMPLEMENTATION ........... 38
   4.1 Implementation Facilities ........... 38
   4.2 Output and Input Ports ........... 38
   4.3 Organization of Model Base ........... 46
       4.3.1 Files for Each Model ........... 46
       4.3.2 How to Build Coupled Model ........... 48
   4.4 Examples ........... 53

iv
TABLE OF CONTENTS—Continued

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5. SUMMARY</td>
<td>54</td>
</tr>
<tr>
<td>APPENDIX A: SUB DOMAIN OF COMPUTER NETWORK</td>
<td>56</td>
</tr>
<tr>
<td>APPENDIX B: SIMSCRIPT CODE FOR DIRECTED-GRAPH COUPLING</td>
<td>61</td>
</tr>
<tr>
<td>APPENDIX C: SIMSCRIPT CODE FOR BROADCAST COUPLING</td>
<td>67</td>
</tr>
<tr>
<td>REFERENCES</td>
<td>69</td>
</tr>
</tbody>
</table>
LIST OF ILLUSTRATIONS

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Design in the Multifaceted Modelling Context.</td>
<td>1</td>
</tr>
<tr>
<td>2.</td>
<td>Composition tree and hierarchically specified model.</td>
<td>2</td>
</tr>
<tr>
<td>3.</td>
<td>Intelligent simulation environment.</td>
<td>3</td>
</tr>
<tr>
<td>4.</td>
<td>Modular discrete model.</td>
<td>7</td>
</tr>
<tr>
<td>5.</td>
<td>Coordinator for a hierarchical system.</td>
<td>9</td>
</tr>
<tr>
<td>6.</td>
<td>Coupled model.</td>
<td>12</td>
</tr>
<tr>
<td>7.</td>
<td>Coordinator module algorithm.</td>
<td>13</td>
</tr>
<tr>
<td>8.</td>
<td>I/O table</td>
<td>15</td>
</tr>
<tr>
<td>9.</td>
<td>Pruned sub-entity structure</td>
<td>16</td>
</tr>
<tr>
<td>10.</td>
<td>Atomic module</td>
<td>17</td>
</tr>
<tr>
<td>11.</td>
<td>Packet format</td>
<td>18</td>
</tr>
<tr>
<td>12.</td>
<td>CRT module.</td>
<td>20</td>
</tr>
<tr>
<td>13.</td>
<td>PCU module.</td>
<td>24</td>
</tr>
<tr>
<td>14.</td>
<td>TIM module.</td>
<td>29</td>
</tr>
<tr>
<td>15.</td>
<td>Gateway module.</td>
<td>31</td>
</tr>
<tr>
<td>16.</td>
<td>Host computer module.</td>
<td>33</td>
</tr>
<tr>
<td>17.</td>
<td>Printer module.</td>
<td>35</td>
</tr>
<tr>
<td>18.</td>
<td>Bus module.</td>
<td>37</td>
</tr>
<tr>
<td>19.</td>
<td>Internal coupling</td>
<td>39</td>
</tr>
<tr>
<td>20.</td>
<td>Data structure for CRT model.</td>
<td>40</td>
</tr>
<tr>
<td>Figure</td>
<td>Description</td>
<td>Page</td>
</tr>
<tr>
<td>--------</td>
<td>------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>21.</td>
<td>CRT input port.</td>
<td>40</td>
</tr>
<tr>
<td>22.</td>
<td>Coordinator for CRT module.</td>
<td>41</td>
</tr>
<tr>
<td>23.</td>
<td>Atomic model implementation</td>
<td>42</td>
</tr>
<tr>
<td>24.</td>
<td>Directed-graph coupling</td>
<td>42</td>
</tr>
<tr>
<td>25.</td>
<td>Directed-graph coupling coordinator implementation</td>
<td>43</td>
</tr>
<tr>
<td>26.</td>
<td>Broadcast coupling</td>
<td>44</td>
</tr>
<tr>
<td>27.</td>
<td>Broadcast coupling coordinator implementation</td>
<td>45</td>
</tr>
<tr>
<td>28.</td>
<td>Part of the HC I/O table</td>
<td>48</td>
</tr>
<tr>
<td>29.</td>
<td>The complete HC I/O table</td>
<td>48</td>
</tr>
<tr>
<td>30.</td>
<td>Coupled model: workstation</td>
<td>49</td>
</tr>
<tr>
<td>31.</td>
<td>Result of CRT joined to PC</td>
<td>49</td>
</tr>
<tr>
<td>32.</td>
<td>Result of CRT join PC join PTR</td>
<td>50</td>
</tr>
<tr>
<td>33.</td>
<td>Coupled model: LAN</td>
<td>50</td>
</tr>
<tr>
<td>34.</td>
<td>Coupled model: Internet</td>
<td>52</td>
</tr>
<tr>
<td>35.</td>
<td>Result of LAN/CSMA join Gateway</td>
<td>52</td>
</tr>
<tr>
<td>36.</td>
<td>Result of LAN/TOK join Gateway</td>
<td>52</td>
</tr>
<tr>
<td>37.</td>
<td>Input parameters</td>
<td>53</td>
</tr>
<tr>
<td>38.</td>
<td>Simulation results, 3 Terminal room, 1 Host computer room</td>
<td>53</td>
</tr>
</tbody>
</table>
ABSTRACT

This thesis presents an implementation of the model base concept in SIMSCRIPT II.5. It shows two principles in designing a flexible general simulation system. First of all, the input and output port intercommunication and synchronization should be done by an I/O coupling scheme, i.e. a coordinator. Second, every input port should be a separate file in the model base. Users can retrieve these files to couple them together and evaluate the simulation run.

A well-designed computer network model base is shown through several examples. It will help the computer network system design to be done in a modular, hierarchical, coordinated fashion.
CHAPTER 1

INTRODUCTION

1.1 Model Base Concept

Based on the model base concepts developed by Zeigler [1], "A model is always a model of a particular entity . . . an entity is a 'key' for retrieving models and experimental frames." A methodology for an integrated, model-based system design is being developed by Rozenblit [2]. This methodology provides a design process supported by modelling and simulation techniques, as shown in Figure 1.

```
+-----------------+          +-----------------+          +-----------------+
| OBJECTIVES      |          | MODELLING       |          | EVALUATION       |
+-----------------+          +-----------------+          +-----------------+          +-----------------+          +-----------------+
| DESIGN          |          | SIMULATION      |          | BASE OF          |
| MODEL BASE      |          |                |          | EXPERIMENTAL     |
|                 |          |                |          | FRAMES          |
```

Figure 1. Design in the Multifaceted Modelling Context.
This methodology constitutes a basis for an intelligent system design environment. The underlying objectives shown in Figure 1 should be any interested subset (pruned substructure) of the system entity structure whose definition is given in Zeigler's book [3].

The embodied knowledge in entity structure—decomposition, taxonomy, and coupling—should be responsible for managing the relationship among the entities. These features help the user to design an interested subset of the whole system. This formalism, enabling users to uniquely specify models based on pruning from entity structure, is called a composition tree [2], as shown in Figure 2. It is a tree with atomic models and a coupled model which couples the atomic models together in a hierarchical form.

![Composition tree and hierarchically specified model.](image)

Figure 2. Composition tree and hierarchically specified model.
S is the system specification, C is the coupling scheme, and H is correspondence.

Applying this methodology to a simulation environment, we should construct a corresponding model for each entity. Then this environment should be able to retrieve the model from the model base, based on the pruned substructure, with the help of simulation studies to select the best design alternative. Based on this concept, an intelligent simulation environment can be constructed by having an: (1) entity structure representation, (2) entity structuring program (ESP), and (3) model base (as shown in Figure 3).

![Diagram of Intelligent Simulation Environment]

Figure 3. Intelligent simulation environment.

The entity structure representation can be used to organize and manage models and experimental frame. The entity structuring program (ESP) is an existing tool which allows interaction with the user to construct entity
structures [3]. The ESP-4 (Entity Structure and Pruner) improves the ESP, enabling designing, accessing, manipulating and pruning entity structures for large model base. This tool has been developed in the AI and Simulation Group of the Computer Engineering Research Laboratory, the University of Arizona. Zeigler and Sevinc [4] applied this tool in establishing an entity structure representation for local area networks. To make this system complete, a well-designed model base is required. This is the goal of the work reported here.

