NOTICE

The quality of this microform is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Reproduction in full or in part of this microform is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30, and subsequent amendments.

AVIS

La qualité de cette microforme dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

La reproduction, même partielle, de cette microforme est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30, et ses amendements subséquents.

APPLICABILITY OF
PROCESS ACTIVITY DIAGRAMS
AS A SYSTEM DESIGN TOOL.

by Jose M. Duran

A thesis presented to the
School of Graduate Studies and Research
of the University of Ottawa
in partial fulfillment of the
requirements for the degree of
Master of Applied Science
in Electrical Engineering.

Ottawa-Carleton Institute in Electrical Engineering.

OTTAWA. Ontario, May 1987.

ISBN 0-315-53781-7

ABSTRACT.

Process Activity Diagrams (PAD's) is a graphical tool that was developped for the specification and design of special purpose real time systems. In this framework, the system is specified by a number of processes which are represented as sequences of activities.

The purpose of this thesis is two fold; first to show the feasability of developing a specification tool for systems defined in terms of PAD's. In line with this, a matrix editor to enter the system specifications was developed. The verification of the information entered is then investigated.

In the second part the applicability of PAD's for the design of multitasking systems is considered. A methodology for the efficient partitioning of the system processes into interacting tasks is presented. This partitioning takes into account the characteristics of the operating system and of the underlying hardware.

ACKNOWLEDGEMENTS.

The author wishes to express his gratitude to his thesis advisor, Dr. Moshe Krieger for his guidance and encouragement. His numerous ideas and comments provided the necessary direction, and his continuous support and counselling were definite assets throughout the duration of the graduate studies of the author.

The author is grateful to his fellow researchers for their help in the development of this thesis, in particular Mr. Robert Joannis, Mr. Charles A. Gauthier, Mr. Vojislav Radonjic, Mr. Marco Naveda and Mr. Jean-Pierre Lachance.

The author does not forget his friends in room B-403 of Colonel By building who made his long years of the Masters program a rewarding experience.

The author wishes to thank the Advanced Real Time Toolset (ARTT) project for their financial support.

THESIS OUTLINE.

LIST OF FIGURES.

CHAPTER 2. page

CHAPTER 3.

CHAPTER 4

CHAPTER I

INTRODUCTION.

The availability of advanced high performance microprocessors and microprocessor based systems has shifted the job of designing computing systems from the computer specialist to the application specialist. Furthermore, the system designer is faced today with the task of designing increasingly complex systems, coordinating efficiently the interaction of the different processes in the system with very few development tools. To implement a well engineered system, the designer must have tools that allow the specification of the system requirements and that map these requirements into the proper hardware and software.

Basically, two types of system specification tools are presently available or being developed, language oriented tools [PERN 84] and graphical tools. Graphical tools may simplify considerably the task of developing the specifications, and are generally also simpler to understand by the user who may not be a computer expert [RAMA 84].

Some graphical tools have been proposed for the specification of complex processing systems, such as Petri Nets [PETR 65] [PETE 77], Data Flow Graphs [HWAN 84], Activity Channel Pools (ACP) [ALLW 80] and Process Activity Diagrams (PAD's) [KRIE 86]. In [JOAN 86] an analysis of the applicability of these tools for the specification and design of real time systems was considered; it was argued that for the non-computer expert, Process Activity Diagrams are simpler to learn and to use.

The purpose of this thesis is two fold; first to show the feasibility of developing a specification tool for systems defined in terms of PAD's. In line with this, a matrix editor to enter the system specifications was developed. The verification of the information entered based on the syntax of PAD's, is then investigated. In the second part the applicability of PAD's for the design of multitasking systems is considered. A methodology for the efficient partitioning of the system processes into interacting tasks, which is a major problem in the implementation of multitasking systems, is presented. This partitioning takes into account the characteristics of the operating system and of the underlying hardware.

In more detail, the chapters of this thesis cover the following:

The first part of chapter two reviews the principal characteristics of PAD's, with it's components the Activity

Sequence Networks (ASN's) and the Activity Timing Charts (ATC's). In the second part of this chapter, an ASN matrix editor is proposed; this relatively simple semi-graphical editor allows the entering and modification of the characteristics of the processes that form the system, as well as some quick visual checking. It was originally expected to implement a full graphical tool, by integrating it with a symbolic editor (SYMED), which is now developed by the Advanced Real Time Toolset (ARTT) project. However, the matrix editor was ready before the graphical editor and the integration of the ASN matrix editor with SYMED is left as future work. This semi-graphical editor serves to prove the feasibility of working with PAD's.

Chapter three covers a major aspect of the modeling and design process: verifying (verification) the correctness of the modeling of the system's behavior, represented by ASN's. This verification involves two types of errors, incomplete and incorrect use of the basic characteristics of the PAD notation, and the inconsistencies appearing from incorrect connectivity in the constructs of the ASN's, like deadlocks and multiple triggering. A software tool was developed for the first type of errors, while three different approaches are proposed for the detection of the second type of problems.

Chapter four discusses the general characteristics of the real time multitasking operating system and of the underlying hardware that have to be considered by the designer when

partitioning the system processes into tasks. Based on this discussion, a set of guidelines for the activity to task allocation of event driven real time system defined in terms of PAD's is proposed. To illustrate this, two examples involving two multitasking operating systems are considered.

Two appendices are included: appendix A includes the listing of the ASN matrix editor (EDITOR), while appendix B includes the listing of the program INSPECT, which covers the verification of the simple errors entered during the editing process.

CHAPTER II

A MATRIX EDITOR FOR SYSTEM SPECIFICATION WITH PAD'S.

2.1 / Introduction.

This chapter presents a simple matrix editor for the specification of special purpose event driven microprocessor based systems. It is assumed that the system considered here will have been specified using Process Activity Diagrams (PAD's) [JOAN 86], [KRIE 86].

Special purpose microprocessor based systems can be classified as embedded, turnkey, distributed, man oriented, real-time, etc.. However, most special purpose systems have fundamentally a common feature in that they react to requests or events, generated asynchronously by the environment in the form of signals or commands, and that the event generated triggers a particular response from the microprocessor based system. This is why these systems are said to be event driven.

The design of event driven systems involves essentially the correct determination of the behavior of the system, as it interacts with the environment. The system reacts to asynchronously generated events by means of responses. For each event there is a particular response associated with it, which

can be characterized by a single action or by a sequence of actions (a process).

These responses may include stringent time constraints, usually less critical in man oriented systems, where the human factor is in general much slower compared to the processor time-scale, than in a real-time situation. Real-time constraints involve the capability of the system to detect asynchronous events and execute the required response within the prespecified time.

This chapter is divided into two sections:

- in the first one, to provide the proper background, an overview of the concepts and characteristics of Process Activity Diagrams will be presented.

- in the second section, an editor that allows entering and manipulating the parameters of a system specified by means of PAD's will be presented.

2.2 / Overview of Process Activity Diagrams.

Process Activity Diagrams (PAD's) is a graphical tool for the specification and design of event driven systems. When defining a system using PAD's, the design engineer must be able to explicitly indicate the behavioral characteristics of a system by providing the sequencing and timing requirements between the environment and the activities in the system.

To indicate these requirements, Process Activity Diagrams

include two graphical elements, the Activity Sequence Networks (ASN) and the Activity Timing Charts (ATC).

2.2.1 / Activity Sequence Networks (ASN's).

Activity Sequence Networks or Nets (ASN's) represent the sequence of activities that have to be executed in response to an external event, called the starting event of the process. An activity is a well defined segment of code whose execution requires only one processor (or executing device) and which once initiated does not need to wait for additional data requirements, from other activities or external events. This means that for an activity to be initiated, all the input parameters required for its execution must be present.

In a system involving several processes, a simple table can associate processes with their corresponding starting events. If the starting event for a process occurs again, while the process is still being executed, then depending on the system implementation, several choices exist:

- ignore any new occurrences of the event while the process is awakened.

- store new incoming events in a queue, if only one instantiation of the ASN can be awaken at any time.

- assign the event to a copy of the ASN. This requires the existence of several copies of the same ASN (all exactly alike), and have the system determine the event to process assignment.

- finally, create a new instantiation of the ASN. This

however adds more overhead and the need of process management capabilities, like the utilization of process descriptors, etc..

2.2.1.1 / The nodes in an ASN.

An ASN is represented by a set of nodes interconnected by means of arcs. These arcs are directed, from node to node to show the sequencing of activity execution. An ASN is thus a directed graph. There are several types of nodes that can be used in a network; these are *activity initiator, branch, merge,* wait, *timer/counter, gate* and *end* nodes. Each one of these nodes has an specific role in defining the execution of the actions in a process.

The fundamental node in the definition of a system using PAD's is the *activity initiator* node, or simply *activity* node. This node requires all of its input triggers to be present before the activity can be scheduled for execution. The presence of these triggers indicates the availability of the parameters (control and/or data) that this activity needs. All the output triggers of the activity node are simultaneously set at the completion of the execution of the activity. This allows the propagation of the flow (control and data) in the process.

In order to indicate the end of a process, an special node is used; this is the *end* node. When this node is reached the process is assumed to be completed. This node does not have any

output triggers. It is possible to imagine this node as an activity node, whose sole 'activity' is to inform the system that the response to the event that started this process has been completed.

In order to introduce decision making capabilities in an ASN, *branch* nodes are used. This node provides the process structure with alternate execution paths, based upon some process (or system) status or condition variables. The current state of the condition variable of this branch node determines what are the output triggers that are activated, when all the input triggers are present.

Because the branch node provides a choice between executable paths, an extra node called the *merge* node is required to join back these paths, into a single stream of execution.

PAD's offer the possibility of specifying concurrency of activity execution within a process. To indicate the end of two (or more) concurrent paths a node like the activity node, but with no actual action associated with it, is used; this is the *wait* node. This node is also used when no particular action is to be taken. Alternately, an activity or a branch node can be used to recombine two or more concurrent paths.

In order to block the propagation of triggers in an ASN, a *gate* node is introduced. This node is initially disabled, and

does not respond until enabled. Only once enabled does the gate node start 'remembering' its input triggers. The gate node transmits an output trigger when all of its input triggers are present. The gate node can be disabled again, thus 'forgetting' all the triggers that were present. One must be careful when disabling a gate node: the source of the disable must somehow reach the end of the process, that is the end node; otherwise no trigger propagates further than this gate node, causing a deadlock situation in the ASN.

The last node in the set is the *timer/counter* node. This is the most complex node. As the gate node, it has one enable and one disable inputs, and it is also initially disabled. However it may only have one input trigger. When the node is enabled, the initial counter value is passed to it and put in a 'counter register'; the register count is decremented by one every time the input trigger is activated, and provided the node has no beer disabled. When the counter register reaches zero, then the output triggers are activated and the node is reset to the disabled state. However, if the node is disabled before the zero condition occurs, then the count is forgotten. The number representing the quantity of the count is passed on to the node every time the node is enabled.

There are two types of input triggers to all of these nodes; 'or' type, as in the case of the *merge* node, which sets all the output triggers as soon as any one of its input triggers is set,

and 'and' type in which case all triggers must be present in order for the output triggers to be activated. All nodes except the *merge* node have an 'and' type of input.

In the case of activity, branch, gate and wait nodes, as soon as all of the input triggers are present, the output triggers of the node are activated and the input triggers are reset. These nodes remember the input triggers so if one trigger occurs more than once while other triggers to the same node are being expected, multiple triggering of the same input does not affect the output.

This is the basic set of nodes that can be used to specify the behavior of a system. These nodes were chosen for their convenience in specifying control oriented problems. However, this set is not closed and other nodes, perhaps more application dependent, can be defined and incorporated into the set.

2.2.1.2 / Example of an ASN.

Figure 2.1 shows one example of an ASN involving all the nodes described above.

In this example, when the process is initiated (evel), the number of times activities act2 and act3 have to be executed is entered as the parameter 'n' (act1). Then these two activities (act2 and act3) are executed the requested number of times (condition true and counter expiration). The process stops as soon as any one of the following two conditions occur: either

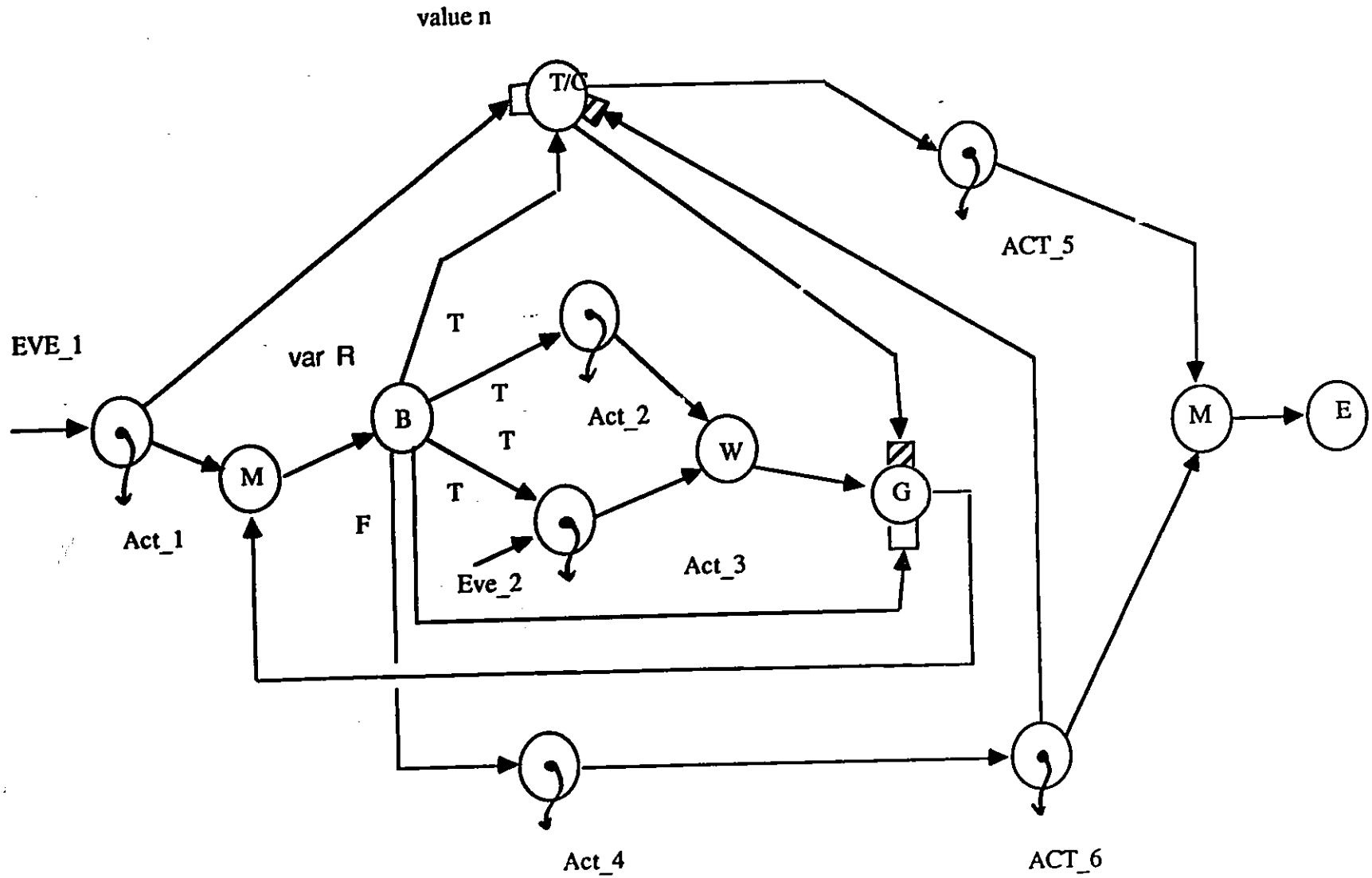


Figure 2.1 Activity Sequence Net ASN1.

activity is incorrectly executed for some reason (condition false) in which case a message to the user is issued (act4), or the counter expires having executed the activities correctly. In both cases the process is terminated.

2.2.1.3 / The ASN matrix.

The network of nodes forming the ASN can be entirely described by an ASN matrix, formed by columns specifying each node in the ASN, and rows of the corresponding events, internal and external. An internal event is the event resulting from the output triggering of a node in the net. If a node, with corresponding internal event i, is connected to the input trigger of a node j, then a 'I' (indicating triggering) is placed at position (i,j) in the matrix. If the output trigger of a node (with internal event k) is connected to the *enable* or *disable* input of a gate or timer/counter node m, then an 'E' or a 'D' respectively is placed at the position (k,m) in the matrix.

Figure 2.2 shows the ASN matrix corresponding to the ASN in figure 2.1.

To represent a system a number of ASN matrices, each corresponding to one ASN, may be required.

2.2.2 / Activity Timing Charts. (ATC's)

To describe activity execution times, a timing bar can be associated with each activity in the system; this bar is

		Ac1	Me1	BrR	Ctn	Ac2	Ac3	Wai	Gat	Ac4	Ac5	Ac6	Me2	End
1	Ac1	0	I	0	E	0	0	0	0	0	0	0	0	0
2	Me1	0	0	I	0	0	0	0	0	0	0	0	0	0
3	Br T R	0	0	0	I	I	I	0	E	0	0	0	0	0
4	Br F R	0	0	0	0	0	0	0	0	I	0	0	0	0
5	Cnt n	0	0	0	0	0	0	0	D	0	I	0	0	0
6	Ac 2	0	0	0	0	0	0	I	0	0	0	0	0	0
7	Ac 3	0	0	0	0	0	0	I	0	0	0	0	0	0
8	Wait	0	0	0	0	0	0	0	I	0	0	0	0	0
9	Gate	0	I	0	0	0	0	0	0	0	0	0	0	0
10	Ac 4	0	0	0	0	0	0	0	0	0	0	I	0	0
11	Ac 5	0	0	0	0	0	0	0	0	0	0	0	I	0
12	Ac 6	0	0	0	D	0	0	0	0	0	0	0	I	0
13	Me 2	0	0	0	0	0	0	0	0	0	0	0	0	I
14	Ex Eve1	I	0	0	0	0	0	0	0	0	0	0	0	0
15	Ex Eve2	0	0	0	0	0	I	0	0	0	0	0	0	0

Figure 2.2 ASN matrix of ASN in figure 2.1

essentially divided into three sectors.

- the first sector indicates the time required for transferring all the data necessary to execute that activity.

- the second sector corresponds to the actual execution time of the processing involved in this activity. Since execution may be data and processor dependent, there may be different values associated with this sector: average estimates, minimum, maximum, etc..

- the last sector corresponds to the output transfer time;

Activity Timing charts (ATC's) represent the timing relationships between the activities in a process, and are similar to Gantt charts. These ATC's are charts whose role is to define the execution time of activities and to show the amount of parallelism that can be found in a particular process at any given time.

Assuming a processor is always available, the total execution time of a process is equal to the sum of the execution times of the longest sequence of activities in the process.

Because in an ASN all nodes except the activity node deal with control of the execution paths of the activities, they do not have timing bars. The presence of other nodes is seen through the definition of the different condition variable based scenarios and through the determination of the ordering and timing of the activities in the ATC's.

ATC's are drawn assuming a processing device is made available as soon as requested (infinite number of processors).

The ATC's can be used, for the specification of the minimum number of processors required in a system, minimum execution time for a process, maximum level of concurrency attainable, etc.. In the case of a uniprocessor system, the minimum execution time is of course the addition of the execution time of all the activities in the process.

The ATC includes one component for each possible condition based scenario in the ASN.

Figures 2.3 and 2.4 show the ATC's of the ASN in figure 2.1, for the two scenarios.

The ATC's can be obtained by means of a single stepped (interactive) simulation of a specially modified version of the PAD simulator. At each unit of time, the activities being executed are recorded and appropriately displayed. Some ATC's obtained like this appear in chapter three.

2.2.3 / Summary of the basic characteristics.

From the above discussion, it is possible to determine the basic characteristics necessary to describe a system using PAD's.

1 - Each ASN of the system must include a well defined starting event, and at a higher level, all the processes of the system should be defined, together with the initiating event for each of them. Failing to specify the starting event for a process will rend that process deaf to external stimuli.

2 - Similarly, failing to specify the process initiated by

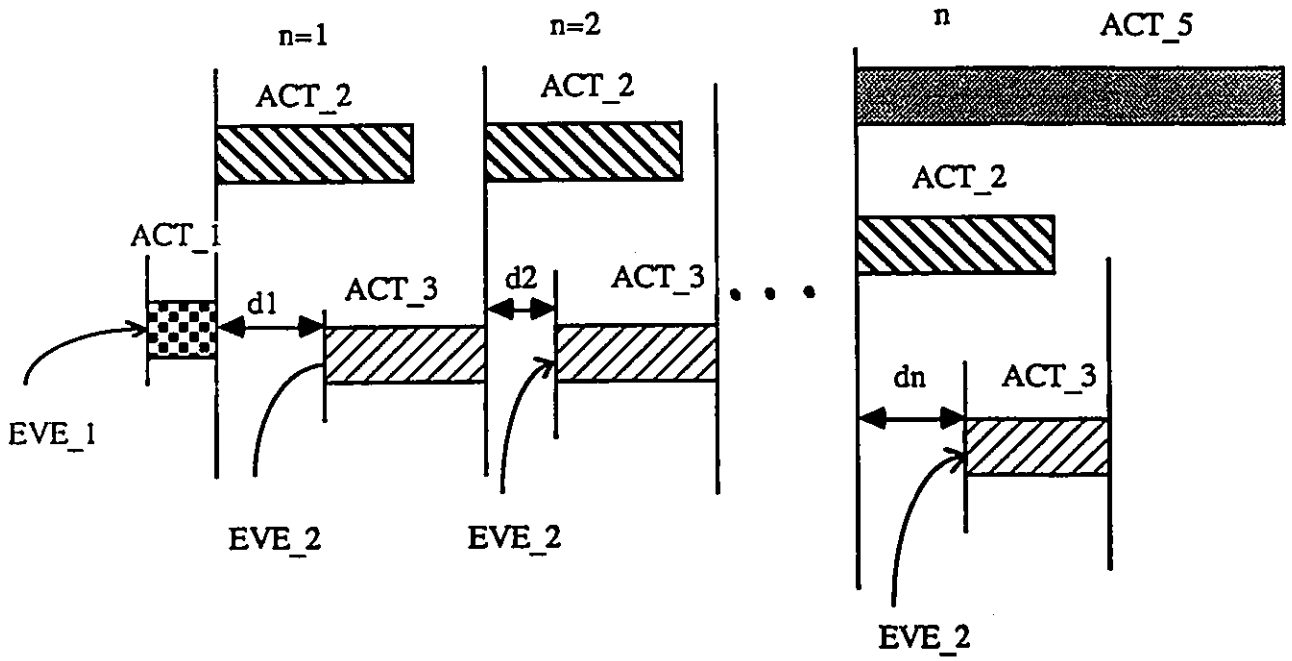


Figure 2.3 ATC of ASN1
if Condition is true.

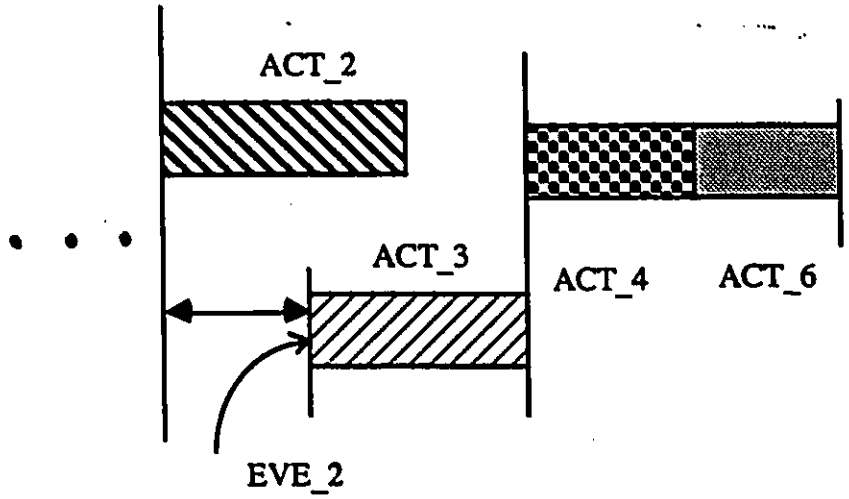


Figure 2.4 ATC of ASN1 if 'R'=false.

the occurrence of an external event will show no response.

3 - Within each process or ASN, the sequence of activities must be correctly and completely specified, indicating the control flow of the process. The data flow in a process is only indicated indirectly in the ASN. A more explicit indication of the data flow in a process may be indicated by an annotated ASN or a interactivity data transfer table.

4 - Each process, activity and external event must be appropriately identified.

In relation to the connectivity in the ASN, one should be able to predict whether or not the control flow indicated by the matrix is executable or if problems, such as deadlocks, may result. This would involve the verification of the connectivity definition of a net. This problem is the topic of chapter three.

In the above description, it was assumed that all activities are *atomic*, that is simple and indivisible. Alternately, an activity can be defined, in system specification terms, as a composite activity, which could in fact involve several of these *atomic* activities in sequence or in any other order. This basically means that the nesting of processes is possible.

2.3 / The ASN matrix editor.

The purpose of this section is to present an ASN matrix editor for the specification of the ASN matrices of a system.

2.3.1 / Role of the EDITOR.

In order to facilitate the use of Process Activity Diagrams as a system specification and design tool, an interactive editor is a basic requirement. Such an editor should allow the following:

- definition of the system processes, one by one, with the capability of adding new processes to the system.

- definition of each process on a per node and external event basis.