1.2 Application to Computer Network Design

A Local Area Network (LAN) is used to connect terminals, computers, printers and other auxiliary devices. Due to the increasing need for office automation (distributed information exchange and processing), a number of LANs need to be interconnected by a number of gateways. Therefore, the complexity of inter-network communication increases, and computer network performance evaluation becomes more important. A design error is inevitable if it is done without performance evaluation. Two major approaches for computer network design and performance evaluation are: analytic modelling, and discrete event simulation. In this work, we apply the discrete event simulation method and model base concept. The reasons we chose this approach are:
1. The different protocols, couplings and performance requirements of computer networks are unpredictable. Therefore the virtues of flexibility, adaptability and generality of the computer network's performance evaluation system is of utmost importance. Performance evaluation for different levels of abstraction is almost impossible without the implementation of a more user-friendly performance evaluation environment. This object can be achieved by designing a generalized simulator to simulate any coupling, any topology, and any protocol of computer communication system.

This fact has been introduced in the paper "A Generalized Simulator for Computer Networks" [5]. Chlamtac and Franta developed a generalized network simulator to help investigate the performance of computer networks at a variety of levels. The program is highly modular so that the user can change one level without affecting the others.

2. The discrete event simulation approach can easily expand the scope of our work to a knowledge-based simulation system. Based on the model base concepts described in the last section, to achieve the long-term goal (an intelligent simulation environment), we need to establish some model base design principles,
which should be able to work consistently with the other part of this environment. We seek to establish these principles with the help of the application in computer network simulation.

In the world of computer network systems, there are many entities such as the host computer, printer, terminal, interface unit and transmission medium. We implement these as atomic modules in the simulation language SIMSCRIPT II.5. The users can then generate any instances they need, couple these modules together, and call the simulator to obtain and display simulation results.
CHAPTER 2

MODEL BASE FRAMEWORK

2.1 Model Base Entry: I/O Port Description

Considering data base design, when data base administrators establish a data base, they need to be concerned with the different types (integer, text, date, memo) and formats (range) of the data.

In designing the model base, our concern is not based on each data unit, but based on each model unit. "To specify modular discrete event models, we must view a model as possessing input and output ports through which all interaction with the environment is mediated" [1]. Consider the following example, shown in Figure 4.

different instances :

```
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>++</td>
</tr>
<tr>
<td>model A(A_id) +-- in_port A_in_1 out_port A_out_1---&gt; in_port B_in_1 out_port B_out_1</td>
</tr>
<tr>
<td>------------</td>
</tr>
</tbody>
</table>
```

Pseudo code for A model

--- internal transition --
SEND: hold( process-time)
send X to A_out_1(A_id)
passivate

Pseudo code for B model

---- external transition --
when receive Y on B_in_1( B_id ) :

Figure 4. Modular discrete model.
In this example, a model base designer should design the following:

1. An output port for each send operation.
2. An input port for each receive operation.
3. An interface for the user to specify the coupling, which includes port coupling and ID (identification of each model instance) coupling.

2.2 I/O Port Communication

From the above, we can see the model base design has at least the last three differences from data base design. Especially when considering building up a model base for an intelligent simulation environment, we need to design our model base based on the hierarchy and coordinator concepts [1], as follows.

First consider three modules which need to communicate with one another, for example, the host computer, printer, and terminal. In simulation, the software should contain different ports to handle different channels' communication. One way to handle this is to create a pseudo-module, called a "coordinator," to couple these ports. Each level has a coordinator which controls messages between components at a lower level. A component cannot communicate with its brother directly; it must send a message to its coordinator, then the coordinator passes this message to its brother. In this way, we can keep each module independent.
For different simulation experiments, only the coordinator module needs to be modified. A general case is shown in Figure 5.

![Diagram](image)

Figure 5. Coordinator for a hierarchical system.

In this example, the destination of ports in the atomic module A, B, and C is initially unspecified. The output link field of each module structure is left unfilled. It is up to the coordinator to decide the coupling of the various links. The simulation system should be able to interactively ask the user to code the coordinator modules. If the user wishes to couple A module to B module, the level
0 coordinator is an interface of channel 1 and 2. If the user wishes to couple module A and module C, level 0 and level 1 coordinators are the interface to connect channel 1, 3, 4 and 5.

Based on the above example, a system model base should be divided into two parts:

1. Atomic modules, the simulation designer construct all of these modules, prepare necessary input and output ports for each possible future coupling.

2. Coordinator modules, which are responsible for coupling lower level modules together.

Once the simulation system is established, users can retrieve different level coordinators and modify the coordinator modules without considering lower level modules' design and implementation.

2.3 Couple Scheme Algorithm

In Section 2.2, we described the atomic modules by its input and output ports. In this section, the relation between each atomic module is described.

In a relational database management system, the relations between each data record are represented by relational tables. Not only to fulfill the similar function of relational tables in a data base system, but also to realize the modularity property, a coordinator model is necessary to
connect instances that are either atomic modules or lower-level coupled modules. The successive coupling of larger and larger components constitutes a hierarchical model base [1].

Consider the coupled model, CCA, shown in Figure 6. It is a set of instances of a class CA model. A CA model is a set of instances of a class A model. Let us image two real cases:

1. If A1 can be thought of as a set of CRT instances and A2 as a communication interface module, then CA1 represents a set of instances of a terminal room. Furthermore, if CA2 is considered as a set of transmission medium and CA3 as a set of host computer room, then the CCA coupled model represents a set of LAN instances.

2. If A1 can be thought of as a set of terminal room instances and A2 a set of host computer room instances, then CA1, CA2, and CA3 represent a set of instances of different LANs and gateway, and the CCA coordinates these together to form long-haul network instances.

This coupling is determined by coordinator modules, and the simulation environment must provide users an interface to modify these modules.
A coordinator module should couple points a, b and c to form a coupled model. There are three types of coupling within each class:

1. a: external input coupling,
2. b: internal coupling,
3. c: external output coupling,

where internal coupling is the coupling within the same level (for instance, A1 to A2, CA1 to CA2, CA2 to CA3); external input and external output coupling couple two successive levels (for instance, CCA to CA1, CA1 to A1, A2 to CA1, etc).

Figure 7 outlines how a coordinator module works. Where ID is an instance identification, each ID is a pointer.
Figure 7. Coordinator module algorithm.
that points to a model structure which has a field \( A_i \)-couple-j.

\( i \) and \( j \) are integers. \( i \) represents different instances of an A model, and \( j \) represents different output port connection of that particular instance.

\( X \) and \( Y \) are communication messages between two modules. \( X \) and \( Y \) have source and destination fields which store the instance identification ID.

To implement the coordinator modules, the macro-instruction generation features [8] of the SIMSCRIPT language are very useful. Further discussion will be given in Section 2.4.

2.4 File Organization and Entity Structure

As described in Section 1.1, every atomic model is an entity which can be pruned from the entity structure (from the user's view) or, say, from the model base (from the model base designer's view). Those atomic models can be coupled to form a coupled model by composition tree formalism.

But considering model base implementation, an atomic model cannot be a file unit in internal file structure because any atomic model has different I/O port descriptions, and what should be retrieved and coupled are those individual ports, not the atomic model.

And as we have already gone from Sections 2.1 to 2.3, the composition tree formalism can be implemented by an I/O
port coupling scheme (Figure 7). This suggests to us that we should make each input port of each atomic model an individual file.

Beside each file name, which corresponds to each input port of atomic model, the model base user should be informed of the coupling constraints, i.e., coupling scheme C in the composition tree (Figure 2). The primitive coupling constraints are represented by the I/O table, which has three attributes, Input port, Output port, and Couple for that particular output port. The model base users should not make any coupling which violates this I/O table information.

<table>
<thead>
<tr>
<th>Input Port</th>
<th>Output Port</th>
<th>Couple</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 8. I/O table.

Figure 9 is a pruned sub-entity structure for a computer network set up by Zeigler and Sevinc [4]. In Section 4.2, we will build up a model base for this substructure. The I/O tables and routine files corresponding to this substructure will be illustrated.
Figure 9. Pruned sub-entity structure.
CHAPTER 3

FORMAL MODEL DESCRIPTION

Based on the above model base framework, a computer network simulation model base has been established. The atomic models are Host Computer (HC), Packet Communication Unit (PCU), Token Interface Module (TIM), CRT terminal, printer, and transmission medium. These atomic modules can be coupled to form LANs with global bus or ring topologies. Further, these LANs can be interfaced via a Gateway.