- manipulation of the nodes and of the events; that is adding, deleting, inserting nodes and external events.

- simple and fast determination of the connectivity definition of an ASN by means of the matrix parameters.

- Finally, one should be able to redefine ASN matrices, activities and external events, as well as deleting processes from the system definition.

Since Activity Sequence Networks can be specified using the graphical notation or the ASN matrix, there was the option of developing a graphical editor or an ASN matrix editor. Although a graphical editor would have been friendlier, the decision to go to a matrix editor was made solely because of time requirements. However, this is enough to show the feasibility of such an editor. This ASN matrix editor was implemented as a menu driven tool because of the simplicity and friendliness of the user

interface.

The editor was written in Turbo Pascal and is intended to be used on IBM personal computers, with two disk drives. The disk drive A contains the executable program, while all the data and information of the PAD defined system is to be in drive B.

Because of the screen limitations of regular 80x25 character screens, there is a low upper limit to the number of nodes and events a particular ASN may contain. However, an ASN with sixteen nodes can already describe fairly complex process, as the one shown in figure 2.1.

2.3.2 / EDITOR. Main Menu Commands.

The editor provides the following menu of commands:

1 - 'Create an ASN'; this command is issued in order to define a completely new ASN. This involves the definition of all the parameters of the ASN.

2 - 'Edit an ASN': if at least one ASN has been already defined and saved in the system definition files, then this command can be selected. Otherwise, the error message 'File of system does not exist' is issued. If the system and the ASN exist, the process to be edited is found by means of its four character label, displayed and set ready to be worked on.

3 - 'List parameters'; the third choice requires as well the existence of an ASN in the system. This selection will:

- list the labels and corresponding numbers of:

- processes (ASN's),

- Activities and

- External events, existing in the system.

Furthermore, a provision has been added to display the contents of any ASN matrix in the system, on a per node basis, but with no editing capability.

4 - 'Change system'; the fourth choice allows the user to work on another, new or already defined system definition.

5 - The last command 'Quit' terminates the current session and exits to DOS.

2.3.3 / Node and Event definition.

Whether the user selects the first or second option (Create or Edit an ASN), the name of the ASN to work on must be specified as a four-character label. Once this label has been entered it is associated with the current ASN.

At this point, the program checks at this level whether the specified label exists in the system definition, and if a match occurs, a different label for the ASN must be specified.

To enter or edit an ASN, the user is faced with four windows in the screen (see figure 2.5).

- On the bottom right, a small window displays the different possible commands in this mode; this is the *menu window*.

- On the bottom left, a rectangular interactive window will display any messages from the EDITOR to the user, and it is here

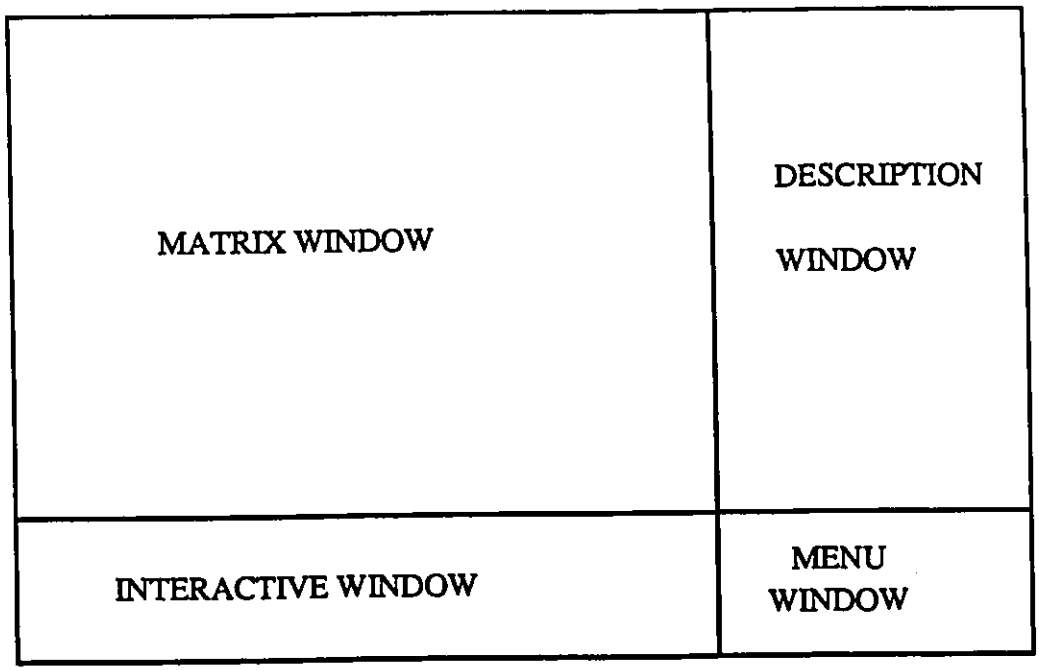


Figure 2.5 Screen during editing session.

where the user enters any requests for information; this is the *interface window*.

- On the top right, in a presently empty window, if the user is in the 'Create' mode, the description of the different nodes and events of the ASN being currently edited are shown. In the 'Edit' mode, the complete description of the invoked ASN appears automatically; this is the *description window*.

- On the top left, and the largest of the windows, the ASN matrix is progressively displayed. Progressively, because it grows in size as long as nodes and events are added. The columns of the matrix represent the nodes and the rows represent the different events, internal and external; this is the *matrix window*.

In the node and event definition mode, the user can select five commands: add or delete node, add or delete external event and exit. First the add node and add event commands are considered and explained.

2.3.3.1 / Add node.

Upon issuing this command, a menu displaying the type of node to be added appears.

The nodes are chosen by selecting a character:

- 'A' for the activity initiator node,
- 'B' for the branch node,
- 'C' for the timer/counter node and

- 'G' for the gate node,
- 'M' for the merge node,
- 'W' for the wait node,
- 'E' for the end node.

Depending on which of the seven types of nodes is chosen, a different sequence of actions will be taken.

- In the case of an *activity initiator* node, the user must produce a label (four characters) to identify the activity. If this label is identical to another activity's label, then a warning message is issued. As it is possible to have more than once the same activity in an ASN, then if the same label is reentered again, it is accepted. While the activity nodes are being specified, a vector that associates the activity labels to the activity numbers, is created. To completely specify the system, it would be ideal to associate at this point all the parameters describing a given activity, such as execution time, processor allocation or starting address of the activity code, etc. This EDITOR does not include such a feature. Such a program (DESCRIPTOR) is presently being developed, but more as a system activity description tool.

A corresponding event is automatically associated with this node.

- In the case of a *branch* node, the user must specify the condition variable associated with this branch node. The

addition of a branch node creates automatically two corresponding internal events, one for the TRUE eventuality associated with the condition variable and a second one for the FALSE. This editing tool allows only for Boolean type of condition variables.

- When adding a *timer/counter* node to the definition of the ASN, a warning message appears first in the interactive window, reminding the user that a hang up resulting in a deadlock situation may occur if this node is not correctly used; no checking is done here and it is just a reminder. The timer/counter node has a counter variable associated that must be specified. One single internal event is generated.

- In the case of a *gate* node, the same kind of warning message as issued in the previous case is displayed, as the GATE node may also lead to the same type of error. One single event is generated.

- If the node to be added is either a *merge* or a *wait* type node, then the node is simply added to the existing node description list and a corresponding internal event is generated.

- Finally, the last node to be added to the list should be the *end*. The inclusion of such a node does not generate any event, as explained in the definition of the node.

Whenever a node is to be added, the EDITOR asks the user

whether the node is to be placed at the end of the description list of nodes or inserted between two nodes in the existing list; if at the end, the node is appended after the last node; otherwise, the position where the node is to be inserted must be specified. In this case, the existing nodes located after the node position are shifted down by one, as well as the corresponding internal events (also one position, but two if the node to be inserted is a branch node).

At this point the user is ready to enter the system definition. Some remarks and recommendations are due here:

1 - It is recommended that nodes in the ASN be numbered starting with node number 1 as the node which receives the starting external event, and the END node being the node with the largest node number. This is only for consistency reasons.

2 - It is recommended that all nodes in the ASN be defined before the external events. This simplifies the understanding of the displayed matrix.

All the nodes of the ASN have been included in the definition and it is now the turn of the external events.

2.3.3.2 / Add External Event.

Only external events need to be added to the node definition, as the definition of each node includes the corresponding internal events.

If an external event is to be entered, then the label of that event is requested. The identity of the event to be added is checked to see whether the event already exists in this process, in which case it is refused. Once a correct event is entered, one event is added to the matrix and the description of this event appears in the top right window.

The starting event of the ASN is (actually must be) entered as an external event and has to be included in the definition of the ASN. This is reminded to the user at the beginning of the creating session and, if no external events are entered for a particular ASN matrix, then when exiting the node and event definition session, the EDITOR forces the entry of an external event in the matrix definition; the user is simply asked to enter an external event before exiting.

Once all the nodes and events have been entered, then the user exits the node and event definition mode by entering the 'Exit' command.

Note that the declaration of the starting event of the ASN is done after the definition of the connectivity in the ASN.

2.3.3.3 / Delete node and Delete External Event.

The EDITOR provides the capability of deleting nodes and events from the definition of the ASN matrix.

This command will ask which node or external event is to be deleted from the definition. Once specified, the description is

rearranged. There are two separate commands, one to delete a node and one to delete an external event.

2.3.3.4 / Result of the node and event definition.

Figures 2.6 to 2.8 illustrate the results of some of the various steps of the editing process. They describe the process of entering ('Create') the matrix describing the ASN in figure 2.1.

Figure 2.9 shows the way the screen appears after the user has correctly entered the node and event definition. Notice that the ASN matrix appearing in the top left window is filled with zeroes (Ø).

This node and event definition is then saved. The user then starts the definition of the connectivity of the nodes in the ASN.

2.3.4 / Connectivity Definition.

This phase involves the specification of the connectivity parameters of the matrix and the definition of the starting event of the ASN.

2.3.4.1 / Matrix Parameter Definition.

In this phase the user defines the connectivity of the matrix by moving the cursor over the matrix and writing in the selected

```
   1
1  Ø
```

NODES	EVENTS
1.ACT act1	1.AC act1

Figure 2.6 Addition of an activity node.

```
   1  2  3  4  5  6
1  Ø  Ø  Ø  Ø  Ø  Ø
2  Ø  Ø  Ø  Ø  Ø  Ø
3  Ø  Ø  Ø  Ø  Ø  Ø
4  Ø  Ø  Ø  Ø  Ø  Ø
5  Ø  Ø  Ø  Ø  Ø  Ø
6  Ø  Ø  Ø  Ø  Ø  Ø
7  Ø  Ø  Ø  Ø  Ø  Ø
```

NODES	EVENTS
1.ACT act1	1.AC act1
2.MERGE	2.MERGE
3.BRAN R	3.BRA T R
	4.BRA F R
4.COUNT n	5.COUNT n
5.ACT act2	6.AC act2
6.ACT act3	7.AC act3

Figure 2.7 Screen during the process of editing ASN1

```
   1  2  3  4  5  6  7  8  9 10 11 12 13
 1 0  0  0  0  0  0  0  0  0  0  0  0  0
 2 0  0  0  0  0  0  0  0  0  0  0  0  0
 3 0  0  0  0  0  0  0  0  0  0  0  0  0
 4 0  0  0  0  0  0  0  0  0  0  0  0  0
 5 0  0  0  0  0  0  0  0  0  0  0  0  0
 6 0  0  0  0  0  0  0  0  0  0  0  0  0
 7 0  0  0  0  0  0  0  0  0  0  0  0  0
 8 0  0  0  0  0  0  0  0  0  0  0  0  0
 9 0  0  0  0  0  0  0  0  0  0  0  0  0
10 0  0  0  0  0  0  0  0  0  0  0  0  0
11 0  0  0  0  0  0  0  0  0  0  0  0  0
12 0  0  0  0  0  0  0  0  0  0  0  0  0
13 0  0  0  0  0  0  0  0  0  0  0  0  0
```

NODES	EVENTS
1.ACT act1	1.AC act1
2.MERGE	2.MERGE
3.BRAN R	3.BRA T R
	4.BRA F R
4.COUNT n	5.COUNT n
5.ACT act2	6.AC act2
6.ACT act3	7.AC act3
7.WAIT	8.WAIT
8.GATE	9.GATE
9.ACT act4	10.AC act4
10.ACT act5	11.AC act5
11.ACT act6	12.AC act6
12.MERGE	13.MERGE
13.END	

Figure 2.8 Screen after all nodes have been defined.

```
   1  2  3  4  5  6  7  8  9 10 11 12 13
 1 0  0  0  0  0  0  0  0  0  0  0  0  0
 2 0  0  0  0  0  0  0  0  0  0  0  0  0
 3 0  0  0  0  0  0  0  0  0  0  0  0  0
 4 0  0  0  0  0  0  0  0  0  0  0  0  0
 5 0  0  0  0  0  0  0  0  0  0  0  0  0
 6 0  0  0  0  0  0  0  0  0  0  0  0  0
 7 0  0  0  0  0  0  0  0  0  0  0  0  0
 8 0  0  0  0  0  0  0  0  0  0  0  0  0
 9 0  0  0  0  0  0  0  0  0  0  0  0  0
10 0  0  0  0  0  0  0  0  0  0  0  0  0
11 0  0  0  0  0  0  0  0  0  0  0  0  0
12 0  0  0  0  0  0  0  0  0  0  0  0  0
13 0  0  0  0  0  0  0  0  0  0  0  0  0
14 0  0  0  0  0  0  0  0  0  0  0  0  0
15 0  0  0  0  0  0  0  0  0  0  0  0  0
```

NODES	EVENTS
1.ACT act1	1.AC act1
2.MERGE	2.MERGE
3.BRAN R	3.BRA T R
	4.BRA F R
4.COUNT n	5.COUNT n
5.ACT act2	6.AC act2
6.ACT act3	7.AC act3
7.WAIT	8.WAIT
8.GATE	9.GATE
9.ACT act4	10.AC act4
10.ACT act5	11.AC act5
11.ACT act6	12.AC act6
12.MERGE	13.MERGE
13.END	
	14.EX eve1
	15.EX eve2

Figure 2.9 Screen after the node and event definition.

location in the matrix one of the possible types of triggers.

The user can enter four characters, I (for 1), D,E or Ø. There is no restriction at this point to where these triggers may be written to, and there is no need to refill the matrix with the zero (Ø) triggers. That input is only used in the case of changing an existing trigger back to a 'Ø'. The resulting matrix and node and event description for ASN 1 appear in figure 2.1Ø.

Once all the parameters of the ASN matrix have been defined, the 'exit' command is selected in order to enter the starting event for the ASN.

2.3.4.2 / Definition of the starting event.

At this point, a list of the external events in the ASN is shown in the bottom left interactive window. The user is then asked to specify the starting event for this ASN. This starting event must have been included in the definition of the external events of the ASN. The EDITOR does not accept any external event that has not been previously entered in the definition of the process.

Because the EDITOR cannot exit the node and event definition session leaving the ASN with no external events, it is guaranteed that when asked to determine which of the external events is the starting event, there will be at least one external event.

Once this connectivity definition has been completed, it is

	1	2	3	4	5	6	7	8	9	10	11	12	13
1	Ø	I	Ø	E	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
2	Ø	Ø	I	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
3	Ø	Ø	Ø	I	I	I	Ø	E	Ø	Ø	Ø	Ø	Ø
4	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	I	Ø	Ø	Ø	Ø
5	Ø	Ø	Ø	Ø	Ø	Ø	Ø	D	Ø	I	Ø	Ø	Ø
6	Ø	Ø	Ø	Ø	Ø	Ø	I	Ø	Ø	Ø	Ø	Ø	Ø
7	Ø	Ø	Ø	Ø	Ø	Ø	I	Ø	Ø	Ø	Ø	Ø	Ø
8	Ø	Ø	Ø	Ø	Ø	Ø	Ø	I	Ø	Ø	Ø	Ø	Ø
9	Ø	I	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
10	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	I	Ø	Ø
11	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	I	Ø
12	Ø	Ø	Ø	D	Ø	Ø	Ø	Ø	Ø	Ø	Ø	I	Ø
13	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	I
14	I	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
15	Ø	Ø	Ø	Ø	Ø	I	Ø	Ø	Ø	Ø	Ø	Ø	Ø

NODES	EVENTS
1.ACT act1	1.AC act1
2.MERGE	2.MERGE
3.BRAN R	3.BRA T R
	4.BRA F R
4.COUNT n	5.COUNT n
5.ACT act2	6.AC act2
6.ACT act3	7.AC act3
7.WAIT	8.WAIT
8.GATE	9.GATE
9.ACT act4	10.AC act4
10.ACT act5	11.AC act5
11.ACT act6	12.AC act6
12.MERGE	13.MERGE
13.END	
	14.EX eve1
	15.EX eve2

Figure 2.10 Screen after the connectivity definition.

saved and the ASN matrix definition session is over.

2.3.5 / Final Output of EDITOR.

The EDITOR creates three files, which are identified by the system name. The system name is in fact the name of the session the EDITOR is working on. These three files are:

- 'system name'.ASN, whose contents is used by the PAD simulator.

- 'system name'.NAM, which contains the user defined labels for the ASN's, the activities and the external events existing in this system. This file is used in the activity descriptor tool, to specify executing processor and execution time of the activity.

- 'system name'.CHA, which contains a brief system description useful for listing and error checking purposes.

These files are created, and updated each time the system definition is saved.

2.3.6 / Matrix Manipulation. MATEDIT.

The second program which forms part of the ASN matrix editing package is MATEDIT.

This program provides the following features:

- to make an identical copy of an ASN, within the same system. The purpose of this is essentially for systems where

multiple instantiations of an ASN are not permitted; it is possible to have the same ASN repeated a number of times, and have the executive do the event to process allocation when required to do so.

- to delete an ASN from the system.

- to rename an ASN, an activity or an external event in the system definition.

It is also possible to read the characteristics of the system, but with no editing capability, in the Listing mode, as in the case of EDITOR.

2.3.7 / Organization of the PAD Editing tool.

This software package is composed of several subpackages, each with its own inherent function. Figure 2.11 shows the organization of these programs.

It was intended to have the complete set be called from a single main program but this was not possible at this stage, because of time constraints and problems encountered when dealing with the characteristics of the processor of the IBM PC. Thus separate programs were implemented. They could however be included into a single large package with some modifications. A third program is presently under development, to serve as a system activity (DESCRIPTOR) descriptor tool, as was previously mentioned.

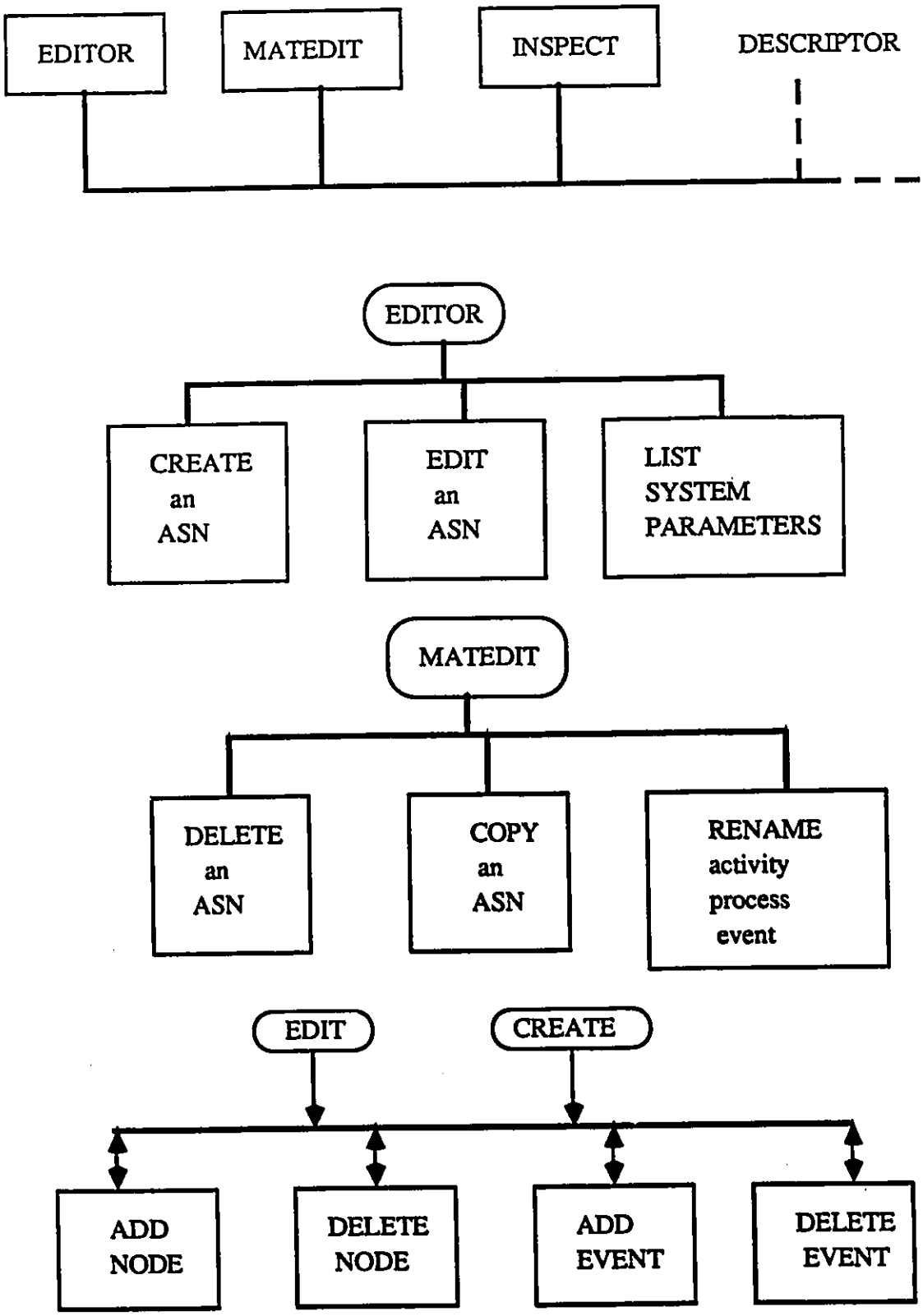


Figure 2.11 Organization of the Editing Tool.

2.4 / Concluding remarks.

In this chapter, the features and principal characteristics of Process Activity Diagrams (PAD's) were reviewed. PAD's is a graphical tool for the specification and design of event driven systems. They are composed of two graphical elements, Activity Sequence Networks (ASN's) which are like data flow graphs, and are used to indicate the control (and data) flow within a process, and Activity Timing Charts (ATC's) which indicate the execution time and possible execution paths, parallel or sequential, for a given process.

In section 2.3, a software ASN matrix editing tool for the ASN's was presented. This tool allows entering, editing and manipulation of the information necessary for the description of a system defined by means of PAD's. Two programs were used for such a purpose, EDITOR for the definition of the matrix of ASN's, and MATEDIT for matrix manipulation. Each of these programs was briefly explained and presented while the code, in Turbo-Pascal, is included as an appendix.

As a last remark, it must be said that for user friendliness a graphical editing tool would have provided a better interface. Such a tool should show the actual ASN's and ATC's on the screen, display instantaneously any modification done to them, provide an instant verification of the information supplied and provide as an output the corresponding ASN matrix in a format acceptable by

the PAD simulator. Although a graphical editor (SYMED) is now available, it was not ready when this project was started. As future work, an interface between the output of SYMED and the editor and the PAD simulator is presently under consideration.

CHAPTER III.

VERIFICATION OF THE SYSTEM DEFINITION.

3.1 / Introduction.

The previous chapter presented an ASN matrix editor that allows the specification of a system in terms of Process Activity Diagrams. For this information to be useful, it must correctly represent the behavior of the system. Consequently a major part of the editing process is the verification of the information entered. The purpose of this chapter is to present the verification aspect of an editor.

During system design it is important to include in the methodology several levels of 'checking', one corresponding to each design phase, to detect all errors introduced during that particular phase.

In simple terms, the design engineer starts the system design with a set of specifications provided by the customer, which can initially be assumed to be error-free [RAMA 84]. The next two steps of the design correspond to the modeling of the system and

to the design of a prototype.

- During the modeling phase the design engineer 'translates' the user specifications into a formal model that describes the system's behavior. This model must be then verified (*verification*) to check for the correct utilization of the modeling tool. Next, the model must be validated (*validation*) to check whether or not it reflects properly the system operation as stated by the user.

During the modeling process, unexpected situations or errors in the system specifications may be detected by the design engineer, in which case he informs the user of the incompleteness and/or inconsistencies of his specifications. This step is iterated until validation of the model.

- During the prototype design phase, the verified model is transformed into a prototype, which must be then *tested* for hardware and/or software problems. Once the prototype has been fully developed it is again checked under real-life conditions by means of on-site tests or simulations to validate the system operation.

Depending on the type of problem encountered (if any), this step may be iterated or it may be necessary to go back and modify the model or the specifications.

As a final step, the prototype is further refined by the product engineer. The testing and maintenance routines for the product are also developed during this phase.

Only the verification of the modeling process is considered in this thesis, and in particular those errors caused by the incorrect use of the PAD notation and errors due to incompleteness and/or incorrect system specification.

The modeling errors that result from these conditions can be classified into three groups:

- Incorrect execution errors, which are those occurring when the sequence of actions specified does not coincide with the requirements.

- Deadlocks, which are situations where the system (or part of the system) is halted because a process or processes are waiting for a particular event to occur and this event can not occur or is only generated by one of the blocked process.

- Temporal errors, which are those that occur because some combinations of events and conditions were not considered in the specifications. One such error is the critical race. These types of errors can be detected by means of a system simulation or during run time.

In the next two sections, the various types of modeling errors that can be introduced when using the ASN syntax notation or the ASN matrix EDITOR are considered. Subsequently, a method to detect these errors is presented in section 3.4.

3.2 / Elementary faults-INSPECT.

These errors include forgotten triggers or nodes and violations of the Activity Sequence Network notation. These faults are referred to as 'mechanical errors' because they may represent possible incorrect entries from an otherwise correctly defined ASN, or features forgotten during the editing process.

The following can be considered under 'mechanical errors':

- Undefined external events, and in particular the starting event of a given ASN;

- A given node of the ASN is never triggered (there are no input triggers to this node);

- A given event in the ASN never triggers (there are no output triggers from the node);

- No END node in a given ASN;

- No activity initiator nodes;

- Incorrect inputs to various nodes. In the case of a GATE and/or a TIMER/COUNTER node it means not including all three triggers (Enable, Disable and trigger) and/or incorrect triggering of other nodes (for instance Disabling an activity initiator node);

- Self-triggering of a node.

As a rule, the checking process for this type of errors is included in the editor; this was not possible at this stage because of constraints imposed by the memory segment characteristics of the processor. The editor only checks for the

presence of a starting event and the repetition of activity nodes. A separate program called INSPECT has been developed to detect all of the above 'mechanical errors'. INSPECT has the following features:

- It checks the list of nodes to detect if activity initiator nodes and an end node have been included in the definition of the ASN. If there are no such nodes a warning message is issued. In some situations it is possible to have an ASN with no end node, as in the case of an infinite feedback loop.

- It checks the input triggers to every node in the list, to see if they can all be triggered.

- It checks the output triggers of each node to determine if there is a 'cut' in the sequence of execution of the process.

- It checks the triggering of every node to determine if the type of triggering to the node is complete and correct. This step also checks if the gate and timer/counter nodes include all three types of triggers (I,D and E), and makes sure that all other five node types do not have D or E type of triggers.

- It checks the self-triggering of nodes and indicates the type of error that may result (multiple triggering in the case of a merge node, deadlock in all other cases).

3.3 / Connectivity errors.

Connectivity errors are faults which result from the

incorrect use of the Activity Sequence Network notation. There are two types of connectivity errors encountered in ASN's: deadlocks and multiple triggering.

3.3.1 / Deadlocks.

Two different types of deadlock can be distinguished in an ASN: *logic deadlocks,* caused by the incorrect use of nodes and/or wrong or missing connections (triggers) in the ASN, and *operational deadlocks,* which result from an incorrect and/or incomplete event specification.

3.3.1.1 / Logic deadlocks.

The errors appearing in the definition of an ASN that can result in a logic deadlock situation are caused by either closed loops or incomplete execution paths.

1 - Closed loops occur when a particular node in the ASN waits for a trigger that can only occur as a result (directly or indirectly) of the output triggering of the same node.

a / The self-triggering of a node is the simplest case of a deadlock caused by a closed loop; see figure 3.1. Because all the nodes except the merge node have an 'and' type of input, the self-triggering of a node causes a deadlock in all of these cases. The self-triggering of a merge node causes another type

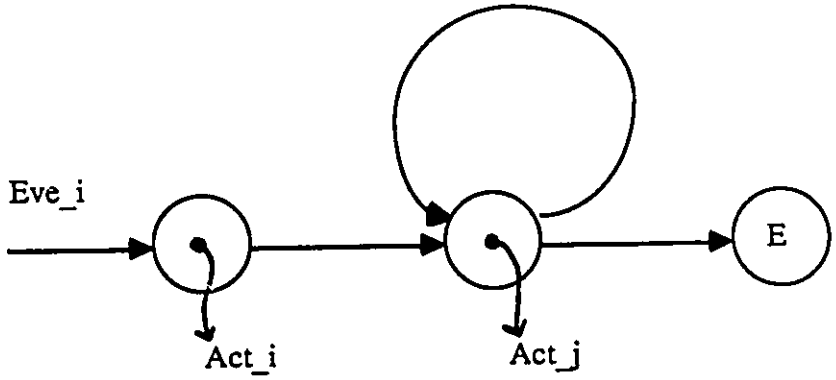


Figure 3.1 Self-triggering deadlock

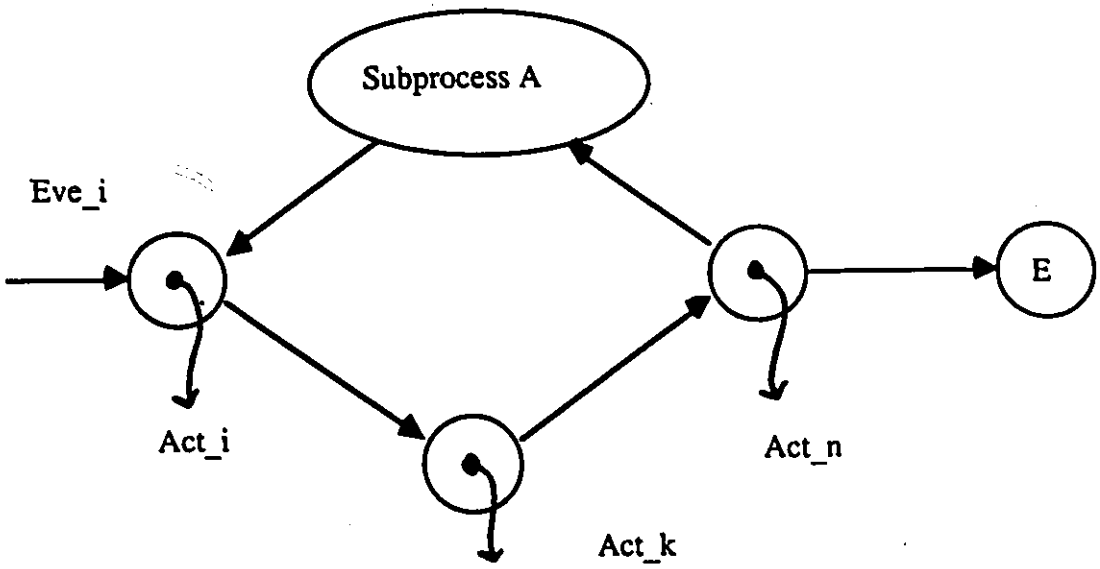


Figure 3.2 Closed loop deadlock.

of problem (multiple triggering) that is considered later in section 3.3.3. Therefore, self-triggering nodes always result in an error (deadlock or multiple triggering situation) in the ASN definition. This situation is detected by INSPECT.

b / The closed loop involving more than one node is a situation similar to the previously described case of self-triggering node. An example of this situation is represented in figure 3.2.

2 - Incorrect or missing connectivity arcs in an ASN lead to incomplete execution paths. There are several situations where an incomplete path can be found:

a / the required connectivity arcs in an execution sequence are missing, resulting in a 'cut' in the execution sequence of a process.

b / nodes are incorrectly used; the following cases are possible:

i / A deadlock can occur when the branch construct is used incorrectly, either if the alternate paths do not 'merge' (recombine at a merge node) or the merge node is not used properly. The first case results in a deadlock because the node used for recombination, as it is not a merge node, can never be executed because both triggers can never be present simultaneously.

The situation of two alternate paths not 'merging' together yields a deadlock whether the value of the condition variable is

true or false. Consider now the case shown of the ASN in figure 3.3; in this case, because the merge node is not used properly, a deadlock occurs if the condition is false. However, if the variable is found to be true, another type of error results, multiple triggering.

ii / Another situation that can cause a deadlock is when the output triggers of a branch node do not cover all the eventualities associated with the condition variable. The deadlock occurs when the unspecified condition happens. As an example consider a branching condition depending on the value of a variable x; assume that two alternate paths are possible, one for $x<4$, and the other for $x>4$, and that the value of x is $x=4$. In this case, because the condition occurring has not been specified, no action is taken and the branch node doesn't trigger. In this thesis however, as stated in chapter two, the type of the branch condition variable is considered to be Boolean.

It should be noted that, as was stated in the definition of branch nodes when using the EDITOR in section 2.3.3, the two internal events covering both states of the Boolean condition variable associated with the branch node are automatically defined in the event definition of the ASN. Thus if a matrix does not include one of the two conditions, it means that one of the corresponding events belonging to the branch node is filled with '0' triggers and does not trigger any node. This situation

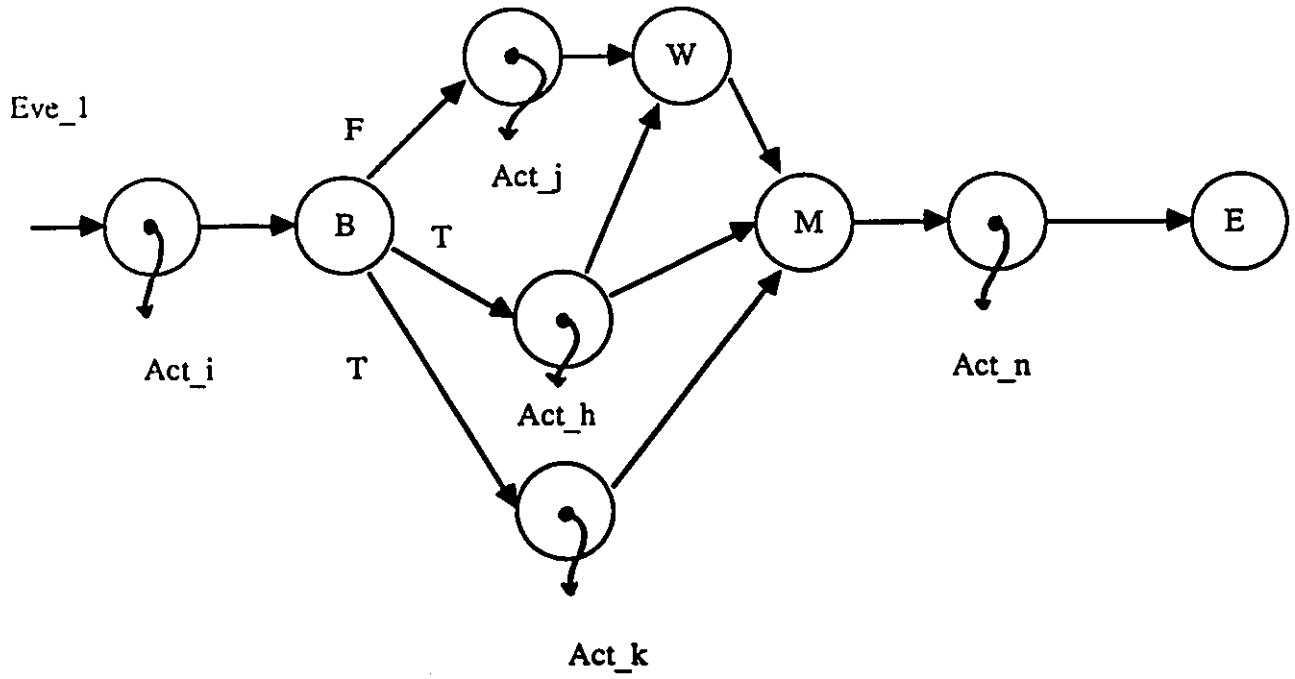


Figure 3.3 Deadlock and multiple triggering.

is detectable by INSPECT.

c / The incorrect use of gate and timer/counter nodes can be also a source of deadlock. Examine figure 3.4; in this case, if the gate node is never enabled (event 'p' occurs before event 'n' or does not occur at all, or event 'm' occurs after 'p') before the activity act_i is executed, then the ASN is deadlocked. As a rule, there must be an execution path that can reach the end node of the ASN, whether the gate is enabled or disabled.

3.3.1.2 / Operational deadlocks.

An operational deadlock is one which reflects an incorrect or incomplete description of the operating conditions (event and/or data dependent) of the system and does not reflect any violation of the characteristics of the model. An operational deadlock occurs if:

- a node requires the presence of an external event to trigger but this external event never occurs;

- the event occurs in an incorrect sequencing order or it has not been correctly indicated in the definition of the ASN.

Examine again figure 3.4; in this figure the ASN is deadlocked if the events occur in an improper sequence, like event 'p' occurring before event 'm' or event 'p' before event 'n'; this situation can be considered also as an operational

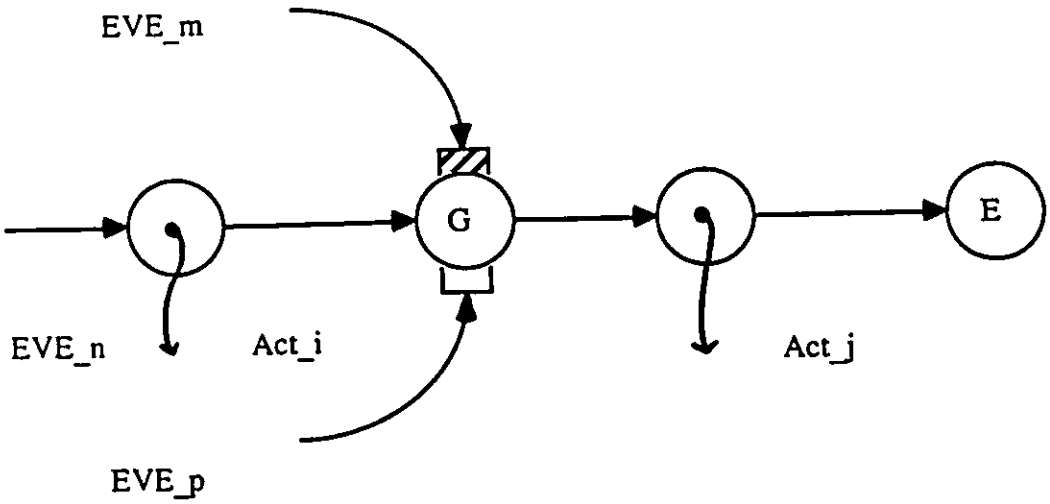


Figure 3.4 Deadlock with a Gate node.

deadlock.

- Another situation where an operational deadlock can occur is when the resulting data creates a condition that was not anticipated in the system behavior.

In the case of an incomplete definition, the problem may or not be detectable during the modeling phase. If it is, such an error can only be detected through a simulation.

3.3.2 / Multiple triggering.

Multiple triggering is the second type of connectivity error that can occur in an ASN. This problem corresponds to the arrival of several spurious triggers to the same input arc of activity, branch, wait and enabled gate nodes with several input triggers, to the same D or E input of a gate and/or timer/counter node and in the case of merge nodes.

The effect of multiple triggering depends thus on the type of node involved and on the timing of occurrence of this condition.

i / In the case of a merge node, the output trigger(s) is set as soon as one input trigger reaches the node; if several triggers reach the merge node, they propagate right through to the output triggers, activating all of them once per spurious trigger.

ii / In the case of the other nodes, if one of the input arcs

to the node is triggered more than once, then depending on the timing arrival of these triggers the following situations arise:

- If all the spurious triggers reach the node before it is activated, then there is no effect and no error occurs, because upon activation all input triggers to the node are reset and the erroneous triggers are cancelled.

- However, if one or more of these triggers reach the node after it has been activated, then the trigger may create subsequent problems; the type of error that can result is multiple activation of the subsequent subprocess.

iii / In the case of spurious triggers reaching the D and/or E input of timer/counter and gate nodes, an error can occur in the following two situations:

- In the case of a timer node, which is enabled repeatedly while the value in the count register is being decremented. When the timer/counter node is enabled, the initial count value is passed to the node and is entered in the counter register; if the E input is triggered during the counting process, the present value of the count is forgotten and the counting process starts from the original value.

- An error arises in both types of nodes if the E input is triggered in the midst of a stream of spurious triggers reaching the D input of the node. This condition results in the non-consideration of the occurrence of the E trigger, as it is cancelled by the next D trigger. Similarly, in the case of the gate node, the same problem occurs if the D input is triggered

during the multiple triggering of the E input of the gate. In this case, the occurrence of the only D trigger is cancelled by the stream of triggers to the E input of the node.

Although in some cases the effects of these spurious triggers are filtered out, these situations must be detected as they may result in inconsistent system behavior. There are three situations which can yield multiple triggering:

- The first case is the incorrect use of parallel constructs; multiple triggering results because the recombination of two or more concurrent paths is done using a merge node. If the concurrent paths have different execution times then the merge node is triggered several times (once per parallel execution path), with intervals corresponding to the difference between execution times of the paths. This situation is represented in figure 3.5.

Assuming different execution times for act_2 and act_3 (act_2 has an execution time shorter than act_3), it appears that the merge node is triggered once by act_2 before act_3 is completed; act_4 is executed a first time with data provided exclusively by act_2, while act_3 is still being executed. The completion of act_3 results in a second execution of act_4.

If act_4 required data from act_3, then it was executed the first time with insufficient data and thus yielded incorrect results.

- The second source of multiple triggering is the self-

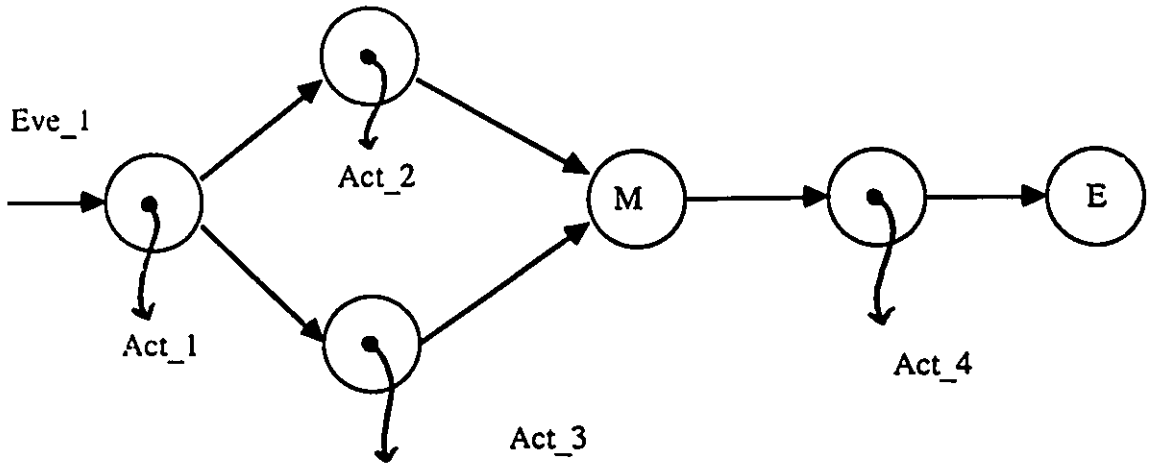


Figure 3.5 Multiple triggering

triggering of merge nodes. As soon as a self-triggering merge node is triggered for the first time, a continuous stream of spurious triggers is supplied to the rest of the process. This condition is detectable by INSPECT.

- The third source of multiple triggering is the incorrect use of feedback constructs in the network. When using feedback constructs, the following rules must be observed:

i / all the execution paths in the subprocess inside the feedback loop must be completed before this subprocess is restarted. In figure 3.6-a, if the arrival rate of the event Eve_2 is larger than the processing rate of the subprocess A, then multiple triggering results in the feedback construct.

ii / the return node of the feedback loop must be a merge node. If the return node is not a merge node, then it results in a deadlock situation.

iii / A feedback loop can also create multiple triggering in situations where subsequent subprocesses are started at the same time that the feedback loop is restarted. This condition may or may not be desirable. Two examples of this condition are illustrated in figures 3.6-b and 3.6-c. In the case of figure 3.6-b, if the state of the condition variable is true, then both subprocesses B and A are started simultaneously. Once subprocess A is completed for a second time, and if the execution time of B is larger than that of A, then multiple triggering results in B. In figure 3.6 b, subprocess B is executed every time the subprocess inside the feedback construct is completed. Again.

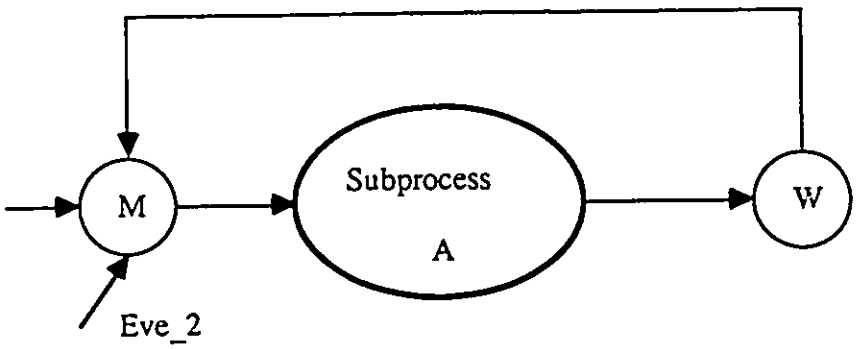


Figure 3.6 a Multiple triggering if
Execution time of A > arrival rate of Eve_2

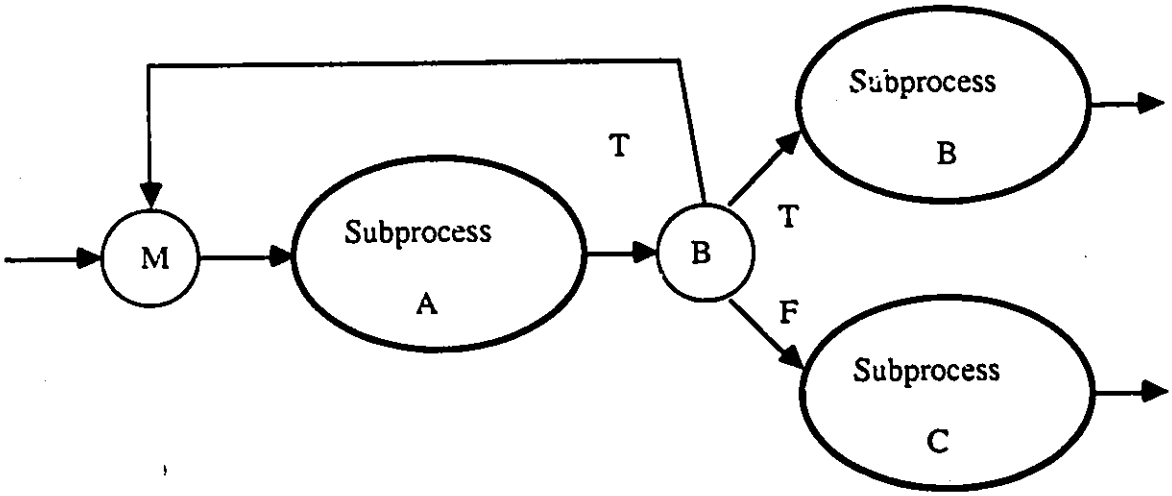


Figure 3.6 b Multiple triggering in
subprocess B.if condition true
and Execution time of A < Execution time of B

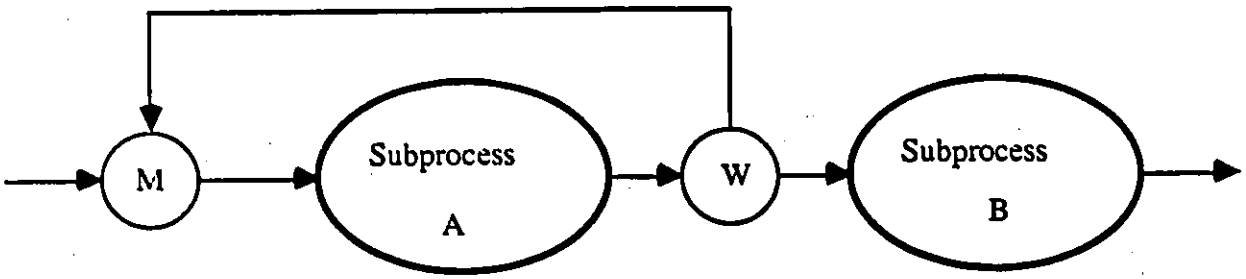


Figure 3.6 c Multiple triggering in
subprocess B.

depending on the execution time of the subprocesses, multiple triggering may or may not occur.

A feedback loop can also worsen a multiple triggering situation in a net if the starting node of a feedback loop is a merge node which is already involved in a multiple triggering situation; in such case, the feedback loop accentuates the problem as it feedbacks these spurious triggers.

3.4 / Verification of connectivity definition.

In this section three approaches for the verification of the connectivity definition of an ASN are presented:

- The first approach is a systematic 'walk-through' of the network which can be considered in a sense as a 'node reachability' analysis. This walk-through can be performed manually or automatically by using the PAD simulator [JOAN 86].

- The second approach is a reachability analysis, which considers the state of the different triggering arcs in the network.

- The last approach is based on the generation of ASN expressions.

From these verification approaches, only the walk through by means of the simulation scheme can be done automatically; the other two are presented as alternatives if the simulator is not

available.

Two points must be mentioned before considering the verification of the connectivity:

- The ASN must be tested by INSPECT, as some errors detectable by INSPECT may be the origin of more complicated connectivity errors.

- In modeling a system using PAD's, several processes (ASN's) may be required to fully describe the behavior of the system. To completely verify the behavior of a system, each ASN and the interaction between the existing processes must be checked.

3.4.1 / Walk-through of the ASN.

This method is based on the 'execution' of the process under each and every condition based scenario to determine if the network is operational. This can be done automatically if a simulator (developed as previous work) is available.

A tree structure is used to illustrate this verification method. In this structure, the *nodes* of the tree represent event states that correspond to the triggering requirements of the various nodes of the network. Therefore, a *node* in the tree corresponds to a column in the ASN matrix and to a node in the network.