Each atomic module has one or two buffers, a processor, and a fixed number of input and output ports (Figure 10).

As the above figure shows, each atomic module communicates with the other atomic modules through the input and output ports. The external events are carried by message packets, whose format is shown in Figure 11.
A modular system should have clear input and output port specification, and the internal design of each atomic module should be:

1. independent: any atomic module should be able to couple with any other atomic module if not colliding with realistic system constraints.

2. self-content: any atomic module can be simulated without coupling with other modules, only by adding an experimental frame.

The most different part between each atomic module is the processor design, if describing the processor as a finite state machine. The states are implemented as phase labels in the simulation language.

3.1 CRT Module Description

There are two modes to describe the CRT module. One is the "command mode," and the other is the "data mode." When in command mode, the CRT tries to establish a session with its destination computer by sending a request signal. The CRT does not enter the data mode until the "session
established" signal comes back from the network. When in
data mode, the CRT simulates the editing of a user's file on
the terminal. The processor of one CRT handles the keyboard
commands issued by the users.

1. Push the character into the out_buffer when the user
keys in a character which is not a backspace, a
delete key, or a line feed character.
2. Pop the character out from the out_buffer when the
user keys in a backspace or delete key.
3. Send the information on the out_buffer to the inter­
face device when the out_buffer is full or when the
user hits the line feed key.

The following is the pseudo code of CRT module.

CRT_module:
when receive pk(pk_id) from CRT_inl(crt_id)
    if cf(pk_id) = "ack_session"
        kybd_to_crt_ok(crt_id) := true
        send activate_kybd to KYBD
        passivate
    else
        hold(re_txmt time)
        kybd_to_crt_ok(crt_id) := false
        send activate_kybd to KYBD
        passivate
CRT module

+------------+-----------------+
| CRT in2    | CRT outl output |
| input port |                  |
+------------+-----------------+
|             |                  |
| RE_TXMT     |                  |
| +------------+-----------------+
| PROCESSOR_CRT|                  |
| +------------+-----------------+
| +-------------+-----------------+
| +-> Key Board|                  |
| +------------+-----------------+
| +-------------+-----------------+
| CRT in1 input port |

Figure 12. CRT module.
when receive pk(pk_id) from CRT.PROCESSOR(crt_id)
    send pk(pk_id) to CRT_outl(crt_id)
    passivate

**Key Board module:**

when receive activate_kybd from INITIALIZATION or CRT(crt_id)
if kybd_to_crt_ok(crt_id) = false
    ky_type(char_id) := "req"
    send char(char_id) to PROCESSOR_CRT(crt_id)
    passivate
else

**DATA.MODE:**

    ky_type(char_id) := "character" or "backspace" or "CR"
    send char(char_id) to PROCESSOR_CRT(crt_id)
    hold(ky_int time)
    go to DATA.MODE

**PROCESSOR CRT module:**

when receive char(char_id) from KYBD(crt_id)
if ky_type(char_id) = "req"
    cf(pk_id):="req", dest_addr(pk_id):="destination module"
    send pk(pk_id) to CRT(crt_id)
    passivate
if ky_type(char_id)= "character"
    insert(character, que_crt(crt_id))
else

    if ky_type(char_id)="backspace"
        char := last(que_crt(crt_id))
        que_crt(crt_id) := rest(que_crt(crt_id))
    else
        if ky_type(char_id)= "CR"
            cf(pk_id):="send_data", dest_addr(pk_id):=
            "destination module"
            send pk(pk_id) to CRT(crt_id)
            end if
    end if
end if

if full(que_crt(crt_id))
    clear que_crt(crt_id)
    cf(pk_id):= "send_data"
    dest_addr(pk_id):= "destination module"
    send pk(pk_id) to CRT(crt_id)
end if

if current time > logout time(crt_id)
    kybd_to_crt_ok(crt_id) := false
end if

3.2 Interface Module Description

An interface module (PCU in global bus system, TIM in token passing system) has an input buffer, a processor and a output buffer. These modules implement different protocols
by implementing different state transitions inside their processors. The interface module can be either transmitter or receiver.

In the CSMA (Carrier Sense Multiple Access) protocol, the processed packet must wait in the output buffer of the PCU until the transmission medium becomes available. In the Token passing protocol, the processed packet needs to wait in the output buffer of the TIM until this TIM has received the token.

The following is the pseudo code for the CSMA protocol which is implemented in the PCU module, and Token-Passing protocol, which is implemented in TIM module.

3.2.1 Packet Communication Unit

PCUT module:

when receive pk(pk_id)
    send pk(pk_id) to BUFFER_PCUT(pcut_id)
    passivate
when receive pk(pk_id) from OUTPUT_BUFFER_PCUT(pcut_id)
    send pk(pk_id) to PCUT_out1(pcut_id)
    passivate
when receive pk(pk_id) from BUFFER_PCUT(pcut_id)
    send pk(pk_id) to PCUT_out2(pcut_id)
    passivate
Packet Communication Unit module

PCUT_in2 input port   PCUT_out1 output port

| PROCESSOR_PCUT |
+------------------+
| Done             +

< BUFFER_PCUT <

PCUT_out2 output port   PCUT_in1 input port

| PROCESSOR_PCUR |
+------------------+
| Done             +

< BUFFER_PCUR <

PCUR_out2 output port   PCUR_in1 input port

PCUT: PCU Transmitter part  
PCUR: PCU Receiver part

<table>
<thead>
<tr>
<th>input</th>
<th>output</th>
<th>phase variable</th>
<th>data variable</th>
<th>parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>PCUT_in1</td>
<td>PCUT_out1</td>
<td>proc_pcut_work</td>
<td>tproc_free</td>
<td>proc_pcut_time</td>
</tr>
<tr>
<td>PCUT_in2</td>
<td>PCUT_out2</td>
<td>proc_pcut_work</td>
<td>que_pcut_full</td>
<td>num_que_pcut</td>
</tr>
<tr>
<td>PCUR_in1</td>
<td>PCUR_out1</td>
<td>proc_pcur_work</td>
<td>rproc_free</td>
<td>proc_pcur_time</td>
</tr>
<tr>
<td>PCUR_in2</td>
<td>PCUR_out2</td>
<td></td>
<td>que_pcur_full</td>
<td>num_que_pcur</td>
</tr>
</tbody>
</table>

Figure 13. PCU module.
**BUFFER PCUT module:**

when receive pk(pk_id) from PCUT(pcut_id)

if empty(que_pcut(pcut_id))

& processor_pcut_free(pcut_id) = true

send pk(pk_id) to PROCESSOR_PCUT(pcut_id)

else

insert(pk_id, que_pcut(pcut_id))

end if

if full(que_pcut(pcut_id))

cf(pk_id) := que_full

send pk(pk_id) to PCUT(pcut_id)

end if

passivate

when receive done from PROCESSOR_PCUT(pcut_id)

processor_pcut_free(pcut_id) := true

if full(que_pcut(pcut_id))

cf(pk_id) := que_not_full

send pk(pk_id) to PCUT(pcut_id)

end if

if not empty(que_pcut(pcut_id))

pk_id = first(que_pcut(pcut_id))

que_pcut(pcut_id) := rest(que_pcut(pcut_id))

send pk(pk_id) to PROCESSOR_PCUT(pcut_id)

end if

passivate
PROCESSOR_PCUT module:

    when receive pk(pk_id) from BUFFER_PCUT(pcut_id)
        go to TPROC.WORK

TPROC.WORK: processor_pcut_free(pcut_id) := false

    hold(pcut_process_time)
    send pk(pk_id) to OUTPUT_BUFFER_PCUT(pcut_id)
    send done to BUFFER_PCUT(pcut_id)
    go to TPROC.WORK

OUTPUT_BUFFER_PCUT module:

    when receive pk(pk_id) from PROCESSOR_PCUT(pcut_id)
        if cf(pk_id) = "req" or "send_start"
            send pk(pk_id) to PCUT(pcut_id)
            passivate
        if cf(pk_id) = "send_data"
            insert(pk_id, que_pcut_out_bf(pcut_id))
            passivate
    when receive pk(pk_id) from PCUT(pcut_id)
        if cf(pk_id) = "ack_data"
        & not empty (que_pcut_out_bf(pcut_id))
            pk_id := first(que_pcut_out_bf(pcut_id))
            que_pcut_out_bf(pcut_id)=rest(qque_pcut_out_bf(pcut_id))
            send pk(pk_id) to PCUT(pcut_id)
            passivate
        else
            if cf(pk_id) = "not_ack_data"
send pk(pk_id) to PCUT(pcut_id) --re_transmission
passivate

PCUR module:
when receive pk(pk_id)
    send pk(pk_id) to BUFFER_PCUR(pcur_id)
    passivate
when receive pk(pk_id) from PROCESSOR_PCUR(pcur_id)
    send pk(pk_id) to PCUR_out1(pcur_id)
    passivate
when receive pk(pk_id) from PROCESSOR_PCUR(pcur_id)
or BUFFER_PCUR(pcur_id)
    send pk(pk_id) to PCUR_out2(pcur_id)
    passivate

BUFFER_PCUR module:
when receive pk(pk_id) from PCUR(pcur_id)
    if empty(que_pcur(pcur_id))
    & processor_pcur_free(pcur_id)=true
        send pk(pk_id) to PROCESSOR_PCUR(pcur_id)
    else
        insert(pk_id, que_pcur(pcur_id))
always
if full(que_pcur(pcur_id))
    cf(pk(pk_id)) := que_full
    send pk(pk_id) to PCUR(pcur_id)
    passivate
when receive done from PROCESSOR_PCUR(pcur_id)

    processor_pcur_free(pcur_id) := true

    if full(que_pcur(pcur_id))
        cf(pk_id) := que_not_full
        send pk(pk_id) to PCUR(pcur_id)
    end if

    if not empty(que_pcur(pcur_id))
        pk_id := first (pk_id, que_pcur(pcur_id))
        que_pcur(pcur_id) := rest(que_pcur(pcur_id))
        send pk(pk_id) to PROCESSOR_PCUR(pcur_id)
    end if

PROCESSOR_PCUR module:
when receive pk(pk_id) from BUFFER_PCUR(pcur_id)

    go to RPROC.WORK

RPROC.WORK:

    processor_pcur_free(pcur_id) := false
    hold(processor_pcur_work time)

    if cf(pk(pk_id)) = "req"
    & there are sessions available
        cf(pk(pk_id)) := "ack_session"
    else
        --cf(pk_id) = "send start" or "send data"
        send pk(pk_id) to PCUR(pcur_id)
        cf(pk_id) := "ack_data"
    end if

    dest_addr(pk_id) := pcut
send pk(pk_id) to PCUR(pcur_id)
send done to BUFFER_PCUR(pcur_id)
passivate

3.2.2 Token Interface Module

The input-output port design of the TIM (Token Interface Module) is the same as the PCU module (Figure 13), but the different protocol is represented by a different processor module. The pseudo code for TIM processor, which is based on a simplified MAC finite state machine diagram, is shown below.

```
+---+----------------+----------------+----------------+-
| model| phase variable| data variable | parameters |
+---+----------------+----------------+----------------+-
| TIM | data_transfer  | num_que_tim    | data_txmt_time |
|     | token_pass     | timproc_free   | data_receive_time|
|     |                |                | tok_txmt_time   |
|     |                |                | tok_accept_time |
+---+----------------+----------------+----------------+-
```

Figure 14. TIM module.

PROCESSOR_TIM module

when receive pk(pk_id)
  if type(pk_id) = "data"
    and tim_txmt_ok(timt_id) = true
    tim_proc_free(tim_id) = false
USE_TOKEN: hold(tim_proc_time)
  tim_proc_free(tim_id) = true
send done to BUFFER_TIM(tim_id)
type (pk_id) = "data"
send pk(pk_id) to TIM_outl(tim_id)
passivate
if type(pk_id) = "token"
    tim_txmt_ok(tim_id) := true
USE_TOKEN: hold(token_hold_time)
    tim_txmt_ok(tim_id) := false
PASS_TOKEN: hold(tok_pass_time)
type (pk_id) = "token"
    send pk(pk_id) to next_station(tim_id)
passivate

3.2.3 Gateway Module

Gateway is a network interface module which improves interoperability between different networks. Gateway must know multiple protocols, one for each network [15]. Consider the interconnection between network A and B, shown in Figure 15. The gateway model can be represented as a host computer providing processor A and B to execute its major functions, which include message buffering, address mapping, and protocol transformations.

GATEWAY module:

    when receive pk(pk_id) from LAN_CSMA_outl
        if cf(pk_id) = "data"
            send pk(pk_id) to BUFFER_A
        passivate
else
    if cf(pk_id) = "transmit_ok"
        transmit_to_bus_A_ok := true
    else
        transmit_to_bus_A_ok := false
        passivate
when receive pk(pk_id) from BUFFER_A
    send pk(pk_id) to LAN_CSMA_in2
    passivate
when receive pk(pk_id) from PROCESSOR_A
    send pk(pk_id) to LAN_TOK_in2
    passivate
when receive pk(pk_id) from LAN_TOK_out1
    if cf(pk_id) = "data"
        send pk(pk_id) to BUFFER_B
        passivate
    else
        if cf(pk_id) = "transmit_ok"
transmit_to_bus_B_ok := true
else
transmit_to_bus_B_ok := false
passivate
when receive pk(pk_id) from BUFFER_B
send pk(pk_id) to LAN_TOK_in2
passivate
when receive pk(pk_id) from PROCESSOR_B
send pk(pk_id) to LAN_CSMA_in2
passivate

3.3 Host Computer Module

The host computer contains an input buffer and a processor. When the buffer receives a packet from another module, it checks if the buffer is empty and the processor is free. If it is, then it sends the packet to the processor; otherwise, the packet must wait in the buffer until the buffer receives a "done" signal from the processor. The pseudo code for host computer module is:

HOST COMPUTER module:

    when receive pk(pk_id)
        send pk(pk_id) to BUFFER_HC(hc_id)
        passivate
    when receive pk(pk_id) from PROCESSOR_HC(hc_id)
        send pk(pk_id) to HC_outl(hc_id)
        passivate
Host Computer module

---

HC_out1 output port

<table>
<thead>
<tr>
<th>PROCESSOR_HC</th>
<th>done</th>
</tr>
</thead>
<tbody>
<tr>
<td>BUFFER_HC</td>
<td></td>
</tr>
</tbody>
</table>

HC_out2 output port

<table>
<thead>
<tr>
<th>HC_in1</th>
<th>HC_out1</th>
<th>proc_hc_work</th>
<th>hproc_free</th>
<th>num_que_hc</th>
<th>que_hc_full</th>
<th>proc_hc_time</th>
</tr>
</thead>
<tbody>
<tr>
<td>HC_in2</td>
<td>HC_out2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

when receive pk(pk_id) from BUFFER_HC(hc_id)

send pk(pk_id) to HC_out2(hc_id)

passivate

BUFFER_HC module:

when receive pk(pk_id) from HC(hc_id)

if empty(que_hc(hc_id)) & processor_hc_free = true

send pk(pk_id) to PROCESSOR_HC(hc_id)
else
    insert(pk_id, que_hc(hc_id))
end if
if full(que_hc(hc_id))
    cf(pk_id) := "que_full"
    send pk(pk_id) to HC(hc_id)
    passivate
when receive done from PROCESSOR_HC(hc_id)
    processor_hc_free := true
if full(que_hc(hc_id))
    cf(pk_id) := que_not_full
    send pk(pk_id) to HC(hc_id)
end if
if not empty(que_hc(hc_id))
    pk_id := first(que_hc(hc_id))
    que_hc(hc_id) := rest(que_hc(hc_id))
    send pk(pk_id) to PROCESSOR_HC(hc_id)
    passivate
PROCESSOR_HC module:
    when receive pk(pk_id) from BUFFER_HC(hc_id)
        go to HPROC.WORK
HPROC.WORK:
    processor_hc_free := false
    hold(processor_hc_work time)
    send pk(pk_id) to HC(hc_id)
send done to BUFFER_HC(hc_id)

passivate

3.4 Printer Module

The printer module is modelled by a producer and a consumer. The producer accepts the packet and then puts it into the printer buffer; the consumer takes the packet from the printer buffer and outputs this packet according to the printer's speed.