The *branches* of the tree represent the output triggering of the nodes. For this analysis, because the interest is only in

the sequencing relations between nodes, 'zero' execution time is considered for the activities (when done manually).

At any time a *current state vector*, corresponding to the present state of the events of the ASN, is considered and compared to the event triggering requirements for the nodes of the ASN; when there is a match between the elements of the current vector and the *event state vector* of a node, then that node triggers.

In the case of concurrent paths, the multiple output construct initiates several parallel *branches*. Each of the concurrent paths is checked; as soon as any branch of the tree can not continue, it stops. The *current event state vector* is temporarily 'frozen' while the other branches (of the tree) are tested, starting from the *event state vector* that created the parallel construct; the remaining *branches* (of the tree) are supposed to supply the missing triggers for any blocked ('frozen') node in the ASN to continue.

The cases of gate and timer/counter nodes are treated similarly to those of branch nodes. The only differences are that these nodes start disabled and that they may be enabled and disabled.

Feedback representations are possible; the current event state vector reaches a state which indicates that a previously reached merge node has been reached once again. The case of the infinite feedback loop is an exception as there is no end node in the ASN.

The tree structure helps to illustrate the two possible errors:

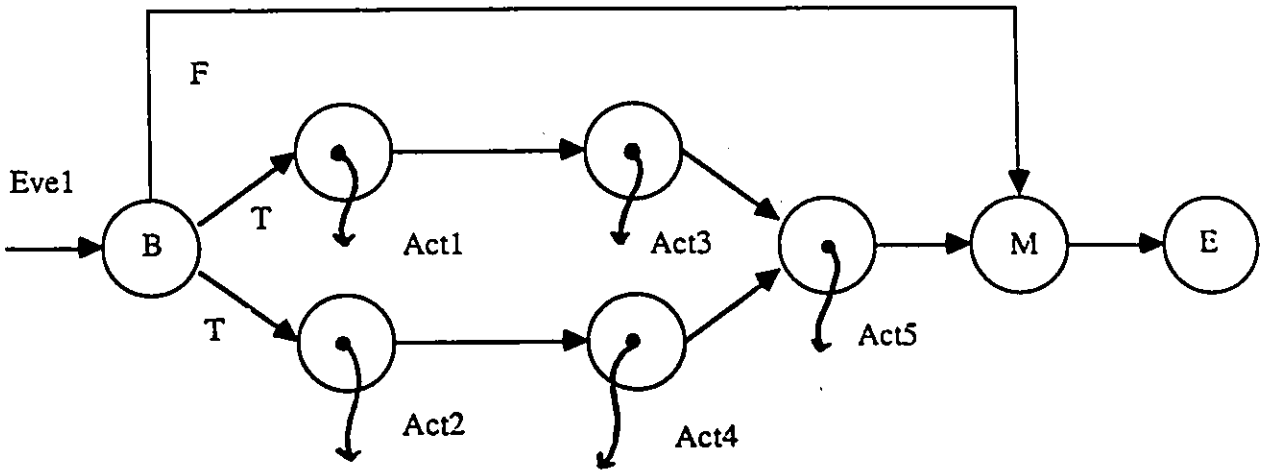
- Unless unconditional feedback loops are present in the ASN, all the *leafs* of the tree must correspond to the end node; a *leaf* is the last element of a *branch* of a tree. Deadlocks are consequently detected if in a tree one or more *leafs* do not correspond to the end node of the ASN.

- If an ASN has 'n' condition variables (including timer and gates) then the resulting tree must have no less than 2^n *leafs*. This means that for each condition based scenario in the network, the tree structure can only have one *leaf;* if there is more than one *leaf* and if no other errors are present, then there is a multiple triggering situation for that particular condition based scenario.

To illustrate this method, three examples are introduced. The first example corresponds to a simple and correct network and is used to show the generation of the tree structure. The other two examples are used to represent both types of errors, deadlock and multiple triggering.

Figure 3.7 shows a simple ASN, its corresponding ASN matrix and the tree structure as derived from the ASN matrix.

Initially, the elements of current the state vector are set to '0'. As soon as the element corresponding to the external starting event in the *event state vector* is set to 'I', the ASN



	Bra	Act1	Act2	Act3	Act4	Act5	Merg	End
BraT	0	1	1	0	0	0	0	0
BraF	0	0	0	0	0	0	1	0
Act1	0	0	0	1	0	0	0	0
Act2	0	0	0	0	1	0	0	0
Act3	0	0	0	0	0	1	0	0
Act4	0	0	0	0	0	1	0	0
Act5	0	0	0	0	0	0	1	0
Merg	0	0	0	0	0	0	0	1
Evel	1	0	0	0	0	0	0	0

Figure 3.7 a ASN and corresponding ASN matrix

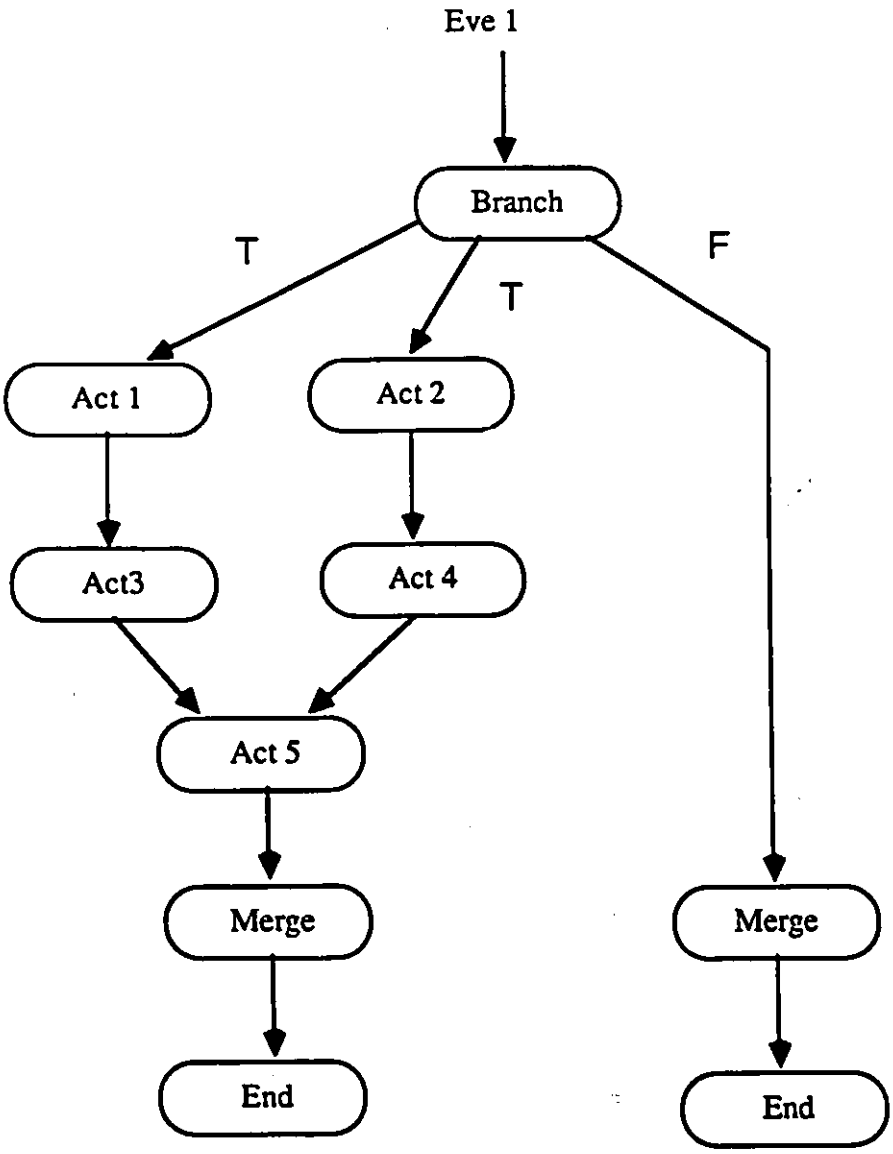


Figure 3.7 b Example of 'Walk through'
of an ASN.

is awakened. The ASN matrix shows that the contents of the *current event state* vector coincide with the requirements for the activity initiator node act_1 to be triggered. The triggering of this node resets the element in the *event state vector* of the external event to '0', and sets the element (in the vector) corresponding to activity node number one to 'I'. This sets the necessary conditions for the branch node to be triggered.

At this point, there are two alternate paths, depending on the condition variable associated with the branch node. Both paths must be checked for, node by node, and up to the end node (if possible). Let's consider the *branch* which corresponds to the true state of the condition variable; in this case the *current state vector* indicates that two concurrent paths can be started and executed simultaneously. Consider the first one; act_1 can be executed because of the state of the event vector. The result of this execution is the activation of the input trigger required for the execution of act_3, which in turn triggers one input arc of the activity node act_5. The other input arc for this node is provided by the second parallel path; when this path is completed, the *current state vector* reaches a state that satisfies the conditions for act_5 to be activated. Upon completion, the merge node and then the end node are reached.

In the case of the condition variable being false, the merge node is immediately triggered, by-passing the subprocess just described.

Consider now the case of the ASN in figure 3.8-a; the

resulting tree structure is found in figure 3.9-a; from this tree structure it can be seen that a deadlock is reached, in both states of the condition variable because the triggering requirements for the wait node can never be satisfied. Both *leafs* in the tree do not represent the end node.

In the case of the ASN in figure 3.8-b, multiple triggering occurs as can be seen in the tree which represents the net, in figure 3.9-b. This tree shows two *leafs* and there are no condition variables (here $n=\emptyset$ and $2^{\emptyset}=1$, so there should be only one *leaf*).

Up till now, the walk through was performed manually. The same results can be obtained by using a modified version of the simulator in an interactive mode. This allows the user to follow through the execution of the ASN and also to obtain the ATC of the ASN for each scenario.

In this case, a deadlock is detected as soon as the network reaches a state from which it cannot move. This can be seen particularly in the resulting ATC; figure 3.10-a represents the ATC of the ASN in figure 3.8-a, assuming there is no deadlock; as can be seen, as soon as the activity act_5 activates the end node, the process goes back to the dormant state. However, in the case of a deadlock, the ASN remains awaken forever, as can be seen in the ATC in figure 3.10-b.

In the case of multiple triggering, this situation is detected when a particular activity is executed more times than

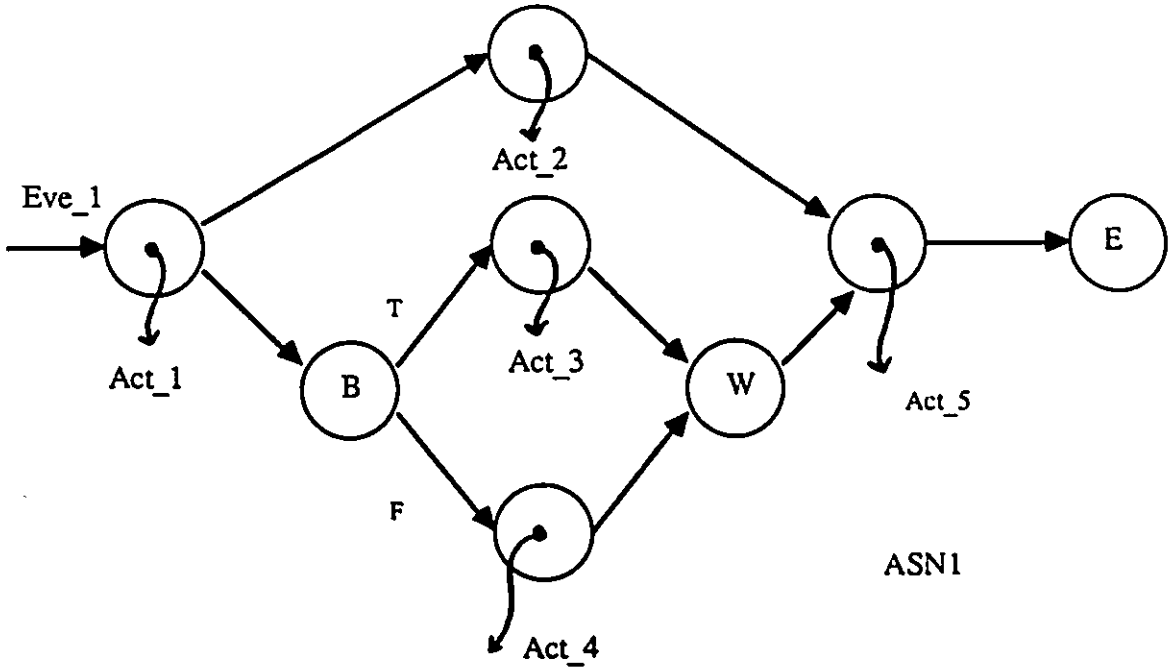


Figure 3.8 a ASN with deadlock for continuous sumulation.

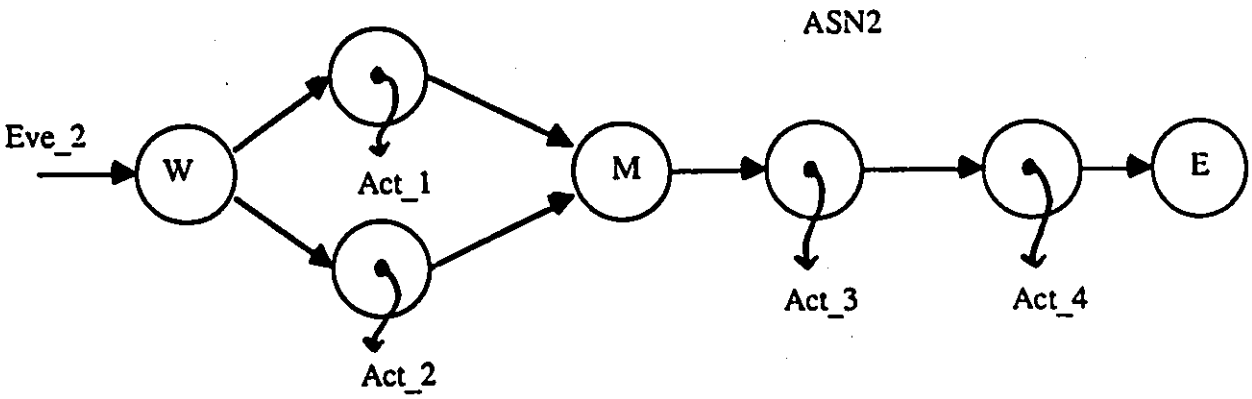


Figure 3.8 b ASN with multiple triggering for continuous simulation.

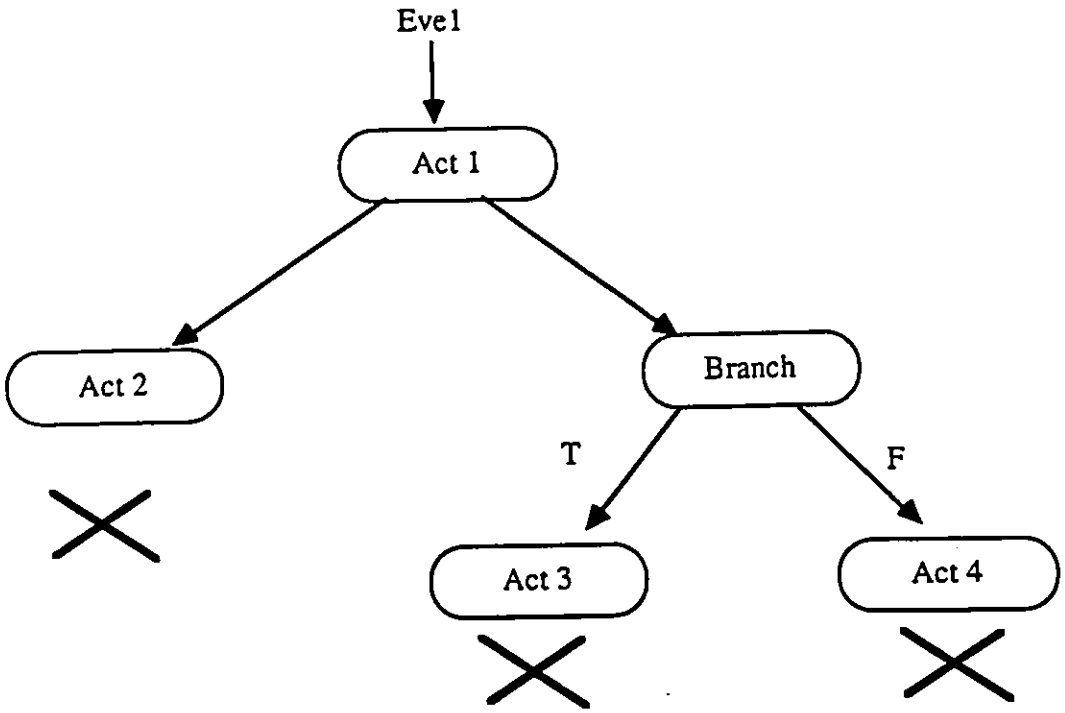
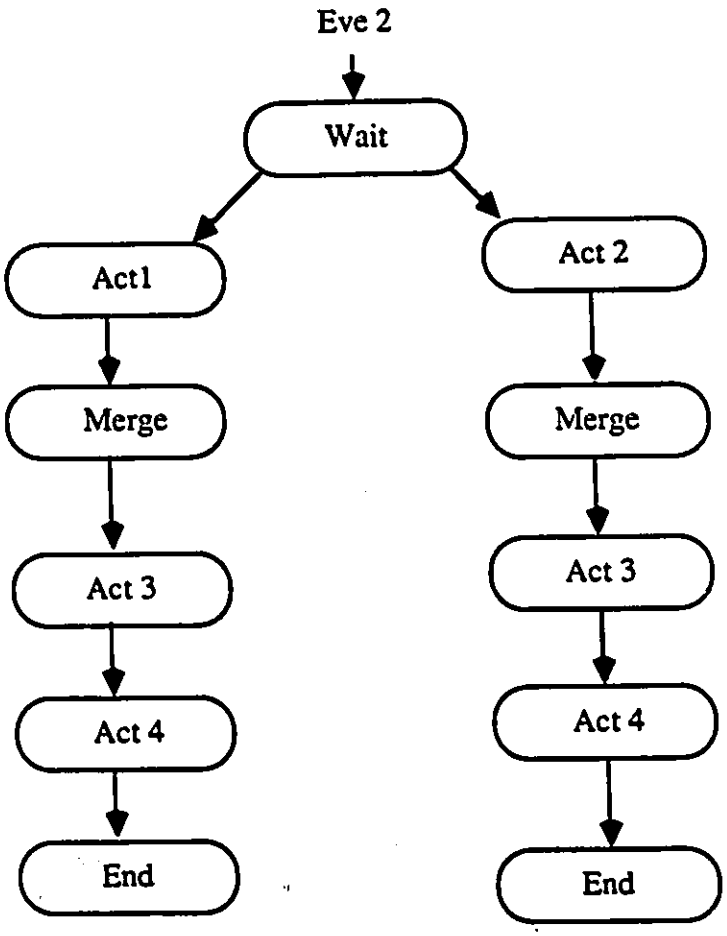


Figure 3.9 a 'Walk-through tree structure of the ASN with deadlock in figure 3.8a.

Figure 3.9 b 'Walk-through structure of the ASN with multiple triggering in figure 3.8 b.

(2 'leafs' appear)



anticipated.

Another possibility is to use the simulator in the continuous mode. In such case some clarifications about the operation of the simulator must be made:

- The simulation yields some statistical results which can be used to detect both types of connectivity errors in an ASN. The detection of these errors is possible considering in particular two characteristics of the simulator:

  - No multiple active copies of an ASN are allowed; while an ASN is awakened, its corresponding starting event is rejected every time it occurs, until the ASN goes back to the dormant state.

  - As soon as the end node of the ASN is reached, the execution of any of the (still executing) activities of the ASN is automatically interrupted.

- To completely check an ASN, all the possible condition based scenarios must be considered. These conditions can be specified prior to the simulation run.

i / Consider first the case of a deadlock; in figure 3.8-a, the ASN has a deadlock because the branch construct is incorrect as the merge node has been replaced by a wait node.

The ASN shown is tested for a large simulation time and comparatively smaller execution times for the activities. The value of these execution times can be seen implicitly in the

Time	Activity # 1	2	3	4	5	State of ASN
1	I					*
2	I					*
3	I					*
4	I					*
5		I	I			*
6		I	I			*
7		I	I			*
8			I			*
9			I			*
10					I	*
11					I	*
12						-
13						-
14						-

- : ASN is dormant.
* : ASN is awaken.

Figure 3.10 a ATC of a correct ASN..

Time	Activity # 1	2	3	4	5	State of ASN
1						
2	I					*
3	I					*
4	I					*
5	I					*
6		I	I			*
7		I	I			*
8		I	I			*
9			I			*
10			I			*
11						*
12						*
13						*
14						*

Figure 3.10 b ATC of an ASN with deadlock..

resulting ATC in figure 3.10-a. The relevant statistics are:

Statistical Results.

- Simulation time: 5000 units.
- Event statistics: Eve_1 occurred 243 times.
  Eve_1 rejected 242 times.
- Network statistics: ASN1 awakened 1 time.
  Completion time 0.0 (the ASN was never completed).

By analyzing these results, and comparing in particular the number of times the starting event of the ASN occurred with the number of times it was rejected, it becomes apparent that only the first event was accepted. The remainder ones were rejected because the ASN was already awaken; this is an indication of a deadlock situation in the ASN.

ii / In the case of multiple triggering, the error can be detected similarly. In the continuous mode, it is just a matter of comparing the number of times the activities should have been executed to the number of times they were actually executed. Given some execution times for the ASN in figure 3.8-b, the statistical results indicate:

Activity execution times:

Act_1 - 5 units; Act_2 - 2 units; Act_3 - 3 units;

Act_4 - 6 units.

Statistical Results.

- Network Statistics: ASN2 awakened 223 times.
- Activity statistics:

Act_1 executed 223 times;

Act_2 executed 223 times;

Act_3 executed 446 times;

Act_4 executed 223 times.

The process yields double execution numbers for activity Act_3 compared to the rest of the ASN. It can be seen that multiple triggering of activity Act_3 has occurred.

3.4.2 / ASN Reachability analysis.

The previous analysis suggests that a reachability analysis, like in Graph Theory, can be used to verify the ASN model. In this section such an analysis is considered.

In Graph Theory, a vertex x_i (a point or a node in a graph) is said to be reachable from a second vertex x_j if there exists a path (arc or sequence of arcs) between x_i and x_j. If there exists a directed arc between the vertices, from x_i to x_j, then x_j is said to be directly reachable; if on the other hand several arcs are involved, x_j is said to be indirectly reachable from x_i [HARA 65], [CHRI 75].

Compared to the previous approach which considers only the events in the network, the blown reachability analysis considers the state of each and every trigger arc (connectivity arc) in the network.

To perform this analysis the following must be done:

i - each connectivity arc in the network must be

independently labeled. The previous analysis considered events; however, an event can include one or several connectivity arcs as an event (internal from a node or external) may trigger several other nodes. If an external event triggers a number *n* of nodes in the ASN, then *n* different triggers have to be considered and represented in the vector. The total number of these arcs in a network becomes in consequence very large if the ASN contains a large number of strongly interconnected nodes.

ii - Similar to the state vector of the previous analysis, a *trigger state vector* is defined. This vector indicates the state of the trigger arcs in the network. For every node in the ASN, there is a vector state that results in the activation of that particular node. This vector which indicates the triggering requirements for a node, contains a '1' in those positions corresponding to the input arcs to the node, and 'don't cares' elsewhere; as soon as the few required triggers are set, execution may continue in the ASN.

The variations including all other possible *trigger state vectors* are part of the reachability analysis. However, a tree structure allows to discard most of these states, and consider only those relevant to the execution of the ASN. In this approach not every *vertex* in the tree structure represents the required triggering state for a node in the ASN, but also transient states that do not activate any node.

This analysis detects deadlocks in the same way as the

previous case, when a *leaf* does not correspond to an end node and multiple triggering, when any parameter in the trigger state vector reaches a value larger than one.

This approach is more analytical and systematic than the previous 'walk-through'; however, it is more complex particularly because of the increase in the size of the state vector, and the need for determining its elements. Because of the complexity of this approach and the fact that this method is considered as an alternative to the one previously described, only a simple case is considered as an example.