Printer module

```
+----------------------------------+
|                                 |
|                                 |
|                                 |
|                                 |
|                                 |
|                                 |
|                                 |
|                                 |
|                                 |
|                                 |
|                                 |
|                                 |
|                                 |
+----------------------------------+

PTR_outl output port

PTR_inl input port

input port | output port | phase variable | data variable | parameters |
-----------|-------------|----------------|---------------|------------|
PTR_inl    | PTR_outl    | consumer_work  | num_que_printer | consum_time |
```

Figure 17. Printer module.
PRINTER module:
when receive pk(pk_id)
    send pk(pk_id) to RECEIVER_PTR(ptr_id)
    passivate

RECEIVER_PTR module:
when receive pk(pk_id) from PTR(ptr_id)
    inser(pk_id, que_ptr(ptr_id))
    if one(que_ptr(ptr_id))
        send pk(pk_id) to CONSUMER_PTR(ptr_id)
        passivate

CONSUMER_PTR module:
when receive pk(pk_id) from RECEIVER_PTR(ptr_id)
    go to CON.WORK

CON.WORK:
    hold(consumer_ptr_work time)
    if not empty(que_ptr(ptr_id))
        pk_id := first(que_ptr(ptr_id))
        que_ptr(ptr_id) := rest(que_ptr(ptr_id))
        send pk(pk_id) to paper
    else
        passivate
    end if
    go to CON.WORK
3.5 Transmission Medium Module

The transmission module contains an infinite buffer. A packet inserted in this buffer waits a transmission delay time, then goes to its destination.

Transmission medium module

```
| input port | list | output port |
+------------+------+-------------|
| BUS_inl    |      | BUS_outl    |
```

Figure 18. Bus module.

The following is the pseudo code for the bus module:

```pseudo
def when receive pk(pk_id):
    insert(pk_id, list_bus)
    go to DELAY

    def DELAY:
        hold(bus_delay time)
        pk_id := first(list_bus)
        list_bus := rest(list_bus)
        send pk(pk_id) to BUS_outl
        passivate
```

```
CHAPTER 4

SIMSCRIPT IMPLEMENTATION

A description of the implementation of the model base is given below.

4.1 Implementation Facilities

The simulation program is written using the SIMSCRIPT II.5 programming language [8] running on a VAX-11/750 computer. In SIMSCRIPT II.5, we use a process as our primary dynamic object. There are many different processes in a model, and there may be many instances (or copies) of a process in a simulation.

In a modular coordinated system, we represent each input port of atomic models as a SIMSCRIPT routine. When a routine is accessed, it may pass through a number of different process states, then send the result to an output port, which may access another routine.

4.2 Output and Input Ports

As shown in Figure 6 (Section 2.3), each module can be coupled to its brothers (internal coupling), to child modules (external input coupling), or to parent modules (external output coupling).
These couplings are done by the user through the coordinator modules (Figure 7). The model base designer must predict the possible input-output ports which the user may couple, and reserve these ports for the coordinator module. This is the major design consideration of each atomic module or coupled module. Let us demonstrate the model base concept by the following examples.

Example 1:
Consider the following figure (Figure 19). The CRT may couple to PCU (in CSMA protocol), or TIM (in Token-passing protocol), or PC (Personal Computer).

![Figure 19. Internal coupling. — (a) CRT-PCU coupling; (b) CRT-TIM coupling; (c) CRT-PC coupling.](image)

That means, CRT.out port may call PCU.in routine (i.e., CRT output ports are coupled to PCU input ports, Figure 19a), or call TIM.in (Figure 19b) or PC.in (Figure 19c). So the CRT output port can be coupled to three possible input ports of the other atomic modules; the CRT module must provide three attributes for that particular output port. (CRT-couple-1, CRT-couple-2 and CRT-couple-3 in the following, Figure 20.)
/* preamble file of SIMSCRIPT II.5 */
temporary entities
every CRT has a id_CRT, a KeyBoard_to_CRT_ok, a que_full,
a CRT_coupl e_1, a CRT_coupl e_2, a CRT_coupl e_3
owns a que_CRT

Figure 20. Data structure for CRT model.

Note that the three CRT_coupl e attributes (i.e., output ports) for a CRT instance are provided, and each input-output port coupling pair can be any instance pairs. For instance, "CRT.out of instance CRT-1, or instance CRT-2 or . . ." can be coupled to "PCU.in of instance PCU-1, or instance PCU-2, or instance TIM-1 or. . . ." So the number of possible coupling pairs is infinite, and we should make these coupling pairs easily modified by the user when the model base is established.

Right now, we should be able to implement this example according to the coordinator module algorithm (Figure 7). Suppose the A module in Figure 7 is a CRT module. Then we can rewrite the A.in routine, which belongs to the atomic module in Figure 7 as follows:

```plaintext
CRT.in( CRT_id, packet )
.......... packet := process( packet )
connect CRT.out
```

Figure 21. CRT input port.
Rewrite the A.out routine, which belongs to the coordinator module in Figure 7, as follows:

```
+-----------------------------------------------+
| CRT.out( CRT_id, packet)                     |
| source( packet ) = CRT_id                    |
| destination( packet ) = CRT_couple( CRT_id ) |
| if destination( packet ) belongs to { same level module : |
| PCU_id, TIM_id or PC_id }                    |
| /********************************************|
| connect PCU.in( CRT_couple_1( CRT_id )== PCU_id ) |
| or connect TIM.in( CRT_couple_2( CRT_id )== TIM_id ) |
| or connect PC.in( CRT_couple_3( CRT_id )== PC_id ) |
+-----------------------------------------------+
```

Figure 22. Coordinator for CRT module.

The last three lines in Figure 22 should be implemented in a modifiable form. In SIMSCRIPT II.5 we can utilize the macro features ("DEFINE TO MEAN" or "SUBSTITUTE" statement). This will become more clear in future examples.

**Example 2:**

Suppose the model base provided the atomic models Generator, HC, and Transducer. Each atomic model has its input and output port description, as in the following figure.
If the model base users need to do the following coupling (Figure 24), they can implement these port couplings according to the coordinator model described in Section 2.3. Implementation is shown in Figure 25.
COORDINATOR implementation:

```c
/* port instance coupling */
routine HC.gen( hcs_id )
for i=1 to num_HC_in_HCS( hcs_id )
do
  if i = 3
    let hc_couple2( inst_hc(i) ) = inst_Transducer(1)
    /* external output coupling */
    let hc_couple1( inst_hc(i) ) = NULL
  else
    let hc_couple2( inst_hc(i) ) = NULL
    let hc_couple1( inst_hc(i) ) = inst_hc( i+1 )
    /* internal coupling */
  always
loop

/* HCS.pre file */
substitute this line for Generator.outl
HC.inl( inst_hc(1),pk_id) /* external input coupling */

substitute this line for HC.out2
Transducer.inl( hc_couple2(hc_id),pk_id )
    /* external output coupling */

/* HC.pre file, internal coupling */
substitute this line for HC.out1
HC.inl( hc_couple1( hc_id ), pk_id)
```

Figure 25. Directed-graph coupling coordinator implementation.

The complete SIMSCRIPT II.5 code of this example is shown in Appendix B.

**Example 3:**

In this example, we have the same model base as Example 2, but users need to do a different coupling (Figure 26). Then the coordinator implementation should be different from Example 2; this is shown in Figure 27.
Figure 26. Broadcast coupling.
Figure 27. Broadcast coupling coordinator implementation.
The complete SIMSCRIPT II.5 code of this example is shown in Appendix C.

4.3 Organization of Model Base

In Section 2.4, we described each input port corresponding to a file in the model base, and the I/O tables which provide information on coupling constraints. We will now illustrates these concepts in greater detail.

4.3.1 Files for Each Model

Refer to the complete SIMSCRIPT II.5 code of Example 2 (Appendix B) of the last section, which shows that the whole program consists of four kinds of files:

1. PREAMBLE files: these include EX2.pre, HCS.pre, HC.pre, Generator.pre, Trasducer.pre, and PK.pre files. These files are declaration files. We also implemented part of the coordinator in the preamble file, as Figure 25 shows.

2. GENERATOR files: these include EX2.gen, HCS.gen, and HC.gen files. Generator files create different instances for each model and couple the input-output port pairs. This is part of the coordinator implementation.

3. ROUTINE files: these include the HC.rtn file. This file represents one input port of the atomic model
HC. It provides three output ports, HCPROC.out1, HCPROC.out2, and HCPROC.out3, for possible coupling.

4. EXPERIMENT file: input parameters, generate input variables, and receive output variables for this particular experiment.

The model base designer should provide all the ROUTINE files which provide all possible input and output ports information and part of the PREAMBLE files. Then the model base users can design their particular experiment through the GENERATOR, EXPERIMENT and part of the PREAMBLE file.

Therefore, the well-designed model base should consist of three kinds of information:

1. Input port information: Each file (the ROUTINE files) should be named by the name of each input port of each model (for instance, HCBF.inl in the above example).

2. Output port information: Each file needs to provide all possible output ports for coupling (for instance, HCPROC.out1, 2, 3 in the above example).

3. Coupling constraints: possible input ports to be coupled for one particular output port.

Figure 28 represents this information for the above example in an I/O table.
Figure 28. Part of the HC I/O table.

But when the model base includes the models PC, TIM or Printer, then HC may couple to these models. The complete HC I/O table for this model base should be:

<table>
<thead>
<tr>
<th>input port</th>
<th>output port</th>
<th>couple</th>
</tr>
</thead>
<tbody>
<tr>
<td>HC.inl</td>
<td>HC.out1</td>
<td>Transducer.inl</td>
</tr>
<tr>
<td>HC.inl</td>
<td>HC.out2</td>
<td>HC.inl</td>
</tr>
</tbody>
</table>

Figure 29. The complete HC I/O table.

In Appendix A we list all the I/O tables for each model in the subdomain of the computer network.

4.3.2 How to Build Coupled Model

Based on these I/O tables, we can easily decide what files should be retrieved to do a particular experiment. In the following three examples we show the possible network coupling, then show how to utilize the join function in
relational algebra to decide which of the files needs to be retrieved from the model base.

In these examples, "model A where (x,y)" represents "from the I/O table of model A, select the tuple whose input port attribute value is A.inx and output port attribute value is A.outy".

**Example 1:**

```
+-----------+ +-----------+ +-----------+ +-----------+ +-----------+
| Generator  |  CRT     |  PC       |  PTR      |  Transducer|
+-----------+ +-----------+ +-----------+ +-----------+ +-----------+
```

Figure 30. Coupled model: workstation.

```
(((CRT where (1,1)) join (PC where (1,1))):
```

<table>
<thead>
<tr>
<th>CRT_in</th>
<th>CRT_out</th>
<th>CRT_couple==PC_in</th>
<th>PC_out</th>
<th>PC_couple</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRT.inl</td>
<td>CRT.outl</td>
<td>PC.inl</td>
<td>PC.outl</td>
<td>PC.inl</td>
</tr>
<tr>
<td>CRT.inl</td>
<td>CRT.outl</td>
<td>PC.inl</td>
<td>PC.outl</td>
<td>PTR.inl</td>
</tr>
<tr>
<td>CRT.inl</td>
<td>CRT.outl</td>
<td>PC.inl</td>
<td>PC.outl</td>
<td>PCUT.inl</td>
</tr>
<tr>
<td>CRT.inl</td>
<td>CRT.outl</td>
<td>PC.inl</td>
<td>PC.outl</td>
<td>TIMT.inl</td>
</tr>
</tbody>
</table>

Figure 31. Result of CRT joined to PC.