Examine the ASN in figure 3.11; in this case the *trigger* state *vector* contains ten elements, labeled from *a* to *j* as indicated in the figure. Initially, the vector contains only '0's; as soon as the trigger arc corresponding to the starting event is set, the branch node is activated. Then depending on the state of the associated condition variable, one of the alternate paths is executed. Considering the case of the true state, trigger arcs *b* and *c* are set; this indicates that two parallel paths can be executed. Consider at this point the case of act_1; when this activity is activated, trigger arc *b* is reset and trigger arc *d* is set. This indicates that act_3 can be activated, as a result of which *d* is reset and *f* set. Nothing further can be activated from here; the other path which can be started in parallel as the result of the true state of the condition variable must be checked now. This path sets at the end the

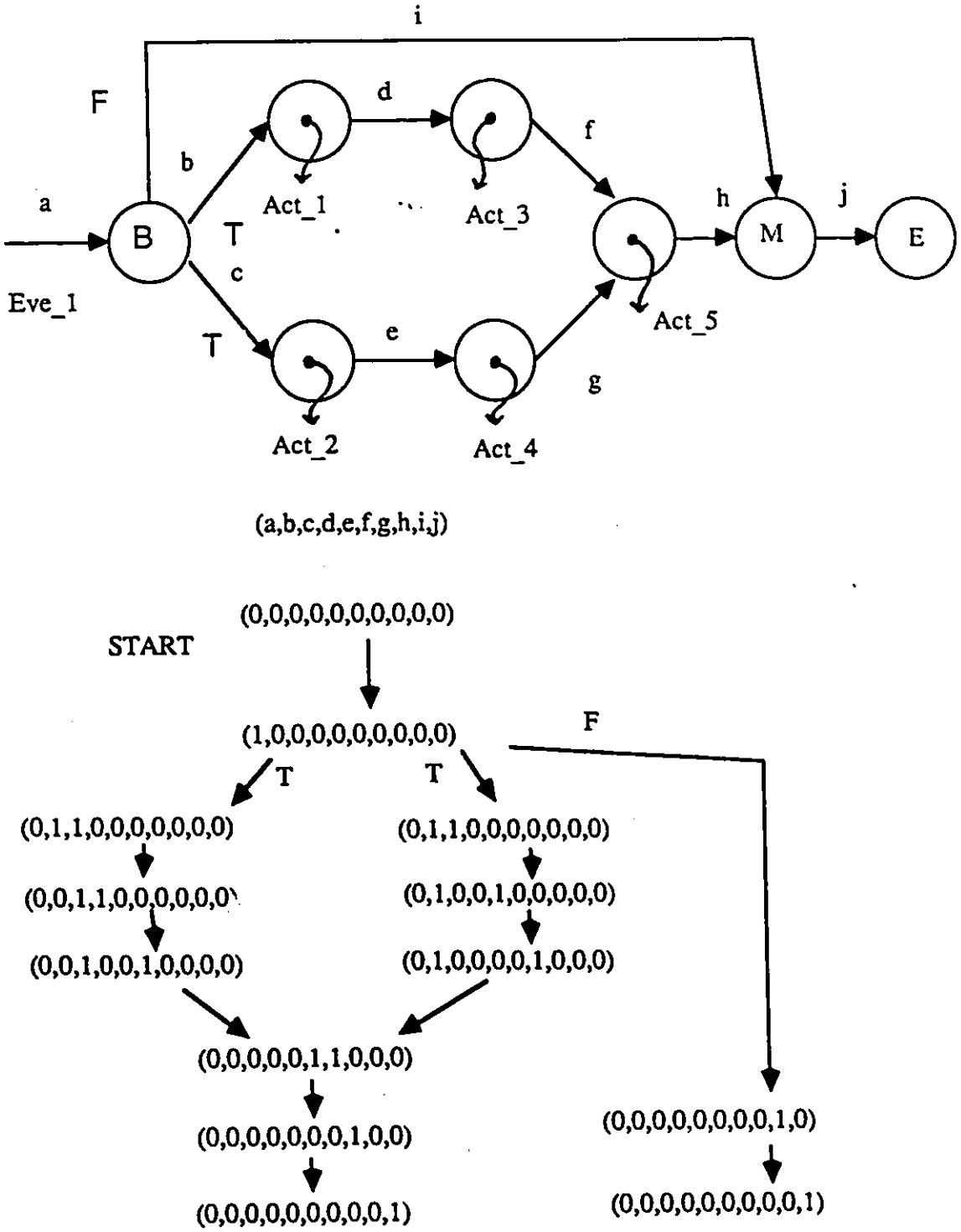


Figure 3.11 Trigger State reachability Analysis.

trigger arc *g*. At this point, both arcs that are required for the next node to be activated are set, but by means of different paths. To recombine (join) them properly, a new vector state must be determined: all the triggers that were reset by either path are reset, and those triggers which are set in either path are untouched unless they were reset by the execution of one of the paths. The resulting trigger vector state indicates that act_5 can be activated. The merge node and then the end node follow, indicating the completion of the execution of the ASN.

3.4.3 / The ASN expressions.

The last verification method deals with the generation of ASN expressions that represent the exact behavior or function of a process. Any process correctly described by means of an ASN can be derived into an error-free expression. These expressions are derived based on some syntax rules; if during the process of forming the expression, a particular rule is not observed, then the corresponding ASN contains an error. The determination of the ASN expression involves the methodic and iterative grouping of subprocesses into composite activities, in order to form a simplified network, simpler to represent in an expression. In these expressions, only activity nodes and end nodes are represented explicitly. The existence of other nodes like branch, merge and wait nodes is shown implicitly by means of the notation.

These expressions are presently obtained by hand and are represented in terms of the following notation:

1 - A '.' indicates a direct sequencing relation between two activity nodes. This operator is not symmetric because the order of execution of the two activities is very important. A sequence of two or more activities can be combined to form a composite activity. For instance, in $Act_i = Act_1.Act_2$, Act_i is the composite activity which is equivalent to the subexpression on the right.

2 - To represent possible concurrent paths, more notations are required. Brackets '{' are used to indicate the start of two parallel paths. If more than two paths are started, then several levels of brackets are needed. Similarly, closing brackets '}' indicate the recombination of two paths, by means of an 'and' input type of node.

The operator '//' indicates that the actions on either side of this operator can be executed in parallel. This operator is symmetric, as $\{Act_1//Act_2\}$ is the same as $\{Act_2//Act_1\}$. A correct parallel construct can be also combined to form a composite activity, for instance:

$Act_j = \{ Act_3 // Act_4 \};$ [3.1]

In this case, Act_j is the composite activity formed by the parallel construct.

3 - Square brackets are used to indicate the start ([) and the end (]) (merge node) of two or more alternate paths; the choice of paths depends on the state of the associated branch condition variable, which is represented at the beginning of each of the alternate paths, enclosed in curve brackets (). The two options are separated by a '/' sign.

If more than two states for the condition variable exist, then the structure of the expression is more complex; as the square brackets represent only the choice between two paths, several brackets have to be used for the expression to remain clear and consistent. For instance, in the following example in figure 3.12 the expression of the subprocess is:

$$Act_1.[(b=1,2)[(b=1)Act_2/(b=2)Act_3].Act_5/(b=3)Act_4]; \quad [3.2]$$

4 - The starting event of the ASN is also represented enclosed in brackets at the beginning of the expression. The requirement of an external event or a set of previous actions from a separate path can be interpreted and consequently represented in a similar way as that of the state of a condition variable of the *branch* node.

At this point it is time to consider an example. Examine the ASN in figure 3.13; the corresponding expression can be derived as follows:

i - The branch construct is correct, as it can be represented in the above notation; the composite activity Act_a can be defined

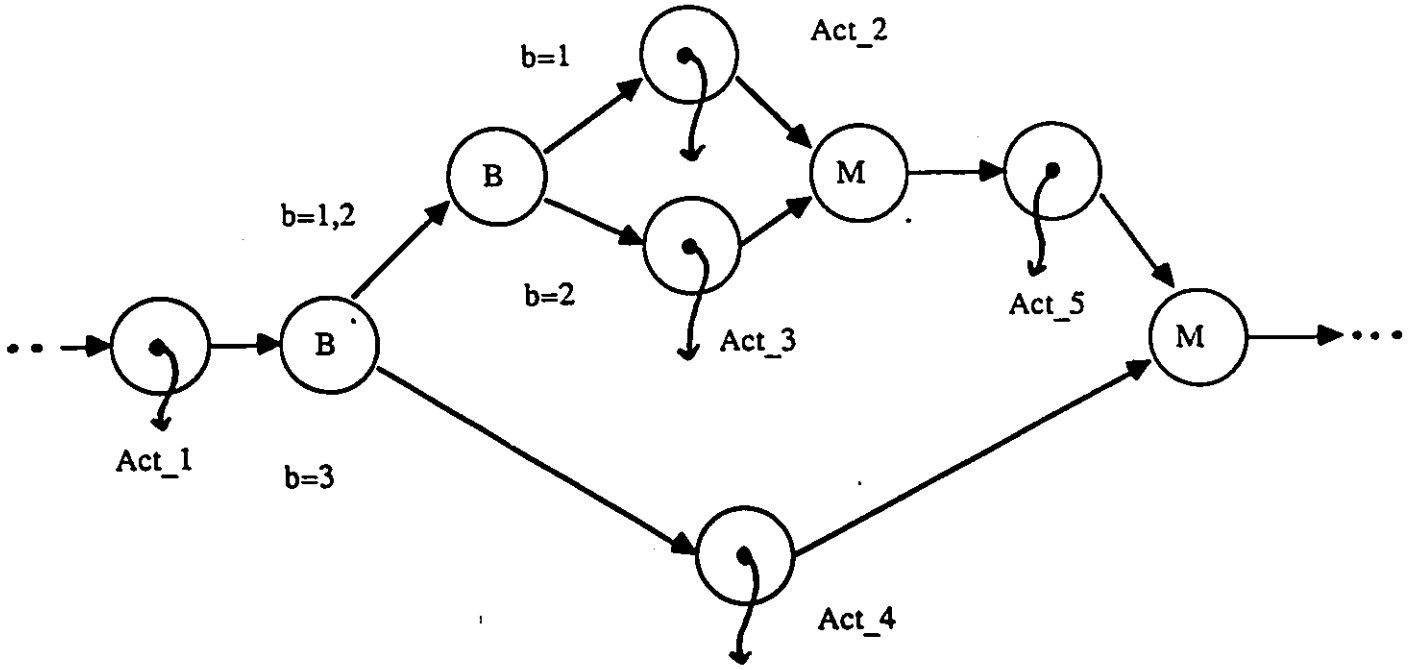


Figure 3.12 ASN for expression [3.2].

as:

Act_a = [(true)Act_2/(false)Act_3]; [3.3]

ii - At this point the path can be combined into a composite activity as it represents a correct sequence of activities. Consequently,

$Act_b = Act_a.Act_5$; [3.4]

The same applies to the other parallel path where Act_c can be correctly defined as

$Act_c = Act_4.Act_6$; [3.5]

iii - Now the resulting network represents a parallel construct; this construct is correct and can be represented by an expression

$Act_d = Act_1.\{Act_b//Act_c\}.Act_7.End$; [3.6]

The complete expression for this network is thus given by:

$Act_1.\{[(true)Act_2/(false)Act_3].Act_5//Act_4.Act_6\}.Act_7.End$; [3.7]

5 - *Gates* and *timer/counter* nodes make no difference, as they can be considered in the same way as the *branch* node.

6 - Feedback paths are represented by stars with an associated number, like '$*_1$'. The number is required in case several feedback paths exist in the process. In the expression, a first star represents the *merge* node of the return loop, and a second star, with the same associated number, is used to show the start of the feedback path. All the sequence of actions within

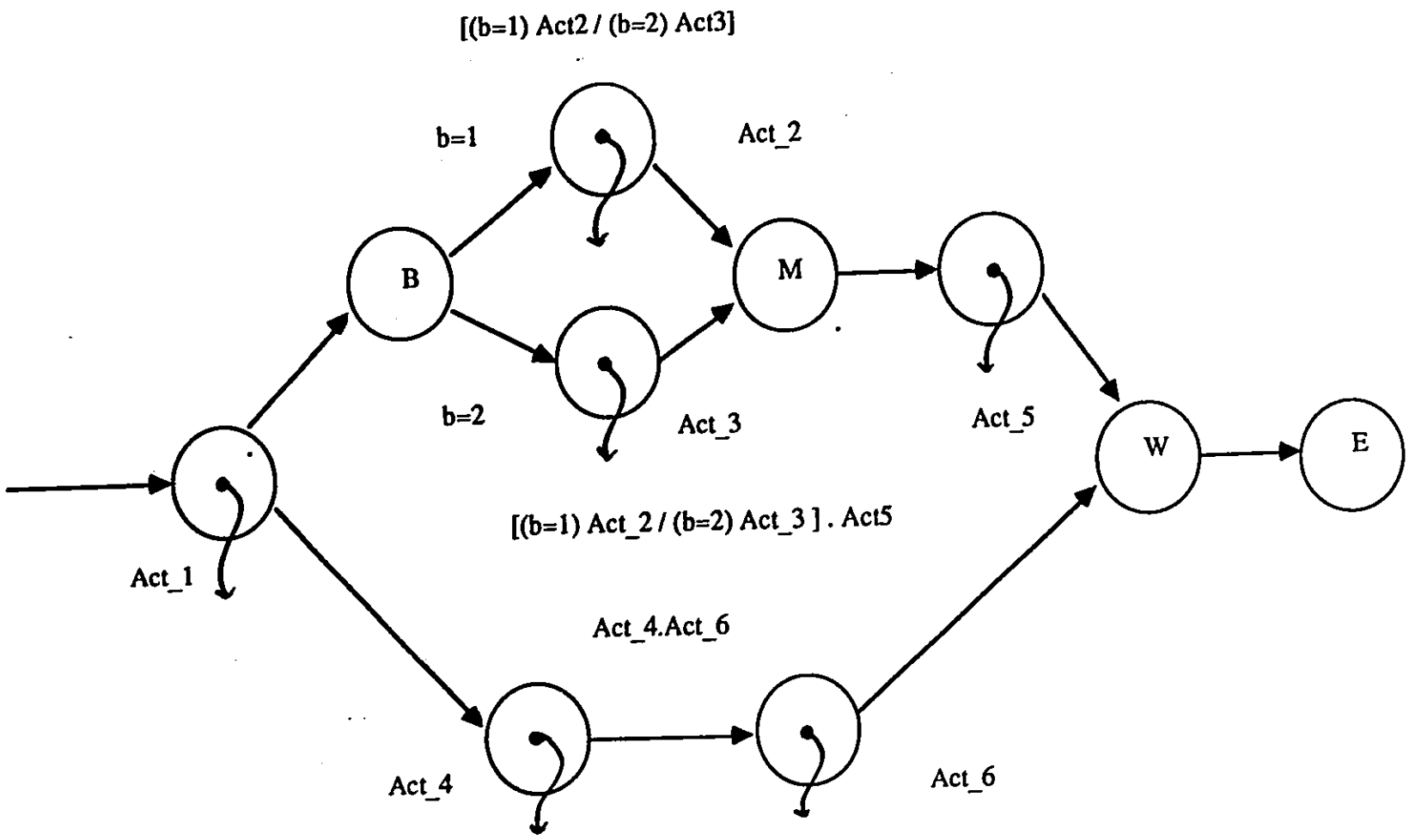


Figure 3.13 ASN used for the generation of an ASN expression.

the feedback loop should form a complete subprocess that must be completed before it can be restarted; otherwise the feedback loop is incorrect.

As an example consider the ASN in figure 3.14; the resulting expression is:

$$(Ex1).(*_1).Act\_1.Act\_2.[(false)Act\_3.*_1/Act\_4.End] \quad [3.8]$$

The simplification step for the derivation of the expression is iterated as long as it is possible to simplify the process. In all, there are three types of sub-processes that can be grouped into a composite activity:

- correct sequences of activities;
- correct and complete parallel constructs;
- correct and complete branch constructs.

If a subprocess can not be replaced by a composite activity because it does not reflect correctly one of the above three types of subprocesses, an error can be detected.

In order to represent the ASN by an expression, it may be necessary to modify the network; this must be achieved however without affecting its function; for instance, as shown in the figure 3.15, the expression does not 'look' like the graph, although it reflects the exact same sequence and timing of actions. In this case the resulting expression is:

$$(Eve_1).Act_1.\{Act_2//Act_3\}.\{Act_4//Act_5\}.Act_6.End; \quad [3.9]$$

In this case, a wait node had to be introduced to show the

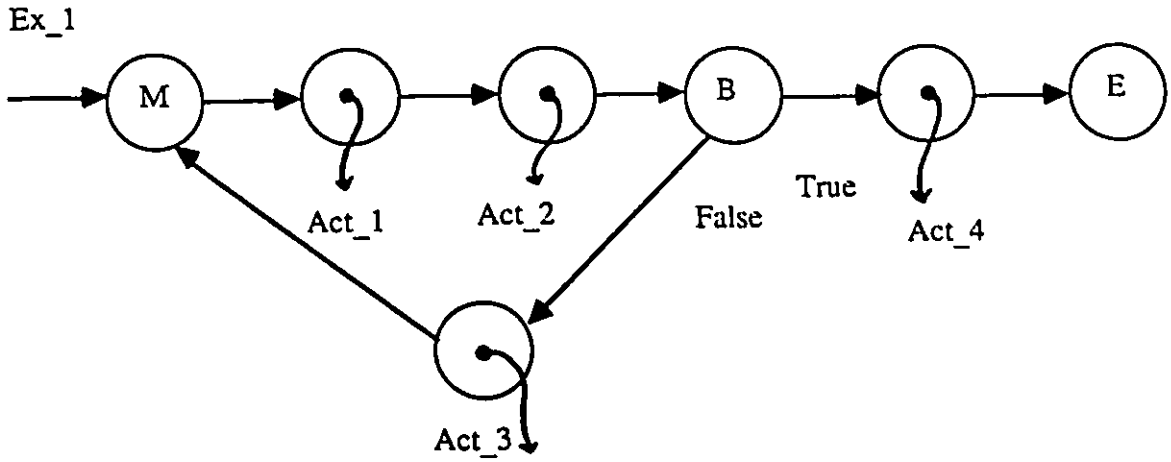


Figure 3.14 ASN for expression with feedback [3.5]

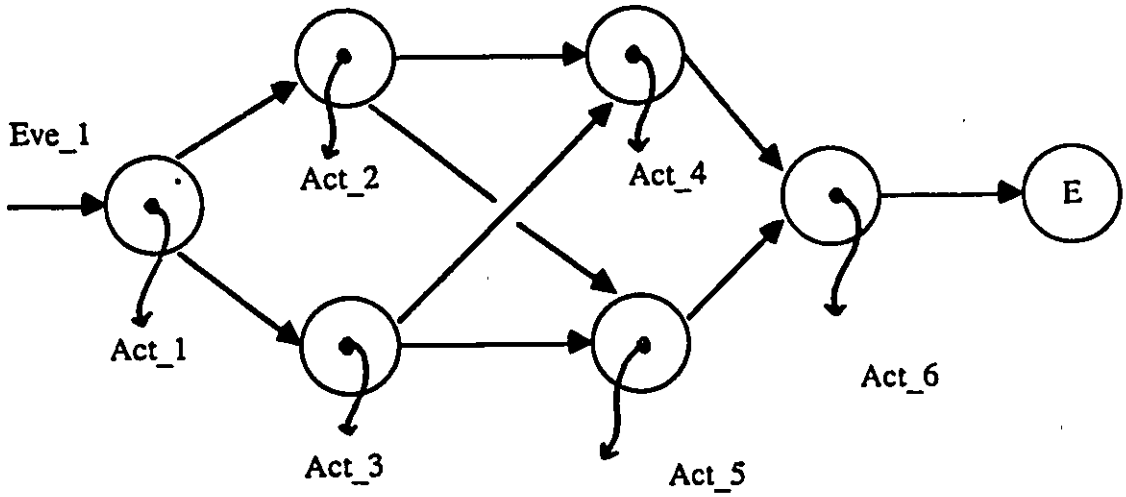


Figure 3.15 ASN of expression [3.6]

function of this process. The required sequencing and concurrency characteristics of the process are in no way modified. A certain amount of expertise is thus required to derive these expressions properly.

To show how errors can be detected when determining the expression of an ASN, consider the case of figure 3.16. In this case, the subprocess can only be reduced by grouping Act_1 and Act_3 into a composite activity Act_i. But it cannot be further reduced at all. The branch construct (including the branch, the merge and activities Act_i and Act_2) cannot be grouped into a composite activity because Act_i triggers a node which is outside of the construct. In this case it can be seen that the problem is a deadlock if the state of the branch condition variable is FALSE.

In the case of an expression with feedback paths, the analysis is more complicated. In the case of unconditional feedback paths, the sub-expression contained between the two '$*_i$' delimiting the feedback loop must be correct; if so it can be collapsed into a composite activity.

In the case of a conditional feedback (not an infinite loop), the 'start star' of the feedback loop must be contained inside a branch construct in one of the states of the condition variable, and the *end* node must be contained in the other condition variable's section of the same branch construct as shown in the expression [3.5].

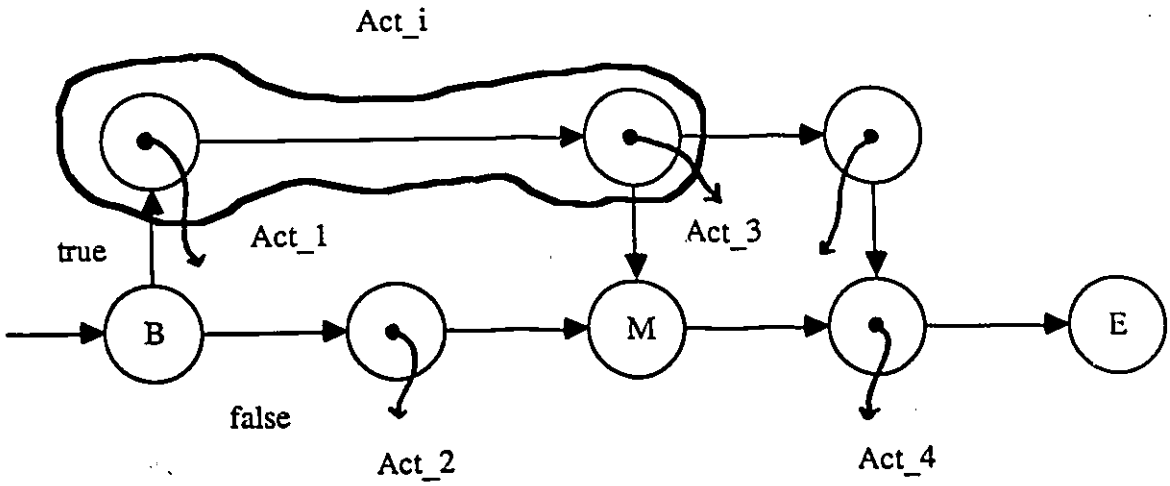


Figure 3.16 Incorrect ASN, with a deadlock when condition false.

3.5 / Concluding remarks.

The purpose of this chapter was to present a verification algorithm for systems defined by means of Process Activity Diagrams.

In the first section, the need for such a verification was presented. In the same section, the different types of errors that can be encountered during the modeling phase of the design process were listed. In the next sections the types of errors that are relevant to the modeling process when using Process Activity Diagrams were considered. The first of these sections dealt with what can be referred to as 'mechanical errors', simple to detect through the ASN matrix. For this purpose a separate program called INSPECT was implemented. In the next section (3.3), the two connectivity errors, deadlocks and multiple triggering, were described and explained. The last section presented three approaches for the verification of the correctness of the connectivity definition of an ASN. The first approach dealt with a 'walk-through' of the ASN matrix, which can be done by means of a modified version of the PAD simulator, to detect both types of problems. This method considers the state of the events (internal and external) in the network, and can in a sense be related to a 'node reachability' analysis. In case the PAD simulator is not available, two other methods can be used

to verify the correctness of the connectivity: a full blown reachability analysis of the network, done manually and considering the state of each and every triggering arc in the network, and a last method, also manual, that uses the generation of the ASN expressions to detect both problems.

A good modeling tool would include many of these verification features at the edition level. In some cases however, this may not prove to be very efficient:

- First the amount of verification during edition is relatively limited to almost 'grammatical' characteristics.

- Second, the flexibility of the modeling tool is restricted; the features of PAD's presented in chapter two could be considered as a basic set for control oriented applications, and depending on the situation, other types of nodes and network characteristics may be defined by the design engineer.

In some cases the violation of some of the rules presented in both chapters two and three may not automatically result in errors. As an example consider the case of an ASN with an unconditional feedback loop, to describe a process that remains active forever performing the sequence of actions in the loop; in such a case the *end* node is not required. A second example is that of the disabling of *timer* and *gate* nodes: there is no need to disable an enabled *gate* node before completing the execution of an ASN. In such a case, the *gate* node may not need to include a 'disable' trigger.

Finally it must be considered that not all possible errors or problems can be found from the verification of the modeling process; the validation step must follow.

CHAPTER IV

PAD's AS AN ACTIVITY TO TASK ALLOCATION TOOL.

4.1 / Introduction.

When designing a real time multitasking system, one of the major decisions involves the partitioning of the various system processes into tasks. The purpose of this chapter is to show how PAD's can be used in this partitioning, considering the characteristics of the operating on which the tasks are running.

To provide the proper background and terminology, the next two sections review the concepts of tasks, processes, real time, multiprocessor and multitasking systems.

Section 4.4 considers the problem of partitioning a system defined in terms of PAD's into tasks. This partitioning is not straight forward for a number of reasons that are presented in this section. As a conclusion to 4.4, some guidelines for the activity to task allocation are drawn.

The last section illustrates two examples of activity to task

allocation when using two different multitasking operating systems, Harmony and VersaDos.

4.2 / Real time distributed systems overview.

Distributed systems are systems in which the processing functions are dispersed among several physical elements [LIEB 85]. Distributed systems are being extensively used in real time applications mainly for the three following reasons:

- A multiprocessor architecture allows parallelism and/or pipelining of execution to improve system performance;
- Reliability and fault tolerance are enhanced by redundancy of the processor's functions and
- System design is modularized.

In distributed systems a number of coexisting processes need to interact and share or compete for resources. The coordination of the various processes is generally provided by a multitasking operating system in either a uniprocessor or multiprocessor environment. Since uniprocessor systems can only provide virtual concurrency, most real time systems are implemented in multiprocessor architecture. In real time embedded systems, since usually a single application program is running continuously, more interaction between the system processes is required.

At this point it becomes important to review the concepts of real time, processes, tasks and multiprocessor architectures.

4.2.1 / Real time systems.