```
(((CRT where (1,1)) join (PC where (1,1)) join (PTR where (1,1)):
```

<table>
<thead>
<tr>
<th>CRT_in</th>
<th>CRT_out</th>
<th>CRT_couple==PC_in</th>
<th>PC_out</th>
<th>PC_couple</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRT.inl</td>
<td>CRT.outl</td>
<td>PC.inl</td>
<td>PC.outl</td>
<td>PTR.inl</td>
</tr>
<tr>
<td>CRT.inl</td>
<td>CRT.outl</td>
<td>PC.inl</td>
<td>PC.outl</td>
<td>PCUT.inl</td>
</tr>
<tr>
<td>CRT.inl</td>
<td>CRT.outl</td>
<td>PC.inl</td>
<td>PC.outl</td>
<td>TIMT.inl</td>
</tr>
</tbody>
</table>
From the last table after joins, choose the ".in" attribute values as the file names we should retrieve from the model base. Based on these files (CRT.inl, PC.inl and PTR.inl), together with the coordinator files (PREAMBLE and GENERATOR file) and experiment file, we accomplish this simulation experiment design.

Example 2:

Figure 32. Result of CRT join PC join PTR.

Figure 33. Coupled model: LAN.
define view TermRm as
(CRT where (1,1))
   join   ((PCUT where (1,1))
   join   ((TermRm where (1,1))
define view HCRm as
   ((HCRm where (1,1))
   join   (PCUR where (1,1))
   join   (HC where (1,1))
   join   (PCUT where (1,1))
)
define view PTRRm as
   ((PTRRm where (1,1))
   join   (PCUR where (1,1))
   join   (PTR where (1,1))
)
TermRm
join (BUSCSMA where (1,1))
join HCRm
join PTRRm

After these operations, we will get the following files:
CRT.inl, PCUT.inl, TermRm.inl, BUSCSMA.inl, HCRm.inl,
PCUR.inl, HC.inl, PTRRm.inl, PTR.inl.
Example 3:

Figure 34. Coupled model: Internet.

(LAN_CSMA where (1,1) or (2,2))
join (GW where (*,*)

Figure 35. Result of LAN_CSMA join Gateway.

(LAN_TOK where (1,1) or (2,2))
join (GW where (*,*)

Figure 36. Result of LAN_TOK join Gateway.
From the above operation, we know we need to retrieve these files: GWC.inl, GWT.inl, LAN_TOK.in2, and LAN_CSMA.in2.

4.4 Examples

Experiments were performed in which the behavior of the Local Area Network (CSMA and Token_Passing protocol) was studied in terms of average turnaround time and throughput. The following simulation results (Figure 38) are based on the fixed parameters shown in Figure 37. In future research, the random number generator should be designed to generate input parameter distributions.

<table>
<thead>
<tr>
<th>CRT number/ TermRm</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>average turn around time</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CSMA</td>
<td>44.90</td>
<td>46.57</td>
<td>47.66</td>
<td>49.14</td>
</tr>
<tr>
<td>Token_Passing</td>
<td>46.97</td>
<td>48.23</td>
<td>48.56</td>
<td>49.23</td>
</tr>
<tr>
<td>average throughput</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CSMA</td>
<td>4.32</td>
<td>3.83</td>
<td>3.53</td>
<td>3.12</td>
</tr>
<tr>
<td>Token_Passing</td>
<td>4.22</td>
<td>3.49</td>
<td>3.16</td>
<td>3.02</td>
</tr>
</tbody>
</table>

Figure 38. Simulation results, 3 Terminal room, 1 Host computer room.
CHAPTER 5

SUMMARY

In this thesis we try to implement the model base concept in the SIMSCRIPT II.5 simulation language. We clarify the algorithm to couple atomic model through the I/O port, which will enforce the model base based system design to be in a modular coordinated form. This will enhance the design process of an intelligent simulation system. Using the model base concept, we can easily build a computer network simulation system. The user of this simulation system should be able to do experiments at any level under the system constraints and to design the network with quick simulation evaluation feedback.

The simulation language that we chose in this work, SIMSCRIPT II.5, is a powerful, English-like, general purpose simulation language. Once the model pseudo code is decided, the SIMSCRIPT coding is very straightforward. The most important advantage of this language is that it provides many statistical tools to generate, measure, and display the random phenomena.

For the long-term goal, such as an intelligent simulation system, the simulation programming language not only has to implement a model base concept easily, but also has to
have the compatibility between object-oriented programming paradigms and a discrete event world view [14]. DEVS-SCHEME is a non-procedural language which includes most of the features of DEVS algorithm [3]. Future research should compare the DEVS-SCHEME and the SIMSCRIPT approach discussed here.
APPENDIX A

SUB DOMAIN OF COMPUTER NETWORK
SUB DOMAIN OF COMPUTER NETWORK

Atomic model:

CRT

<table>
<thead>
<tr>
<th>input port</th>
<th>output port</th>
<th>couple</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRT.in1</td>
<td>CRT.out1</td>
<td>PCUT.inl</td>
</tr>
<tr>
<td>CRT.in1</td>
<td>CRT.out1</td>
<td>TIMT.inl</td>
</tr>
<tr>
<td>CRT.in1</td>
<td>CRT.out1</td>
<td>PC.inl</td>
</tr>
<tr>
<td>CRT.in2</td>
<td>CRT.out1</td>
<td>PCUT.inl</td>
</tr>
<tr>
<td>CRT.in2</td>
<td>CRT.out1</td>
<td>TIMT.inl</td>
</tr>
</tbody>
</table>

PCU

<table>
<thead>
<tr>
<th>input port</th>
<th>output port</th>
<th>couple</th>
</tr>
</thead>
<tbody>
<tr>
<td>PCUT.in1</td>
<td>PCUT.out1</td>
<td>BUSCSMA.inl</td>
</tr>
<tr>
<td>PCUT.in1</td>
<td>PCUT.out2</td>
<td>CRT.in2</td>
</tr>
<tr>
<td>PCUT.in2</td>
<td>PCUT.out1</td>
<td>BUSCSMA.inl</td>
</tr>
<tr>
<td>PCUR.in1</td>
<td>PCUR.out1</td>
<td>HC.inl</td>
</tr>
<tr>
<td>PCUR.in1</td>
<td>PCUR.out1</td>
<td>PTR.inl</td>
</tr>
<tr>
<td>PCUR.in1</td>
<td>PCUR.out2</td>
<td>BUSCSMA.inl</td>
</tr>
</tbody>
</table>
### TIM

<table>
<thead>
<tr>
<th>input port</th>
<th>output port</th>
<th>couple</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIMT.inl</td>
<td>TIMT.out1</td>
<td>BUSTOK.inl</td>
</tr>
<tr>
<td>TIMT.inl</td>
<td>TIMT.out2</td>
<td>CRT.in2</td>
</tr>
<tr>
<td>TIMR.inl</td>
<td>TIMR.out1</td>
<td>HC.inl</td>
</tr>
<tr>
<td>TIMR.inl</td>
<td>TIMR.out1</td>
<td>PTR.inl</td>
</tr>
<tr>
<td>TIMR.inl</td>
<td>TIMR.out2</td>
<td>BUSTOK.inl</td>
</tr>
</tbody>
</table>

### GATEWAY

<table>
<thead>
<tr>
<th>input port</th>
<th>output port</th>
<th>couple</th>
</tr>
</thead>
<tbody>
<tr>
<td>GWC.inl</td>
<td>GWC.out1</td>
<td>LAN_TOK.in2</td>
</tr>
<tr>
<td>GWC.inl</td>
<td>GWC.out1</td>
<td>LAN_CSMA.in2</td>
</tr>
<tr>
<td>GWC.inl</td>
<td>GWC.out2</td>
<td>LAN_CSMA.in2</td>
</tr>
<tr>
<td>GWT.inl</td>
<td>GWT.out1</td>
<td>LAN_CSMA.in2</td>
</tr>
<tr>
<td>GWT.inl</td>
<td>GWT.out1</td>
<td>LAN_TOK.in2</td>
</tr>
<tr>
<td>GWT.inl</td>
<td>GWT.out2</td>
<td>LAN_TOK.in2</td>
</tr>
</tbody>
</table>

### BUSCSMA

<table>
<thead>
<tr>
<th>input port</th>
<th>output port</th>
<th>couple</th>
</tr>
</thead>
<tbody>
<tr>
<td>BUSCSMA.inl</td>
<td>BUSCSMA.out1</td>
<td>PCUR.inl</td>
</tr>
<tr>
<td>BUSCSMA.inl</td>
<td>BUSCSMA.out1</td>
<td>TIMR.inl</td>
</tr>
<tr>
<td>BUSCSMA.inl</td>
<td>BUSCSMA.out1</td>
<td>LAN_CSMA.inl</td>
</tr>
</tbody>
</table>

### BUSTOK

<table>
<thead>
<tr>
<th>input port</th>
<th>output port</th>
<th>couple</th>
</tr>
</thead>
<tbody>
<tr>
<td>BUSTOK.inl</td>
<td>BUSTOK.out1</td>
<td>PCUR.inl</td>
</tr>
<tr>
<td>BUSTOK.inl</td>
<td>BUSTOK.out1</td>
<td>TIMR.inl</td>
</tr>
<tr>
<td>BUSTOK.inl</td>
<td>BUSTOK.out1</td>
<td>LAN_TOK.inl</td>
</tr>
<tr>
<td>input port</td>
<td>output port</td>
<td>couple</td>
</tr>
<tr>
<td>---------------------</td>
<td>----------------------</td>
<td>--------------------</td>
</tr>
<tr>
<td>HC.inl</td>
<td>HC.out1</td>
<td>PTR.inl</td>
</tr>
<tr>
<td>HC.inl</td>
<td>HC.out1</td>
<td>PCUT.inl</td>
</tr>
<tr>
<td>HC.inl</td>
<td>HC.out1</td>
<td>TIMT.inl</td>
</tr>
<tr>
<td>HC.inl</td>
<td>HC.out2</td>
<td>PCUR.in2</td>
</tr>
<tr>
<td>HC.inl</td>
<td>HC.out2</td>
<td>TIMR.in2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>input port</th>
<th>output port</th>
<th>couple</th>
</tr>
</thead>
<tbody>
<tr>
<td>PC.inl</td>
<td>PC.out1</td>
<td>PTR.inl</td>
</tr>
<tr>
<td>PC.inl</td>
<td>PC.out1</td>
<td>PCUT.inl</td>
</tr>
<tr>
<td>PC.inl</td>
<td>PC.out1</td>
<td>TIMT.inl</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>input port</th>
<th>output port</th>
<th>couple</th>
</tr>
</thead>
<tbody>
<tr>
<td>PTR.inl</td>
<td>PTR.out1</td>
<td>Print message</td>
</tr>
<tr>
<td>PTR.inl</td>
<td>PTR.out2</td>
<td>PCUR.in2</td>
</tr>
<tr>
<td>PTR.inl</td>
<td>PTR.out2</td>
<td>TIMR.in2</td>
</tr>
</tbody>
</table>

Coupled model:

**TermRm**

<table>
<thead>
<tr>
<th>input port</th>
<th>output port</th>
<th>couple</th>
</tr>
</thead>
<tbody>
<tr>
<td>TermRm.inl</td>
<td>TermRm.out1</td>
<td>CRT.inl</td>
</tr>
<tr>
<td>TermRm.in2</td>
<td>TermRm.out2</td>
<td>BUSCSMA.in1</td>
</tr>
<tr>
<td>TermRm.in2</td>
<td>TermRm.out2</td>
<td>BUSTOK.in1</td>
</tr>
</tbody>
</table>
### HCRm

<table>
<thead>
<tr>
<th>input port</th>
<th>output port</th>
<th>couple</th>
</tr>
</thead>
<tbody>
<tr>
<td>HCRm.in1</td>
<td>HCRm.out1</td>
<td>PCUR.in1</td>
</tr>
<tr>
<td>HCRm.in1</td>
<td>HCRm.out1</td>
<td>TIMR.in1</td>
</tr>
<tr>
<td>HCRm.in2</td>
<td>HCRm.out2</td>
<td>BUSCSMA.in1</td>
</tr>
<tr>
<td>HCRm.in2</td>
<td>HCRm.out2</td>
<td>BUSTOK.in1</td>
</tr>
</tbody>
</table>

### PTRRm

<table>
<thead>
<tr>
<th>input port</th>
<th>output port</th>
<th>couple</th>
</tr>
</thead>
<tbody>
<tr>
<td>PTRRm.in1</td>
<td>PTRRm.out1</td>
<td>PCUR.in1</td>
</tr>
<tr>
<td>PTRRm.in1</td>
<td>PTRRm.out1</td>
<td>TIMR.in1</td>
</tr>
<tr>
<td>PTRRm.in2</td>
<td>PTRRm.out2</td>
<td>BUSCSMA.in1</td>
</tr>
<tr>
<td>PTRRm.in2</td>
<td>PTRRm.out2</td>
<td>BUSTOK.in1</td>
</tr>
</tbody>
</table>

### LAN_CSMA

<table>
<thead>
<tr>
<th>input port</th>
<th>output port</th>
<th>couple</th>
</tr>
</thead>
<tbody>
<tr>
<td>LAN_CSMA.in1</td>
<td>LAN_CSMA.out1</td>
<td>GWC.in1</td>
</tr>
<tr>
<td>LAN_CSMA.in2</td>
<td>LAN_CSMA.out2</td>
<td>BUSCSMA.in1</td>
</tr>
</tbody>
</table>

### LAN_TOK

<table>
<thead>
<tr>
<th>input port</th>
<th>output port</th>
<th>couple</th>
</tr>
</thead>
<tbody>
<tr>
<td>LAN_TOK.in1</td>
<td>LAN_TOK.out1</td>
<td>GWT.in1</td>
</tr>
<tr>
<td>LAN_TOK.in2</td>
<td>LAN_TOK.out2</td>
<td>BUSTOK.in1</td>
</tr>
</tbody>
</table>
APPENDIX B

SIMSCRIPT CODE FOR DIRECTED-GRAPH COUPLING
SIMSCRIPT CODE FOR DIRECTED-GRAPH COUPLING

' ' EX2.com file
$!@SIMDEF
$!LIBRARY/create BUFFER
$delete *.obj;*,*.lis;*,*.exe;*
$purge
$simcomp ex2.pre,ex2.gen,-
ex2.rtn library=buffer delete delete options=noxor
$simlink buffer/library/include=main
$!SIMRUN BUFFER

' ' --------------- Preamble files ---------------

' ' Ex2.pre file
PREAMBLE
normally mode is integer
processes include GEN, STOP.SIM
define gen_int_tm as a variable
define connect to mean call

' ' HCS.pre file
temporary entities
every HCS has an id_hcs, a HCS_couple, a num_HC_in_HCS
define id_hcs, num_HC_in_HCS, couple_HCS as variables
define inst_HCS as an integer, 1-dimensional array
substitute this line for HCS.out1
HCBF.inl( inst_hc(1) , pk_id )
substitute this line for HCS.out2
OUT.inl( pk_id )

' ' HC.pre file
processes
every HPROC.WORK has an a.hhcs_id, an a.hhc_id, an a.hpk_id
temporary entities
every hc has an id_hc, a hproc_free, a hc_couple1,
a hc_couple2,
owns a que_h
define que_h as a fifo set
define id_hc, hc_couple1, hc_couple2 as variables
define inst_HC as an integer, 1-dimensional array
define num_que_h, hproc_tm, hproc_free as variables
define HCBF.outl to mean HCPROC.inl
define HCPROC.out2 to mean HCBF.in2
substitute this line for HCPROC.out3
  HCS.in2( hc_couple1( hc_id ), pk_id )
substitute this line for HCPROC.outl
  HCBF.inl( hc_couple2( hc_id ), pk_id )

'' PK.pre file
temporary entities
every pk has a id_pk, a daddr, a saddr, a data, a cf,
  may belong to the que_h
define id_pk, data as variable
define cf, daddr, saddr as text variables
define false to mean 0
define true to mean 1

'' GEN.pre file
substitute this line for gen.outl
  HCS.inl( inst_hcs(l), pk_id )
end
process STOP.sim
  stop
end

'' ---------- ex2.gen file ----------
MAIN
  use unit 2 for input
  use unit 6 for output
  reserve inst_hcs(*) as 5
  reserve inst_hc(*) as 5
call READ.DATA
call HCS.gen
  activate a GEN now
  activate a STOP.SIM in 50 units
  start simulation
end

'' ---------- HCS.gen file ----------
routine HCS.gen
define i,j,num_HCS,num_HC as variables
  read num_HCS
  for i=1 to num_HCS
    do
create a hcs called hcs_id
    let inst_hcs(i) = hcs_id
    let id_hcs(hcs_id)=i

read num_HC
let num_HC_in_HCS(hcs_id) = num_HC
call HC.gen( hcs_id )
loop
return
end

' ' ---------- HC.gen file -----------------
routine HC.gen( hcs_id )
declare i as a variable
for i=1 to num_HC_in_HCS( hcs_id )
do
create a hc called hc_id
let inst_hc(i) = hc_id
let id_hc(hc_id) = i
let hproc_free(hc_id) = true
loop
for i=1 to num_HC_in_HCS( hcs_id )
do
if i = 3
let hc_couple1( inst_hc(i) ) = inst_hcs( 1 )
let hc_couple2( inst_hc(i) ) = 0
else
let hc_couple1( inst_hc(i) ) = 0
let hc_couple2( inst_hc(i) ) = inst_hc( i+1 )
always
loop
return
end

' ' ---------- EXP.rtn file -----------------
routine READ.DATA
read num_que_h, gen_int_tm, hproc_tm
skip 2 lines
print 1 lines with num_que_h, gen_int_tm, hproc_tm thus
num_que_h = *** , gen_int_tm = *** , hproc_tm = ***
skip 2 lines
return
end

' ' GENerator
process GEN
define pk_id, data_value as saved, integer variables
let pk_id = pk_id + 1
create a pk called pk_id
let data_value = data_value + 1
let data(pk_id) = data_value
let id_pk(pk_id) = data_value
connect GEN.outl
wait gen_int tm units
reactivate a GEN now

end

''Output
routine OUT.inl(pk_id)
define pk_id as a variable
   print I line with time.v , id_pk(pk_id) thus
   at time ***.** output pk ****
return
end

''------------------ HCs.rtn file ------------------
routine HCs.inl(hcs_id,pk_id)
define hcs_id, pk_id as variables
call HCS.outl
return
end

routine HCs.in2( hcs_id,pk_id)
define hcs_id, pk_id as variables
if hcs_id = 0
else
call HCS.out2
always
return
end

''------------------ HC.rtn file ------------------
routine HCBF.inl (hc_id, pk_id)
define hc_id, pk_id as variables
if hc_id = 0
else
   if que_h( hc_id ) is empty
      if hproc_free( hc_id ) = true
call HCBF.outl(hc_id, pk_id)
else
   file pk_id in que_h( hc_id )
always
else
   file pk_id in que_h( hc_id )
always
always
return
end

routine HCBF.in2( hc_id, pk_id)
define hc_id, pk_id as variables
let hproc_free( hc_id ) = true
if que_h( hc_id ) is not empty
    remove the first pk_id from que_h( hc_id )
call HCBF.out1 ( hc_id, pk_id )
always
return
end

routine HCPROC.ini( hc_id, pk_id)
define hc_id, pk_id as variables
    activate a HPROC.WORK giving hc_id, pk_id now
return
end

process HPROC.WORK given hc_id, pk_id
    define hc_id, pk_id as variables
    let hproc_free( hc_id ) = false
    work hproc_tm units
    call HCPROC.out2( hc_id, pk_id)
call HCPROC.out1
    call HCPROC.out3
end
APPENDIX C

SIMSCRIPT CODE FOR BROADCAST COUPLING
SIMSCRIPT CODE FOR BROADCAST COUPLING

' HCS.pre file for ex3
temporary entities
every HCS has an id_hcs, a HCS_couple, a num_HC_in_HCS
define id_hcs, num_HC_in_HCS, couple_HCS as variables
define inst_HCS as an integer, 1-dimensional array

substitute these 8 lines for HCS.out1
if id_pk(pk_id) < 5
    call HCBF.inl( inst_hc(1), pk_id )
else if id_pk(pk_id) < 10
    call HCBF.inl( inst_hc(2), pk_id )
else
    call HCBF.inl( inst_hc(3), pk_id )
always
always
substitute this line for HCS.out2
OUT.inl( pk_id )

' HC.gen for ex3
routine HC.gen( hcs_id )
define i as a variable
for i=1 to num_HC_in_HCS( hcs_id )
do
    create a hc called hc_id
    let inst_hc(i) = hc_id
    let id_hc(hc_id) = i
    let hproc_free(hc_id) = true
loop
for i=1 to num_HC_in_HCS( hcs_id )
do
    let hc_couple1( inst_hc(i) ) = inst_hcs( 1 )
    let hc_couple2( inst_hc(i) ) = 0
loop
return
end
REFERENCES


6. Stallings, William (1983), Local Networks University of California Santa Cruz