The basic characteristic of real time systems is that they have to interact directly with their environment. These systems require the ability of responding to events that are generated asynchronously by the environment, and initiate and complete the appropriate response by a given time. In more detail, real time systems have two principal characteristics:

- they interact with the environment, where events can be generated asynchronously, periodically or non-periodically, at any time. The important feature about this event generation is that it may be asynchronous to the state of computation of the system. The system must then be responsive enough to be able to handle all these events.

- in real time systems, time is a critical and often limited resource. This means that the response to the event must not only be initiated before a given deadline, but the process corresponding to the response must be completed before a certain prespecified time. For instance, in the case of periodic activation, the response must have been completed before the new event is generated; otherwise, the events backlog, creating unacceptable delays.

4.2.2 / Tasks and processes.

The two terms *task* and *process* are going to be extensively used in this chapter. The purpose of this section is to clarify their meaning, at least in the context in which they are used in this thesis.

Process and *task* have not been uniquely defined in the literature. This applies particularly for the term *process;* this term is used in most cases as a synonym to 'job' and 'program' to indicate the unit of computational execution in multiprogramming systems. But this is not always the case: for instance in VersaDos, Motorola's multitasking operating system, a process is a Supervisor task while a task is simply a User task. Hoare [HOAR 85] considers a process as a 'function which defines a set of events and actions in which the process is prepared to engage'. In other systems, a task and a process stand for the same level of abstraction [LIEB 85]. In [SHAW 74] for instance, a task is defined as a 'sequential process'.

In contrast with this diversification, the term *task* has been uniquely established in the literature as the unit of functional execution in multitasking systems. A task can be described in such context as a sequence of actions that have a single thread of control.

In this thesis the following terminology is used:

- the process corresponds to the description of the response

to an external event. In this context, a process is a well defined sequence of activities represented by an Activity Sequence Network (ASN).

- the task is composed of a set of 'somehow' related activities; these activities may be related by resource requirements, data transfer, etc..

In this context, a task can be differentiated from a process in two ways:

- by the level of abstraction because the process is used for the description of the response while the term task applies more at the implementation level and

- by the fact that during implementation, a process is generally composed of several interacting tasks.

4.2.3 / Multiprocessor systems.

There are two extreme architectural models in multiprocessor systems:

1 - tightly coupled systems, which are characterized by the use of a shared memory to which all processing units are connected and by means of which all interprocessor communication is done. In such a case, the shared memory must be protected by a memory management mechanism that guarantees its safe access to all executable entities (tasks and/or processes). In these systems, the memory access represents a potential bottleneck, and the larger the number of processors in the configuration, the

larger the bottleneck.

2 - loosely coupled systems are systems where each processor includes a local memory and interprocessor communication is achieved exclusively via an interconnection network. The main characteristic of these systems is that communication in the network is done by using well defined protocols. These protocols introduce an overhead in order to provide protection, error detection, source and destination identification, routing, etc. [TANE 81]; this overhead is unacceptable in the case of most real time systems.

In between these two extremes are what can be called the moderately coupled systems; these are systems where communication is achieved either via local communication buses or via some shared memory. The different units in the system may have a multiprocessor architecture. In moderately coupled systems, the amount of processor interaction is relatively high and the physical distance between the processing units is relatively small. These systems are also usually referred to as distributed systems.

This thesis considers mainly moderately coupled systems because they are better suited to provide design modularity and can be adapted more easily to the system specifications of embedded real time systems.

4.3 / Multitasking systems overview.

The purpose of this section is to review some of the basic concepts of multitasking systems. Emphasis will be placed on those areas that are particularly relevant to the specification of tasks using PAD's.

When describing a multitasking operating system a number of basic functions have to be considered: synchronization and communication mechanisms, task management, time and memory management, etc.. For the purpose of the activity to task allocation, only task management and intertask communication and synchronization are of interest.

4.3.1 / Task management mechanisms.

A task in a multitasking system can be considered as a 'living entity' which moves from one state to another. Different multitasking operating systems consider different sets of task states; the set can be simple like the one of Intel's iRMX86 (which has five states) [INTE 84] or very complex and diversified like the one supported by Motorola's VersaDos (which has twelve states). Figure 4.1 represents a generic task state diagram and it includes the following five states:

- dormant, which is the initial state of the task when it is created;

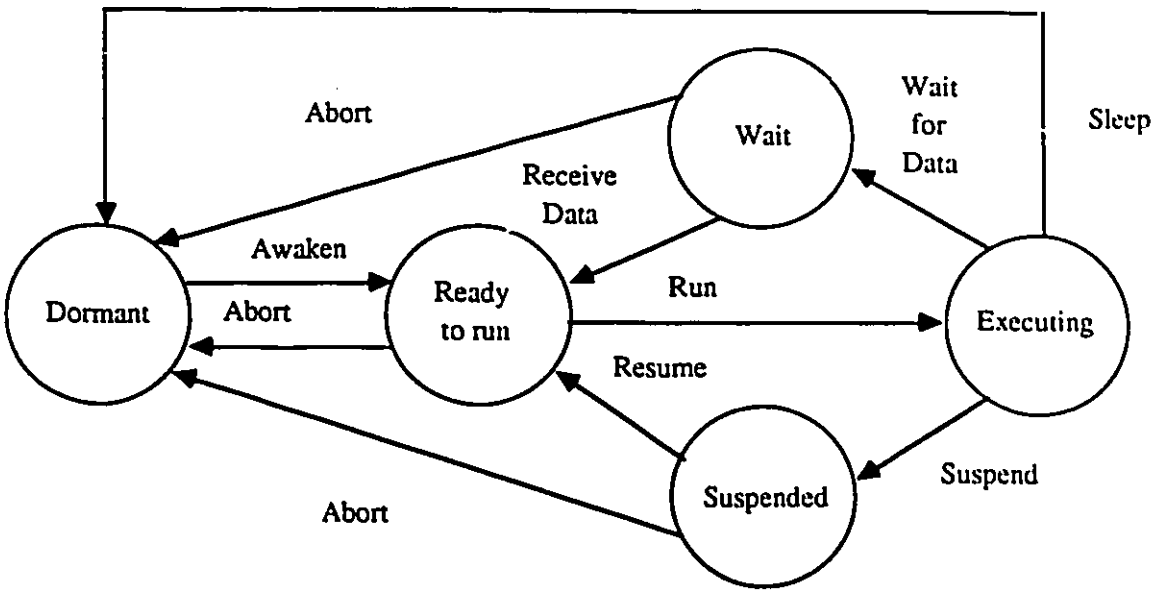


Figure 4.1 Generic Task State Diagram.

- ready to run, indicating that the task can be scheduled for execution;

- active, executing or running;

- waiting, which indicates that the task is waiting for synchronization, resource allocation, delays, etc.;

- blocked or suspended, which is a condition created by an action from the operating system.

Tasks in such context can be moved from one state to another by using operating system primitives. To keep track of these states, each task requires a descriptor to determine precisely all the features necessary to describe this task. This control information can have various names like Task Control Block (TCB) in VersaDos, Descriptor in Harmony, Process Control Block (PCB), etc.. These are data structures that completely indicate all the attributes of a task. Included in these descriptors are:

- memory location of the task's code (logical address);

- task identity;

- copy of all the active registers used by the task;

- data fields in memory associated with the task;

- attributes of the task, like particular resources, priority (fixed or variable);

- etc..

Most operating systems include in the TCB, the id of a task which has to be informed when the task described by the TCB is destroyed. This task is called the *parent, father, monitor,* etc.

of the task.

Some operating systems use a hierarchical type of relationship which means that the creating task is automatically designated as the *father* of the created task. In other systems, the creating task can specify which task is to be the *monitor* of the created task.

The are two types of task creation and deletion mechanisms used in multitasking operating systems: static and dynamic.

i - In the case of static task management, all tasks that are required in the system's operation are defined prior to run time and task deletion is not considered, except in the case of exception handling. In this case, at the time of the system's startup all tasks are in the ready-to-run state. In this case the associated overhead involves the management of the tables and TCB's of the tasks.

ii - The second scheme is dynamic, and in this case tasks are created (and destroyed) as they are needed during run time. When a task is created all of its attributes have to be specified. This requires between 20 to 512 bytes of information, depending on the characteristics of the operating system. This indicates that an automatic overhead results during run time from the dynamic creation of tasks. In addition, a variable overhead is introduced with each task in the system because of different tables required for task management. Note that not all tasks can be created dynamically; some tasks, like server tasks acting as

resource managers, are created initially.

A major overhead is introduced when deleting tasks dynamically, because of the specific conditions that have to be checked before the task can be properly deleted. There are a number of features to consider when deleting tasks dynamically:

- There must be restrictions as to when a task can be deleted. Consider the case of a task being deleted while it is executing a critical region which is protected by a semaphore; that critical region remains inaccessible to all other tasks in the system. Another example is the situation when a task is blocked waiting for a message from a task that has been deleted, or the case of a task which is deleted while blocked waiting for the acknowledgement of the message (reply) from a second task.

- In some systems when a (parent) task is deleted all of its 'childs' or sub-tasks are automatically deleted as well. In some situations this could create considerable problems because the childs may not have completed execution, or they may be in one of the states mentioned above, etc..

For instance, consider the case of a task *Task_1* which is the parent of a task *Task_2*, and a third task *Task_3* which is the child of *Task_2*; consider a hierarchical task organization and finally assume now the following operation: *Task_1* is waiting to receive an event from both *Task_2* and *Task_3*; as soon as it gets an event, *Task_1* destroys the source. If *Task_2* sends the event first, then both *Task_2* and *Task_3* are deleted, and consequently, *Task_1* is deadlocked as it waits indefinitely for the event from

Task_3.

These conditions yield inconsistent and incorrect results and system behavior. In all, there are a number of issues to be considered and there are several possibilities to their handling [CASH 80].

From this brief discussion, it can be seen that in order to allow a safe deletion of tasks, a lot of management and checking of the state of the task must be performed. This is one reason why most systems do not consider dynamic task deletion, and tasks are implemented in an infinite loop, or are always kept in a ready-to-run state (when inactive).

4.3.2 / Intertask communication and synchronization.

The purpose of this section is to list the basic intertask communication and synchronization mechanisms.

Two tasks synchronize when one of them waits until the other reaches a point where they can both be allowed to continue executing. This synchronization may be required in order to share correctly a resource, to exchange data (communicate), etc..

Consequently, there are two types of communication mechanisms, synchronous where both the source and the destination must reach an agreement (synchronize via some sort of protocol-like mechanism) to pass or exchange data and asynchronous, where the sending task does not wait for the destination task to synchronize before sending the information.

In this section, the intertask communication and synchronization mechanisms are reviewed under two groups, procedural and message based, as proposed in [CASH 80].

4.3.2.1 / Procedural based mechanisms.

Procedural based systems involve the controlled use of a critical region. In this section the following mechanisms are reviewed: semaphores, monitors, communicating sequential processes and the Ada rendezvous.

- The semaphore was proposed initially by E.W. Djikstra [BENA 82]. A semaphore is a low level construct which provides safe access to a shared resource or section of code called critical region. A semaphore is an integer variable which is accessed exclusively by primitives for handling semaphores. These primitives issued by the process are:

- *Wait* or P(s), to enter the critical region protected by the semaphore (decrements the value associated with the semaphore by one) and

- *Signal* or V(s), used when the process exits the critical region (incrementing the value by one).

A semaphore has a value associated with it; if this value is 0 or 1, then it is called a *binary* (full-empty or busy-free) semaphore. If the value associated with the semaphore is an integer, then it is called a *counting* semaphore. In this case, a

positive value indicates the number of free entities (tasks or processes) that can still enter the critical region safely, while if the value is negative, it represents the number of tasks which are waiting to access the critical region protected by the semaphore.

The P(s) primitive tests the semaphore 's' and puts it to 'Ø' if its present value is '1'. Otherwise the primitive puts the name of the process on a waiting list associated with the semaphore and then stops it. An invocation of the primitive V(s) by a task exiting the critical region, would restart the blocked process from the waiting list and reset the value of the semaphore to its original value.

The integrity of the values of the semaphore are guaranteed by the indivisibility of the primitives P and S. If used properly, a semaphore can guarantee the safe access to the critical region it protects.

- A second mechanism is the monitor, which was proposed by [HOAR 74] and by [HANS 75]. The monitor stands at a higher level of abstraction than the semaphore and is theoretically simpler to use. The monitor includes within itself the critical region it protects. As such it can be considered as a generalized critical region, in which a shared resource may be manipulated by different processes according to some prestablished scheduling rules [BOWE 8Ø].

Because the critical region is inside the monitor, the safe

access to the critical region is guaranteed.

- The third mechanism to be considered are the communicating sequential processes (CSP), presented in [HOAR 78]. In this case, *Input* and *Output* call-like primitives are used for communication.

Interprocess communication is established by having the sender specify the destination of the *Output* operation and having the receiver specify the source of the *Input* operation. This is a bidirectional communication mechanism.

CSP are relevant because they represent for the first time the concept of a 'rendezvous'-like mechanism between two communicating entities.

- The last mechanism considered is the rendezvous. The rendezvous is primarily used in Ada task communication and can be considered as well as a variation of the messaging model.

A task that wishes to communicate with a second task issues a *Call* to an entry procedure of the receiving task. The caller then waits until an *Accept* of that entry procedure is executed by the receiving task; at that particular moment the rendezvous takes place. If the receiving task issues the Accept before the caller issues the Call, then it is the receiver which waits. Notice that the Accept statement can include a piece of code to be executed before the caller task is released.

The caller (receiver) is idle for the lapse of time between

the call (accept) to the entry procedure and the accept (call) statement by the receiver (sender).

For further references on these communication and synchronization mechanisms refer to [BOWE 80], [BENA 82], [SHAW 74], [JANS 85], [SAIB 85], [CASH 80] and [MADN 74].

4.3.2.2 / Message based mechanisms.

These mechanisms consist of primitives that are used to send and receive messages for communication and/or synchronization. Several messaging models can be considered, depending on the type of primitives available. The message itself can also have several forms, depending on whether pointers or actual data are transmitted as the message.

The most common model consists of two primitives, *non-blocking send* and *receive*. If a blocking *send* is used, a third type of primitive (*reply*) is necessary to unblock the sender. The primitive *receive* is normally blocking.

The reason for having two types of send primitives is that in the case of the blocking mechanism, synchronization is achieved before the communication starts (synchronous), while in the case of non-blocking, this synchronization is not required (asynchronous).

If both primitives (send and receive) are blocking, the communication mechanism is very similar to the Ada rendezvous.

Other primitives may be added to these in order to make the model more versatile; for instance, *try_receive* allows the receiver not to be blocked when receiving a message. In other cases, the sender is able to specify whether it is ready to wait for a reply or not, or the receiver may be able to reply as soon as the message is received (*receive_and_reply*), before the contents of the message are read.

The use of blocking communication primitives may be questionable in the case of real time systems because of the resulting synchronization delays.

Communication via mailboxes can be considered as a variation of the messaging model. A mailbox is a prespecified memory location used by two or more processes to exchange information. A mailbox has to be created before it can be used.

When a mailbox is created, a number of features are specified: the location in memory and the identification of the mailbox, the list of processes that may access it and what are their respective privileges (read and/or write capability), etc..

When a task sends a message to a second task it accesses the appropriate mailbox and deposits the message or a pointer to it in the mailbox. The sender may be blocked or not depending on whether the receiving task is ready to receive or not. In some cases, the sender specifies if it wishes to wait. If the sender is blocked, it will be unblocked by the reception of the message or by an acknowledgement from the receiver. The receiver

accesses the mailbox and may also be blocked if the message has not arrived yet.

The messaging model can be used in either a tightly coupled or distributed system, while the above procedural mechanisms cannot be implemented directly in loosely coupled systems because the processor need to share some memory.

Actually, in multiprocessor systems these mechanisms for intertask communication (whether procedural or message based) are essentially the same; the fundamental difference with uniprocessors is in the way these primitive calls are implemented and carried out by the corresponding communication manager of the operating system. In fact the primitives should be transparent to the user as to whether the system is implemented in a uniprocessor or in a multiprocessor architecture.

The communication manager of the multitasking operating system must be able to determine, during compilation (static task to processor allocation and static task creation) or during run-time, where the target task resides in order to establish the communication connection.

The directive must be interpreted as a function call whose execution depends on whether an inter or an intra-processor exchange is involved. In the first case, the directive must use the system's interprocessor communication mechanisms; bus contention and bottlenecks, transmission rates, shared memory access, communication protocols, etc. are to be considered in the

expected overhead. The resulting overhead cost associated with an inter-processor communication is thus much larger that in the case of an intra-processor exchange.

4.4 / Activity to task allocation.

When implementing a system in a multitasking environment, one of the major design decisions involves the specification of the tasks. A good partitioning of the system into tasks can enhance the system responsiveness and improve the overall performance. The purpose of this section is to show how Process Activity Diagrams can help in this design step.

When partitioning a system into tasks a number of issues must be considered: whether a uniprocessor or multiprocessor system is used, if it is time sliced, priority based, preemptive, etc.. To consider all possible combinations would be self defeating; consequently in this section the following three aspects are considered:

- Effect of a uniprocessor organization;
- Effect of a multiprocessor organization;
- The effect of the type of intertask communication and synchronization mechanisms used.

In the following discussions three assumptions are made:

- The system must be able to handle N_i (larger than one) events of the same type of response simultaneously.

- The incoming events are detected by an event handler, which can be considered as interrupt driven. This event handler interprets the occurrence of an event as an interrupt, during which it determines the corresponding type of response.

- Tasks do not all have the same priority.

4.4.1 / Task partitioning in a uniprocessor organization.

The fundamental characteristic of a uniprocessor organization is obviously that only virtual parallelism, by means of time sharing, is possible. In consequence, the execution of system processes and tasks must be appropriately interleaved in order to satisfy the timing requirements of the responses.

In this context, because there is no point in taking advantage of the concurrent characteristics of the activities in the ASN's, each process 'i' is organized as a sequence of L_i consecutive tasks. The node receiving the starting event of the process is allocated to the first task in the sequence, while the end node of the ASN must be included in the last task in the sequence. The activities in the process are allocated to the tasks mainly depending on protection of data and resources; in addition, the amount of interactivity transfer, task management and scheduling mechanisms of the operating system have to be considered for the partitioning of tasks. Assume hence that there are L_i consecutive tasks for a given process 'i'.

To be able to meet real time requirements in a non-time

sliced system, tasks should be defined as short as possible, whether preemption is used or not. If task preemption is used, although the tasks are active more often, there is an increased overhead caused by the context switches. On the other hand and in general, systems which are not time sliced and do not consider task preemption are not responsive enough for event driven real time applications.

In time sliced systems there is the overhead caused by the periodical context switches. This overhead varies between the operating systems, but in general is in the order of a few percentage points of the time slice. In uniprocessor systems it is an overhead that the designer has to live with.

The next question concerns the type of task creation mechanism that the operating system supports: static or dynamic.

If task creation is static, then N_i copies of each task must be created initially. This results in a total of $N_i x L_i$ tasks to be created per process. This allows the 'simultaneous' processing of up to N_i responses. However, if the execution of the responses can be pipelined, this upper limit can be extended to a theoretical maximum of N_i times L_i responses.

If the system supports dynamic task creation and deletion, there are two alternatives for the task management:

- In the first case a *response manager* task is created upon event detection by the event handler. This manager is responsible for the creation of all the tasks in the sequence, involved in the execution of the process. The response manager

starts then the execution of the first task. When a task completes, it provides the necessary information for the next task in the sequence to start. When the last task in the sequence finishes it execution, it informs the response manager which is responsible for the deletion of all the tasks it created for this particular response. In this configuration, only N_i responses can be processed simultaneously.

- In the second scheme only the first task of the sequence is directly created by the event handler as the result of the occurrence of the starting event. The subsequent tasks in the sequence are created when the preceding task in the sequence completes its execution; they are destroyed when the next task starts executing and in accordance with on the intertask relationship of the operating system. This process of task creation and deletion must be done carefully and coherently to avoid deadlocks. The last task in the sequence informs the event handler of the completion of the response.

It must be realized however, that not all the tasks in the system (like for instance tasks that act as managers of a resource) can be created and destroyed dynamically.

4.4.2 / Task partitioning in a multiprocessor organization.

True parallel execution is only possible in a multiprocessor organization; therefore in this case there is no immediate need to use time slicing. For that reason only priority based

preemption is considered in this discussion.

In a multiprocessor architecture, the activity to task partitioning should be done in a way to take maximum advantage of the concurrent characteristics presented by the activities in the ASN's. However, this must be done taking into account the following:

- Activities that require the access to a shared resource should be declared as a joint task; this task is considered as a server task that acts as the resource manager. This also guarantees the safe access to the resource.

- Two activities with a large information transfer should be allocated to the same task, or at least in two tasks but in the same processor. This is in order to minimize the costlier interprocessor communication, compared to intraprocessor communication.

- If dynamic task creation and deletion are used avoid when possible the relatively larger overhead resulting from dynamically creating tasks in remote processors.

4.4.3 / Effect of Intertask communication and synchronization on the task partitioning.

When allocating activities to tasks, the amount of inter-activity transfer of information has to be considered. The amount of intertask information transfer is given by the sum of the information transfers of the connectivity arcs between

activities that were cut when partitioning the processes. The purpose then is to minimize intertask and interprocessor communication by allocating when possible activities with large data transfers in the same task. In such context, a minimum cut algorithm could be considered.

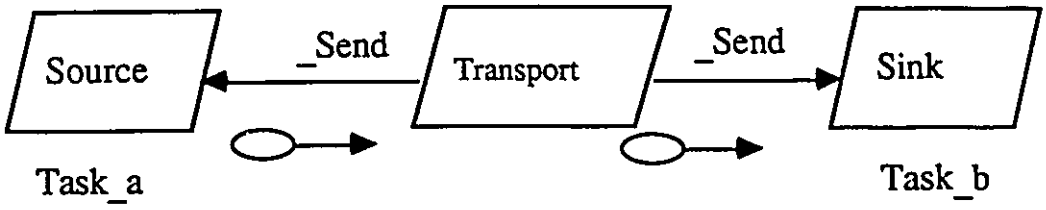
Further on, the partitioning of tasks is affected by the type of communication mechanism available. The information passing type of mechanism implicitly represented by the connectivity arcs in an ASN is a *non-blocking_send* and *receive*. As a result, if asynchronous type of mechanisms are available, then the definition of the tasks in an ASN is in most cases simpler since it can be directly mapped to the node connectivity in the ASN. However, other particularities, such as inter-activity information transfer and resource requirements may still have to be considered.

If only synchronous (blocking) communication is available, then the sending task must 'wait' for an acknowledgement from the target task; to avoid this a *Courrier* or *Transport* [BUHR 84] task is used to do the waiting, while the 'sending task' continues its execution normally. This is possible because of the modularized structure of the task which has been defined in terms of activities. The task sends a message to the *Courrier* after executing an activity, is almost immediately unblocked, and continues executing the next activity. In such context, message passing is performed only between activity executions.

A *Courrier* task is a subordinate task used by two tasks to communicate when one of the tasks does not need to synchronize, but simply to pass the required signal or information. A *Courrier* task can be implemented in two ways, *receive - blocking_send* construct or *blocking_send - blocking_send* construct. This second construct is called a *Transport* task.

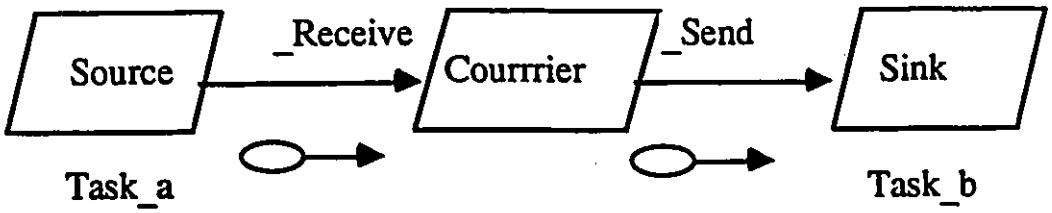
The two options are represented in figure 4.2. In the first case the sending task transmits to the Courrier task; this task then acknowledges the information (reply), unblocking the sender. The task Courrier then waits until the target task is ready to receive, while the sender is free to resume its normal execution. In the second case, the Transport waits for the sender to transmit; the information is passed in this case in the reply, which is also used to unblock the Transport. At the receiving end, it is the same thing as in the previous case. The Transport task is often preferred because it can break a deadlock cycle, like tasks which are in a 'deadly embrace'.

Let's consider the examples of figure 4.3 a and b; in the case of 4.3a, if a synchronous communication mechanism is available, then there is no problem as the situation requires *Task_a* and *Task_b* to synchronize; this situation represents in fact a rendezvous. If only an asynchronous mechanism is available, then the rendezvous would have to be 'simulated' by using the non-blocking *send*, like




```
TASK_A                  TRANSPORT               TASK_B
  •                                               •
  •                                               •
Receive (Transport );   Send (Task_a);          Receive (Transport);
Reply (Transport );     Send (Task_b);          Reply (Transport);
  •                                               •
  •                                               •
```

Figure 4.2a Send-Send Courrier (Transport)




```
TASK_A                  COURRIER                TASK_B
  •                                               •
  •                                               •
Send (Courrier);        Receive(Task_a);        Receive(Courrier);
  •                     Reply(Task_a);          Reply (Courrier);
  •                     Send (Task_b);            •
                                                  •
```

Figure 4.2b Receive-Send Courrier.

```
Task_a;                          Task_b;
    -                                -
    -                                -
Execute_Activity_1;              Execute_Activity_2;
Send (Task_b);                   Send (Task_a);
Receive (Task_b);                Receive (Task_a);
Execute_Activity_3;              Execute_Activity_4;
    -                                -
    -                                -
```

On the other hand, in the case of figure 4.3b, there is no need for task *Task_a* to synchronize with *Task_b*. As a result, if only an asynchronous mechanism is available, the task *Task_a* sends to *Task_b* and proceeds executing activity Act_2 without waiting for a reply from *Task_b*; if *Task_b* reaches this point first, then it has to wait on the *receive* primitive until *Task_a* sends.

In the case of a synchronous mechanism, to allow *Task_a* not to be blocked waiting for *Task_b*, a Courrier or Transport task may be used.

4.4.5 / Activity to task allocation guidelines.

The purpose in this section is to present some general recommendations and guidelines for the activity to task allocation of systems defined in terms of activity diagrams.

1 - If the system has a uniprocessor architecture, there is no point in specifying concurrent tasks. Each process should be partitioned into tasks in such a way as to define consecutive

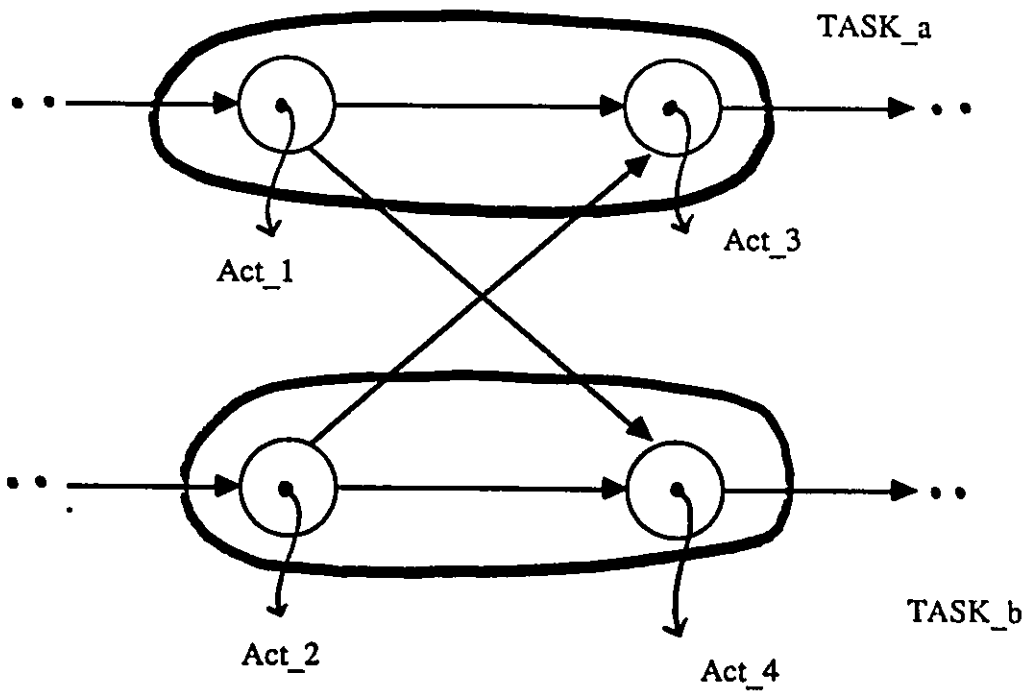


Figure 4.3 a First case with synchronization.

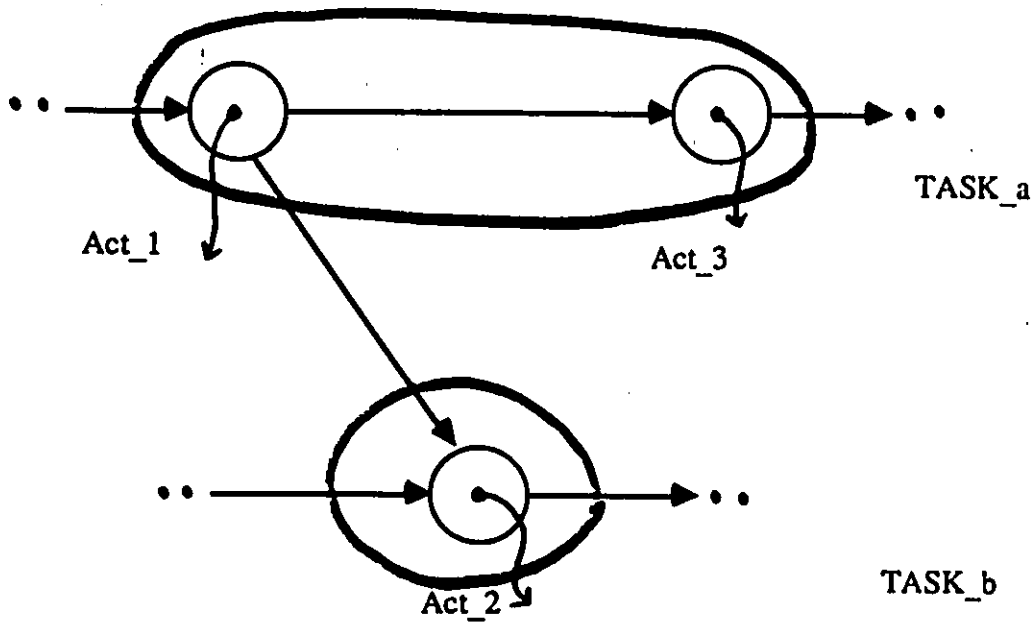


Figure 4.3 b Example with no synchronization

tasks; these tasks could be eventually pipelined to allow multiple event handling.

2 - If the system is a multiprocessor based, then one should take advantage as much as possible of the parallel characteristics described in the ASN's. This must be done considering a number of restrictions, like not to include in the same task, two activities that must be executed by different processors, do the partitioning as to minimize interprocessor data transfers compared to lower cost intra processor transfers and, if dynamic task creation is considered, avoid when possible the dynamic creation of tasks in remote processors.

3 - From a system point of view, define tasks that are general to all the processes and that require the use of specific resources. For instance if similar activities are present in several processes (ASN's), define them as a single task; Activities that access specific resources should be defined allocated to a same task. In such case, this resulting task is defined as the resource manager (server); this provides also mutually exclusive access to the resource.

4 - In terms of intertask communication, it becomes more economical to allocate activities with large inter-activity data transfer in the same task. This becomes particularly significant in a multiprocessor system; avoid as much as possible the

allocation of two activities with a large inter-activity transfer to two separate tasks that are assigned to different processors. A minimum cut algorithm can help in the partitioning.

If the operating system provides only a synchronous type of communication, whether it uses mailboxes or messaging, then *Courrier* or *Transport* tasks must specified whenever there is the need of intertask communication and/or synchronization. Otherwise the intertask communication can be almost directly mapped from the triggering of the ASN.

5 - Once those specific tasks have been defined in the system level, the allocation of activities to tasks should be done on a per process basis. In each process try to define a main task; this task is the one called by the event manager when the starting event occurs; this task is responsible for creating all other sub-tasks necessary for the completion of the response and for the signalling back to the event manager that the response has been completed. This main task should include at least the first and last nodes of the ASN. This approach is particularly simpler in the case of dynamic task creation and deletion.

As an exception to this, if multiple event handling is required, one way to deal with it is not to define a main task per process and define more granular tasks; the reason for this is to allow the 'pipelining' of the responses to the events through the 'network' simultaneously. In such case 'response descriptors' are necessary to keep track of the execution of the

process responses.

6 - Timer and gate nodes should be implemented as separate tasks. A *timer* task would be implemented as a watchdog, running independently of other tasks in the process. Once enabled, the task is continuously waiting for input counting triggers; when the count has been completed, the task provides an output signal. In such case, this task could take advantage of features of the operating system that have built-in timers.

A *gate* task would have associated with it a mailbox-like data structure, with a depth of one, where the message 'enable' or 'disable' is going to be placed (with overwriting capability) by other event sources in the process. Only when the gate task is triggered, is this mailbox read.

7 - The last feature to consider is the determination of priorities for the tasks. For a more efficient scheduling and overall system timing performance, distribution of priorities is to be considered during the partitioning, and depending on the system's scheduling policies and priority assignment to the processes in the system. Basically, there are two rules to follow: the partitioning of tasks should be done on a per process basis, activities of very different priority level should not be allocated together, unless they (the activities) share a common resource.

4.5 / Examples of Multitasking system specification.

The purpose in this section is to show the applicability of Process Activity Diagrams for the specification of the tasks and their interaction in a multitasking environment. This applicability is shown by considering two operating systems, Harmony and VersaDos, whose relevant characteristics are listed prior to the activity to task allocation examples.

4.5.1 / Harmony operating system.

Harmony [GENT 83] is a multitasking operating system, written in C language and implementable in a multiprocessor environment. Harmony is specifically designed and is entirely devoted to running a multitasking application program.

4.5.1.1 / Relevant characteristics of Harmony.

In Harmony, a task is defined as the unit of sequential and synchronous execution and as the unit of resource ownership.

A task in Harmony creates a second task by calls to the _Create primitive. Task creation is dynamic; a task template, specifying the principal characteristics of the task to be created, must exist. A typical template would be for instance:

{type_i, Task_j, Sts, N, P};

where *Task_j* stands for the name of this instantiation of the task *type_i;* the task type represents in fact a pointer to the task's code (or root). *Sts* is the stack size (used for variable storage), *N* is the priority (which is fixed for scheduling purposes) associated with this task, and *P* which is the LOCAL_TASK_MANAGER, or the processor executing this task. Harmony does not support task migration.

The functions _My_id and _Father_id are used to specify the id of the task itself and that of the creating task. These are often necessary for communication and destruction or deletion purposes. The father-child relationship in Harmony is hierarchical.

In order to create a task, a message is sent to the Local Task Manager, who gets the resources to be attached to the child's (the task being created) queue. In the parameter field of the _Create call, the user must specify the name of the task instantiation (GLOBAL_INDEX). The parameters of the task are found in the task template, as specified in the beginning of the main program.

In Harmony a task can be deleted in two ways:

- The task can destroy itself by calling the _Suicide primitive; a _Suicide call is usually used for exception handling.

- A task can also be deleted if another task calls _Destroy(id), id being the name of the task to be destroyed. All the descendant tasks (direct and indirect) of id will

automatically be destroyed as well. When the task is terminated, all its resources are automatically liberated (presently, the only resource available in Harmony is memory).

In terms of intertask communication, Harmony uses four primitives to provide a message passing communication model. A message is defined as a set of contiguous blocks of storage with variable length. These four primitives are:

- _Send, which is used to send messages, and which is of the type *blocking_send;*

- _Receive, which is also blocking; a task can be specified also to be waiting to receive some specific messages (_Receive_Specific_Block);

- _Try_receive which is a non-blocking variation of the above _Receive and

- _Reply, which is a non-blocking *send* used for acknowledgement and un-blocking of *send* primitives.

From the above, it can be determined that a task in Harmony can be in one of the following states: Running, Send_Block, Receive_Block, Receive_specific_Block, Wait and Non_existant. See figure 4.4 for the Harmony state Diagram.

4.5.1.2 / Activity to task partitioning in Harmony.

At this point, the relevant characteristics of Harmony that

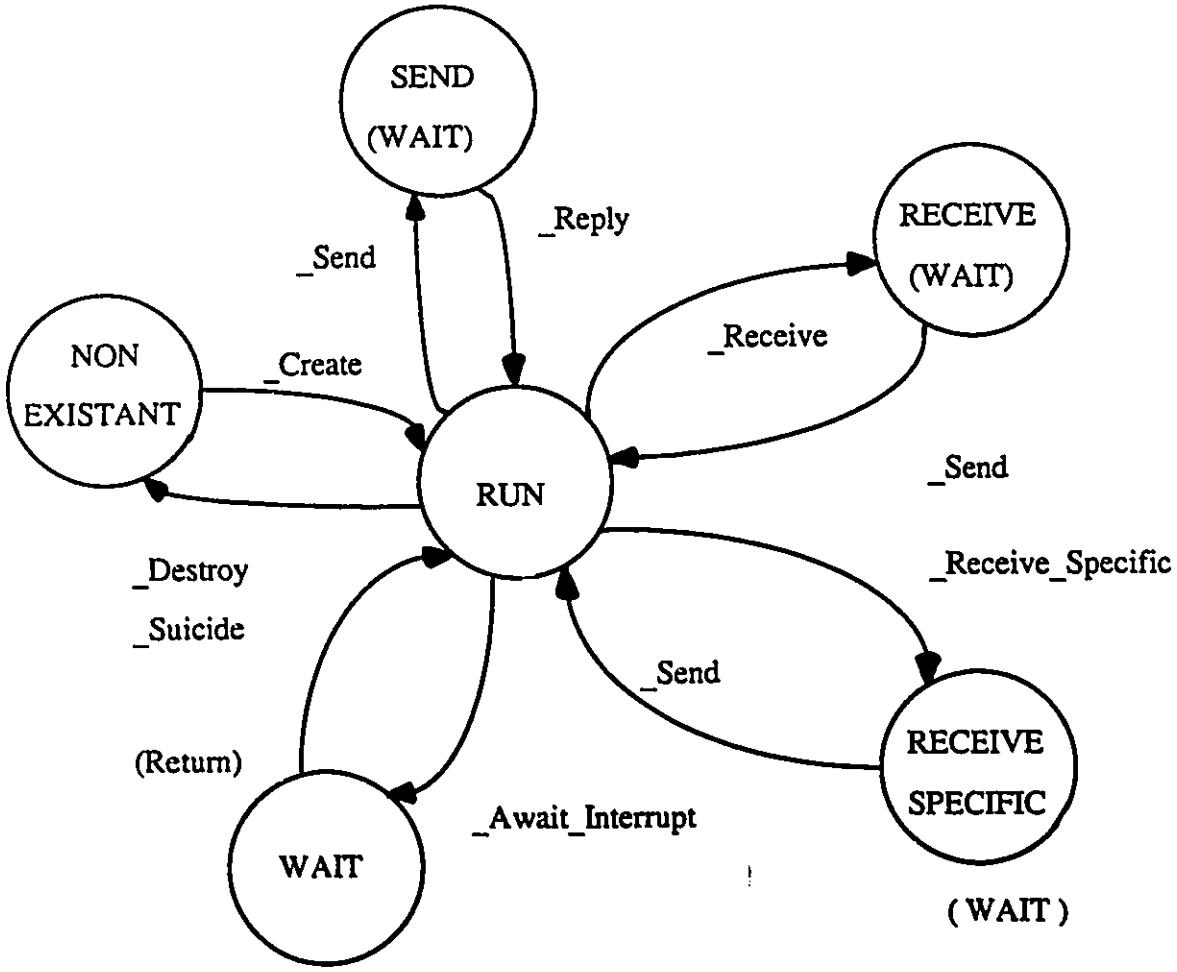


Figure 4.4 Harmony State Diagram.

affect the partitioning of activities into tasks have been reviewed. It is now time to see how they can be applied to an actual partitioning problem; several network examples will be considered.

Consider the case of figure 4.5; in this case, it is possible to define three tasks, *Task_1* of activities 1, 2 and 5, *Task_2* and *Task_3* of activities 3 and 4 respectively. Assuming three processors are available and in terms of Harmony pseudo code, this could be seen as:

```
unsigned_Pnumber=P_1;
struct TASK_TEMPLATE_Template_list[] =
{
     { Directory, _Directory, 268, 7, P_1}        /* P_1 = 0. */
     { Gossip, _Gossip, 226, 7, P_1}
     { 1, Main, Sts_1, N_1, P_1}     /* Corresponds to Task_1.*/
     {0,0,0,0,0}
};

unsigned_Pnumber=P_2;
struct TASK_TEMPLATE_Template_list[] =
{
     { Task_2, Child_1, Sts_2, N_2, P_2}
     {0,0,0,0,0}
};

unsigned_Pnumber=P_3;
struct TASK_TEMPLATE_Template_list[] =
{
     { Task_3, Child_2, Sts_3, N_3, P_3}
     {0,0,0,0,0}
};

Main()
{
     Execute_Activity_1;     /* Definition of data_1, rply_1.*/
                             /* Definition of rply_2.*/
     Child_1=_Create (Task_2);
     Child_2=_Create (Task_3);
     _Send ( data_1, rply_1, Child_1);
     _Send ( data_1, rply_2, Child_2);
     Execute_Activity_2; /* Definition of data_2 and Data_3. */
     _Receive (data_2, Child_1);
     _Receive (data_3, Child_2);
```

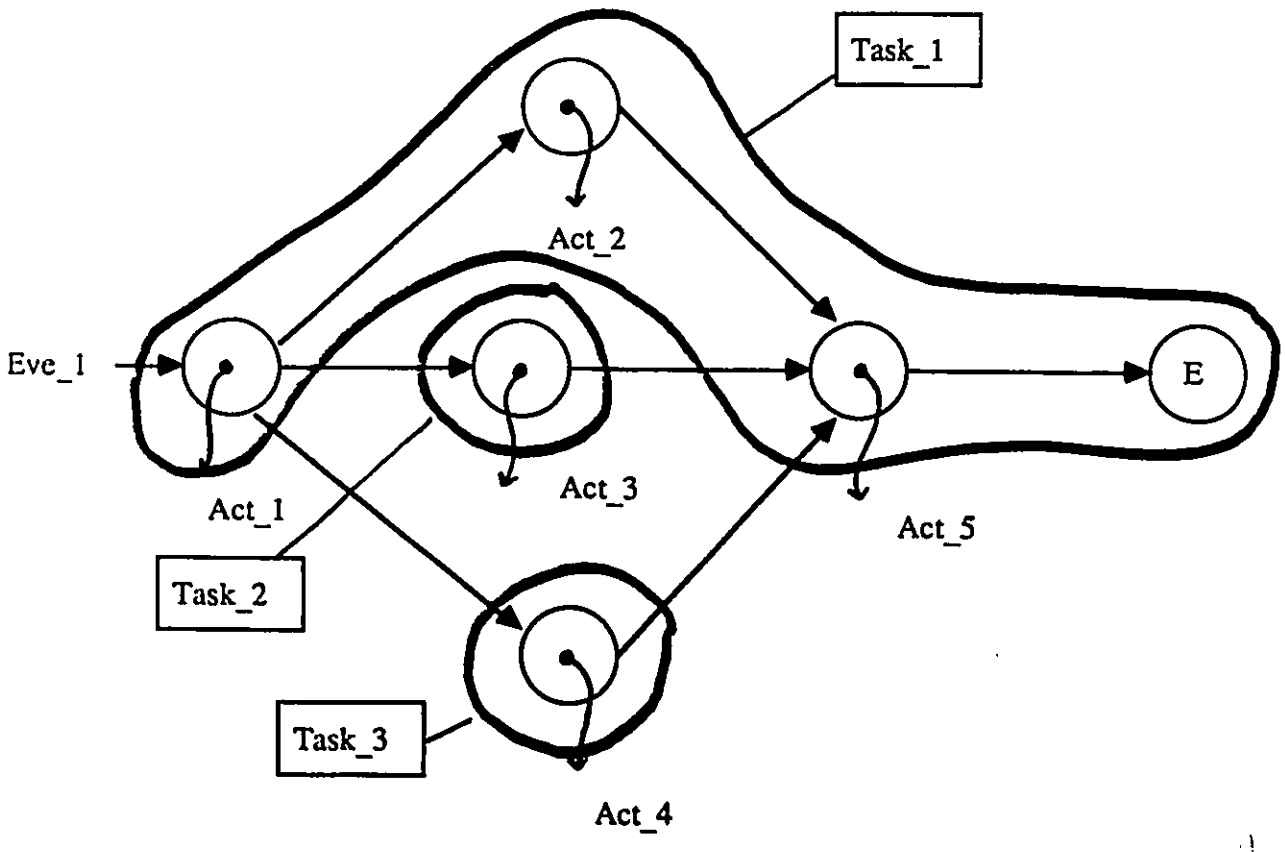


Figure 4.5 First Example of activity to task allocation

```
     _Reply (rply_1, Child_1);
     _Reply (rply_2, Child_2);
/* It does not make much difference to use here     */
/* a blocking or non-blocking receive.              */
     _Destroy ( Child_1 );
     _Destroy ( Child_2 );
     Execute_Activity_5;
};

Child_1()
{
     _Receive ( data_1, Main);
     _Reply (rply_1, Main);
     Execute_Activity_3;  /* Definition of rply_3. */
     _Send ( data_2, rply_3, Main);
};

Child_2()
{
     _Receive ( data_1, Main);
     _Reply (rply_2, Main);
     Execute_Activity_4;  /* Definition of rply_4. */
     _Send ( data_3, rply_4, Main);
};
```

The task *Main* is called by a principal program, the event manager, responsible for the detection of events and the calling of the corresponding process responses.

If (in order to allow multiple event handling by 'pipelining' requests in the ASN and using 'event response descriptors') it is decided not to follow the above rule of defining a main task in the process, then the task management becomes more complicated.

In the above example three tasks were specified to allow maximum possible parallel execution. Keeping this in mind a second possible partitioning could be one defining three tasks, by having the following allocation, as in figure 4.6:

Task_1': activities 1 and 2;

Task_2': activities 3, 5, 6 and the End node;

Task_3': activity 4;

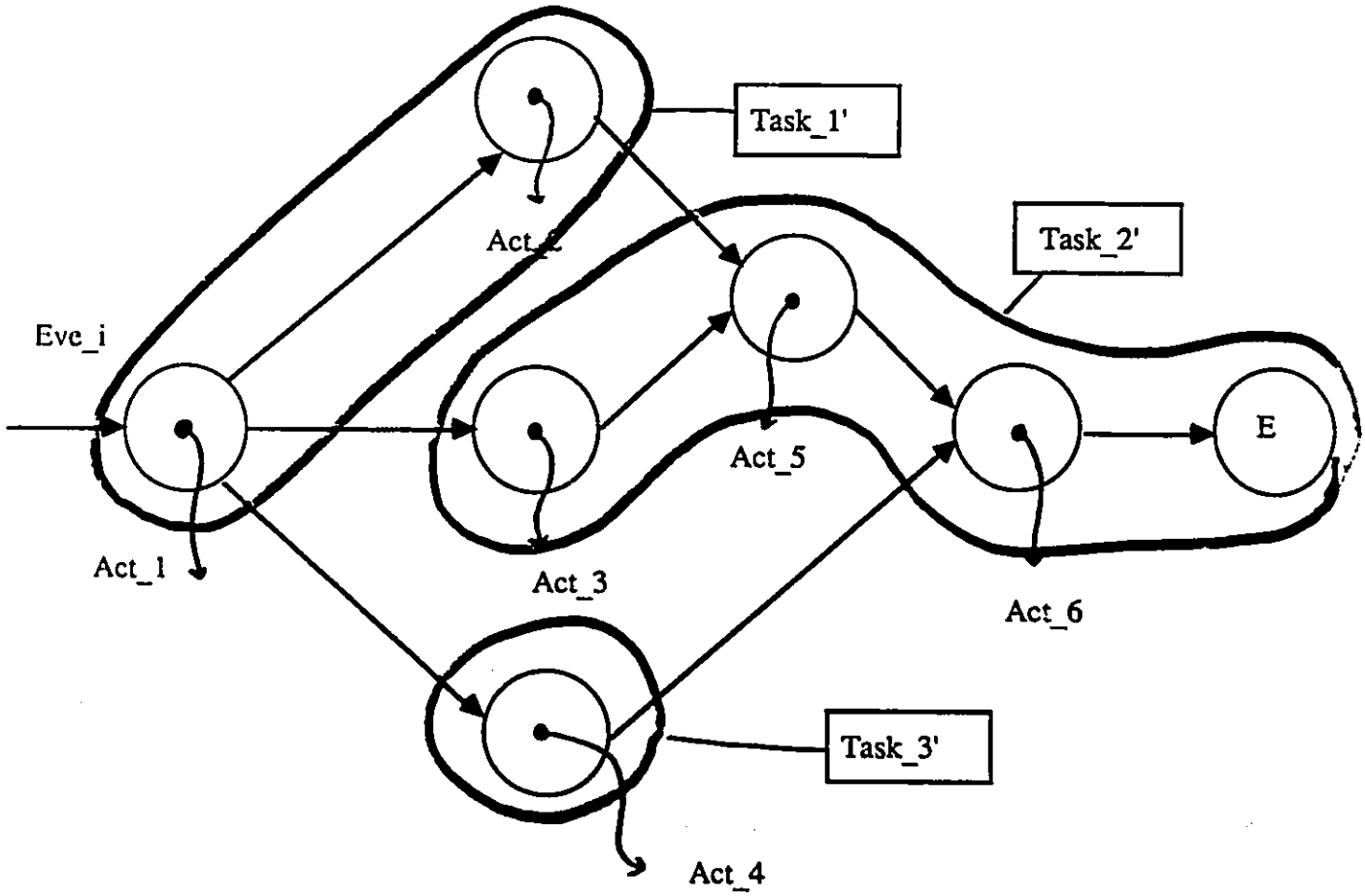


Figure 4.6 Second example in Harmony.

This partitioning however allows also for maximum parallelism (in a multiprocessor architecture of at least three processors) but, compared to the previous example, the following extra features have to be included:

- *Task_1'* is called by the event manager; however *Task_2'* is the one completing the response and informing back the event manager.

- Tasks *Task_2'* and *Task_3'* are both created by *Task_1'*; but this task cannot _Suicide on completion as it would destroy both the tasks, and in particular *Task_2'*.

- *Task_3'* is created by *Task_1'* but can be destroyed by *Task_2'*.

As can be seen, one has to be more careful in the task management of these tasks than in the above allocation example. Although the amount of intertask communication has not been reduced, the hierarchical organization of the tasks in example of 4.5 makes the task code more structured. The definition of three tasks is the minimum number that can specified for maximum execution concurrency. Specifying less than this number will result in having sequential execution of activities that may be executed in parallel.

Let's assume now the case of figure 4.7; in this case, assume the following partitioning:

- *Task_a* :activities 1, 3, 5, 7 and End node;

- *Task_b* :activity 2 and

- *Task_c* :activities 4 and 6.

Because of the blocking nature of the _Send primitive, *Task_a* cannot execute activity 3 unless it is unblocked by _Reply's from both tasks *Task_b* and *Task_c*; further on, if activity 4 is completed before activity 3 then *Task_c* will be blocked waiting for a reply from *Task_a* that it does not need; *Task_c* could be instead executing activity 6. To allow this, *Task_c* should synchronize with *Task_a* by means of a task of type *Transport* (see section 4.4), which will be the one blocked, while *Task_c* may continue its execution. The same *Transport* type task should be used by *Task_a* after executing activity 1. The interest focuses here particularly on *Task_c* and *Transport*. Notice that two tasks of type *Transport* are required.

```
unsigned_Pnumber= P_c;
struct TASK_TEMPLATE_Template_list[] =
{
     { Task_c, Child_c, Sts_c, N_c, P_c}
     {0,0,0,0,0}
};
unsigned_Pnumber=P_x;
struct TASK_TEMPLATE_Template_list[] =
{
     { Transport_x, Mail_deliver_x, Sts_x, N_x, P_x}
     {0,0,0,0,0}
};
unsigned_Pnumber=P_y;
struct TASK_TEMPLATE_Template_list[] =
{
     { Transport_y, Mail_deliver_y, Sts_y, N_y, P_y}
     {0,0,0,0,0}
};

Child_c()
{
    /* GLOBAL_INDEX of Main is Task_a. */
    /* Both Transports were created in Main. */
     _Receive ( data_1, Main);
     _Reply ( rply_1, Main);
     Execute_Activity_4;
```

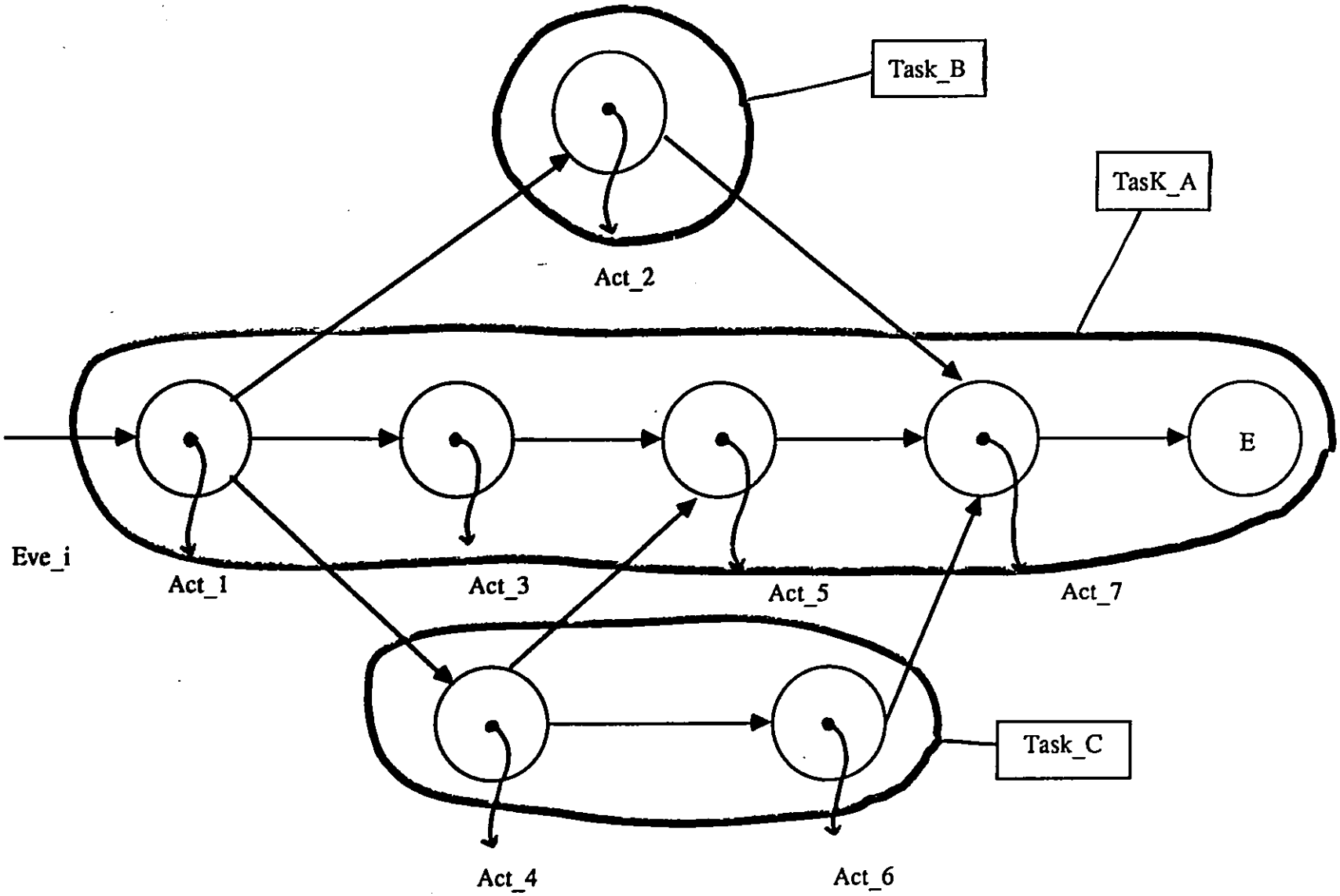


Figure 4.7 Third allocation example in Harmony.

```
      _Receive (data_4,  Mail_deliver_x);
      Execute_Activity_6;
      _Send ( data_6, rply_6, Main);
};

Mail_deliver_x()
{
      _Send ( data_4, rply_4,Child_c);
      data_Mail:=data_4;
      _Send ( data_M, rply_5, Main );
      _Suicide;
};
```

As was mentioned in 4.4, tasks like *Transport* should be created by the main task, *Task_a* in this case. These tasks can be considered as support tasks in this case for communication and synchronization purposes. An other possibility is to create all these tasks initially (like in a static scheme), and only call them when required.

4.5.2 / VersaDos.

VersaDos [MOTO 84] is a real-time multitasking operating system for the Mc 68000 family of microprocessors, based on the RMS 68k Real-time Executive. It has a modular multilayered structure, is event driven and has provision for server tasks. The kernel provides multitasking control, supports memory management, contains the physical I/O facilities, allows for intertask communication and conflict resolution of access to the MPU. It does not provide support for a multitasking environment in a multiprocessor architecture but is priority driven and time sliced (if time slicing is enabled).

In VersaDos, a process is defined as a program running in the

supervisor mode, while a task runs in the user mode. System tasks can only be created by other system tasks.

4.5.2.1 / Relevant characteristics of VersaDos.

The executive of this uniprocessor multitasking operating system includes a number of managers: time manager, task manager, event manager, memory manager, semaphore manager, etc..

Of these, the interest will be focused on two, the Event manager and the Task manager.

The Event manager enables the communication between tasks. There are two modes of communication: synchronous and asynchronous. Although in many systems these adjectives are used when referring to the use of blocking or non-blocking send primitives, it does not apply in the case of VersaDos, as shall be explained later on.

Tasks may communicate by sending messages (moving data or passing pointers to data). The messages can have variable length and are automatically queued at the Asynchronous Service Queue (ASQ) of the target task. The ASQ is a buffer-like data structure consisting of a control block, an overflow portion and a variable length portion that withholds the messages. In order to communicate in either mode (synchronous or asynchronous), the receiving task must thus have an ASQ associated with it. The primitive *GTASQ* allocates an ASQ to a target task; once allocated the ASQ is identified with the task. The ASQ can be considered

like a dedicated mailbox, as many tasks may write into it, but only the owner can read from it. As such, the above directive *GTASQ* (jointly with *SETASQ*) can be looked up as a Create_mailbox primitive. It is also possible to bypass the ASQ and place the data in a prespecified buffer, the Default Receive buffer.

The directive *SETASQ* is used to determine the characteristics of this ASQ. The ASQ, its associated Asynchronous Service Routine (ASR), and the Default Receive buffer can be enabled or disabled. If the ASQ is disabled, new incoming events will be rejected. If the target task does not have an ASQ allocated, or the ASQ is full, error messages are generated if new events arrive.

A message is sent by means of the directive *QEVNT*, which queues the message at the target's task ASQ. This directive is non-blocking.

The differences between the communication modes appear at the receiving end of the communication protocol:

- In the synchronous mode of communication there are two directives available, *GTEVNT* (get an event) and *RDEVNT* (read an event). In the case of the first directive, the task waits for the event to arrive (it is consequently blocking, like Receive), while in the second one the task is not blocked whether the event is present or not (non-blocking, like Try_receive). In the case of *GTEVNT* the task goes to GET_AN_EVENT wait state.

- To provide an asynchronous communication mechanism,

VersaDos has an interrupt like mode for event handling, which is different from the in-line code. This is necessary for real-time situations, where handling of incoming events is required immediately. This event handling code is called the Asynchronous Service Routine (ASR) and it is identified when specifying the ASQ for the target's task. The primitives used in this mode are: *WTEVNT*, which makes sure that both ASQ and ASR are correctly enabled, and then puts the task in a WAIT_FOR_EVENT state (the ASR is entered if the ASQ is not empty); *RTEVNT*, which is the last directive of the ASR and which returns execution to the in-line code.

Essentially, in the context of VersaDos, in the synchronous mode the receiver determines when the event is going to be handled, while in the asynchronous mode the event is handled when it occurs, temporarily interrupting normal execution of the receiving task and provided the appropriate ASR is enabled.

The job of the Task manager is to keep track of the state, maintenance and processing of the existing tasks in the system. The task manager is responsible for the creation and deletion of tasks. In VersaDos a task can be in one of the following states: suspended, ready to run, running, dormant and waiting. Figures 4.8 and 4.9 show a basic and expanded state diagram of tasks in VersaDos.

A task is completely specified by a Task Control Block (TCB)

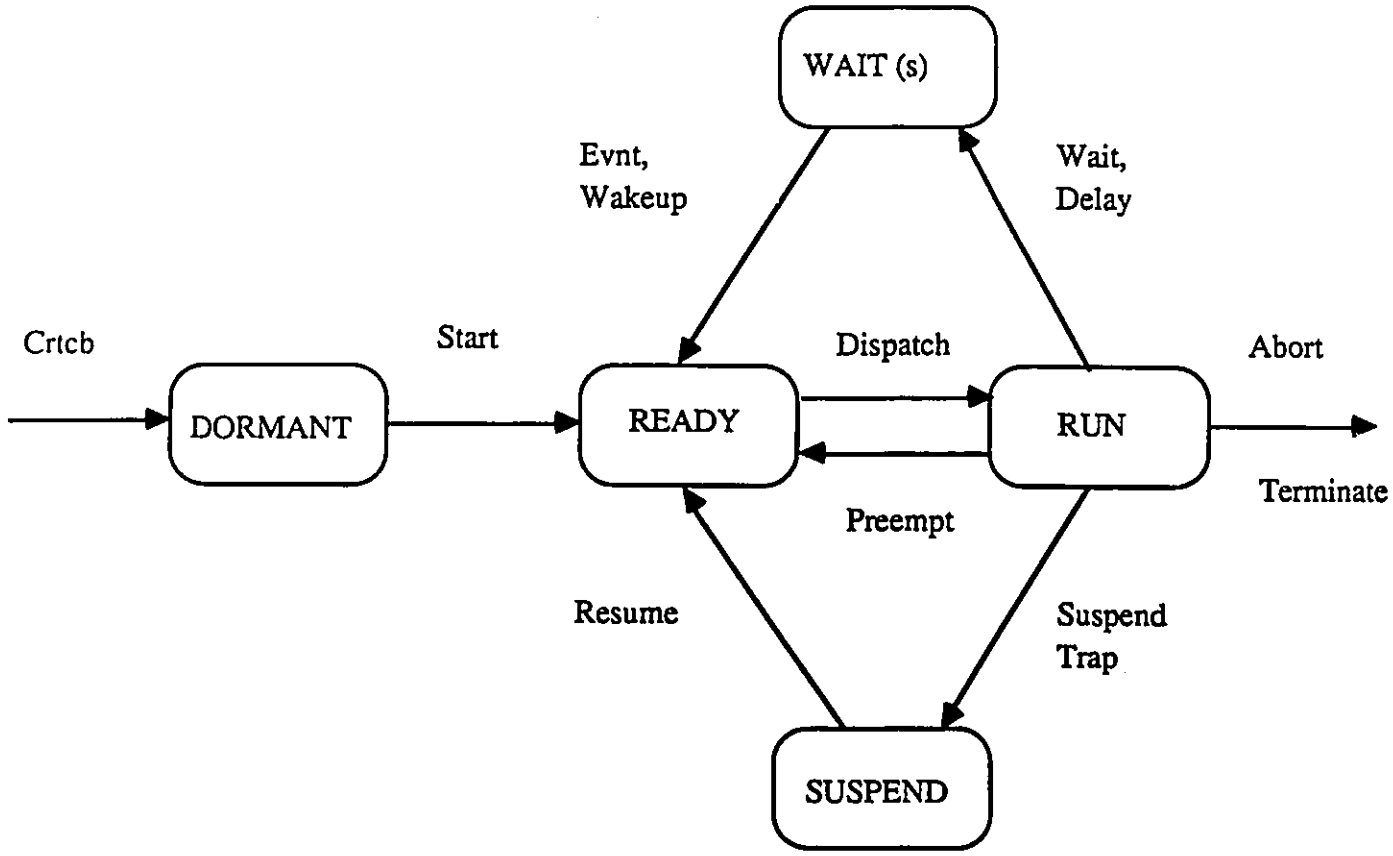


Figure 4.8 Basic Task States of VersaDos.

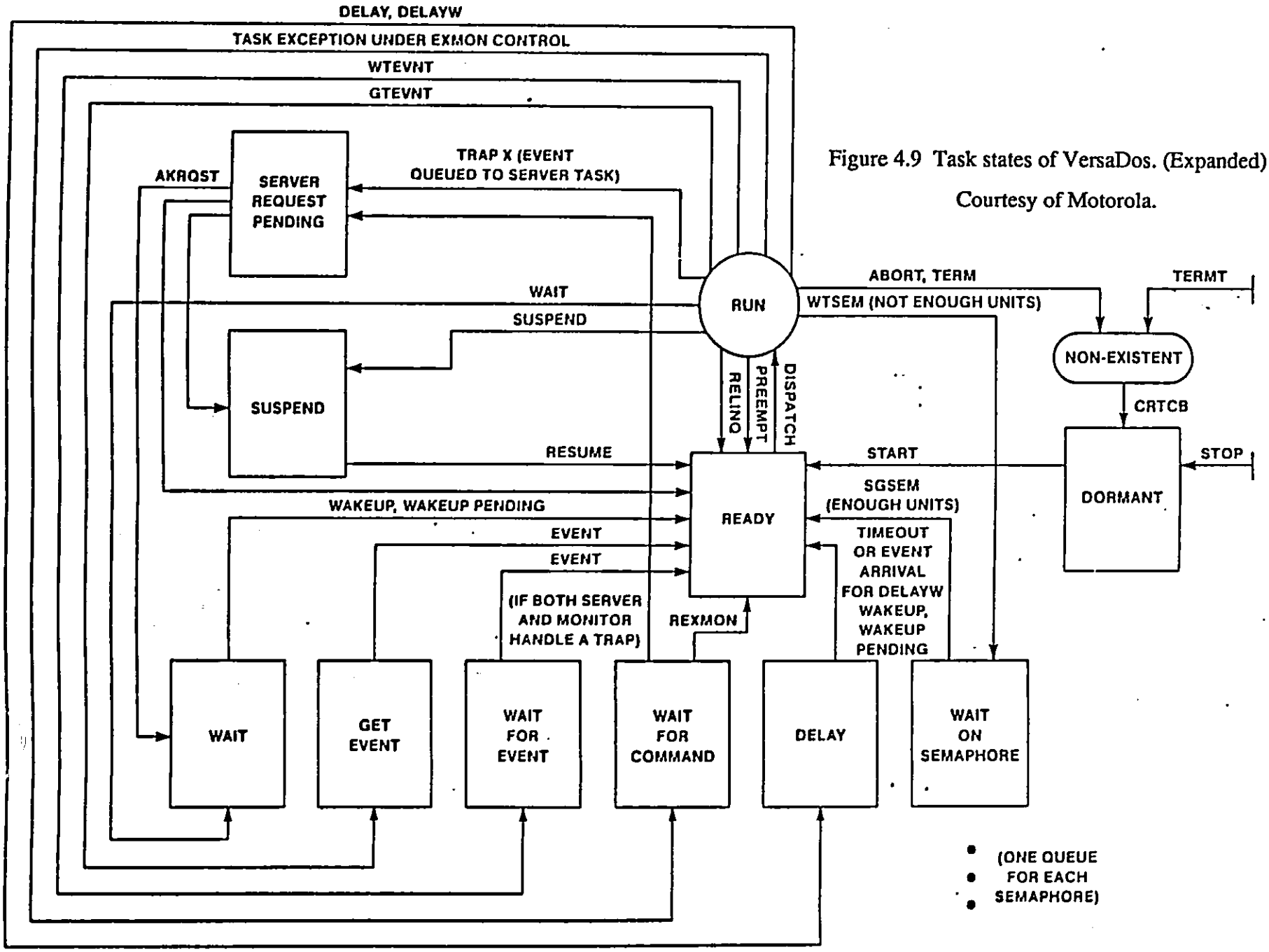


Figure 4.9 Task states of VersaDos. (Expanded)

Courtesy of Motorola.

which contains information about the task's name and code, it's priority and attributes like the associated code segment, ASQ and some more control and protection portions.

There are several directives provided for task management: some are for task state transitions, like *START, WAIT, RESUME,* and others are for task creation and deletion, like *CRTCB* to create a TCB for a task, and *ABORT, TERMT* and *TERM* which are used for task termination and release of the associated resources.

A task is created when its TCB is specified. Upon creation, an initial priority (which the task never exceeds) is given; this priority may be modified at any time after creation. Finally a monitor task is assigned to the newly created task or sub-task; this monitor can be the task requesting the creation, the monitor of the requestor or a third task. When the subtask terminates, it is this monitor task which receives a corresponding event (code $05) in its ASQ indicating the id of the task terminated and the author of the deletion.

4.5.2.2 / Activity to task partitioning in VersaDos.

In this case, the problem of activity to task allocation considers a uniprocessor environment. For this reason and as was previously explained, the partitioning should be done as to define a set of consecutive tasks. The determination of which activities are to be allocated to the same task is dependent in priorities, resource requirements, interactivity transfers,

activity execution times, etc.. For this reason the following are considered:

- the only resource required is memory; consequently there is no parallelism provided by peripheral units, and no requirement for their management.

- priorities are assigned to the processes; all the activities in a same ASN have initially the same priority. Thus the partitioning is to be done on a per ASN basis.

- an interactivity information transfer table is considered; in this table the parameters indicate the weight of the transfer: a load of '1' indicates just signalling of completion of the activity, while for the rest of the parameters the value is proportional to the total number of bytes that must be transferred to the next activity.

Considering the ASN represented in figure 4.10, the following table is considered:

	Act1	Act2	Act3	Act4	Act5	Act6	Act7
Act1	0	2	5	3	0	0	0
Act2	0	0	0	0	0	0	1
Act3	0	0	0	0	4	0	0
Act4	0	0	0	0	8	2	0
Act5	0	0	0	0	0	0	1
Act6	0	0	0	0	0	0	4
Act7	0	0	0	0	0	0	0

In order to provide multiple event handling, and considering that VersaDos supports dynamic task creation and deletion, no main task is going to be defined and a maximum number N of simultaneous copies of the same task is going to be enforced.

Given these considerations, the following activity to task allocation is proposed:

- *Task_a*: activities 1, 2 and 3;

- *Task_b*: activities 4 and 5;

- *Task_c*: activities 6 and 7 and the end node.

A continuously running task, the *Event_Manager* is considered: this task is responsible for event detection and starting of the responses, in particular for the start of the first task in the sequence.

The pseudo code for these tasks could be:

```
Task_a:
GTEVNT (Input_Event);      Get input data from Event_Manager.
EXEC_ACT1;
CRTCB (Task_c);      Create Task_c; monitor task is
                        Event_manager;
GTASQ (Task_c);      Associate an ASQ to Task_c;
N_c:=N_c-1;
CRTCB (Task_b);      Create Task_b; monitor task is Task_c;
GTASQ (Task_b);      Associate an ASQ to Task_b;
QEVNT (Task_b, Data_1);
EXEC_ACT3;
QEVNT (Task_b, Data_3);
EXEC_ACT2;
QEVNT (Task_c, Data_2);
START  (Task_b);  By only starting Task_b after completion of
                     execution of Act_2, we reduce the
                     number of context switches.
TERM;

Task_b:
GTEVNT (Data_1);
EXEC_ACT4;
QEVNT (Task_c, Data_4);
EXEC_ACT5;
START (Task_c);
TERM.

Task_c:
GTEVNT (Data_4);
EXEC_ACT6;
GTEVNT (Data_2);
GTEVNT (Data_5);      The order here is unimportant.
EXEC_ACT7;
TERM.                 Informs the Event_manager Task of the
                        completion of the process.
```

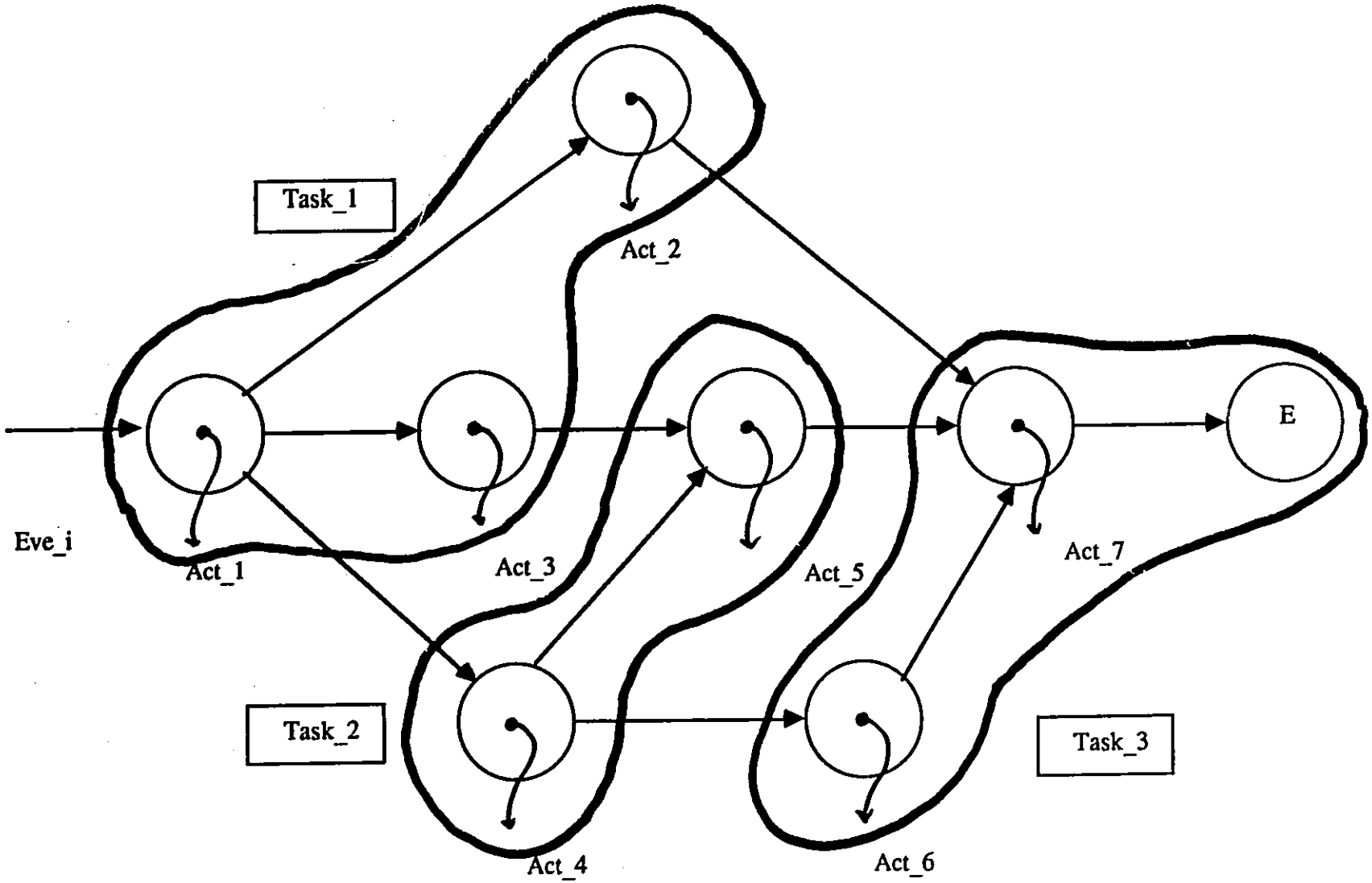


Figure 4.10 Allocation example in VersaDos.

The Event_Manager task is responsible for the creation of Task_a. This task is always in a state ready to process an incoming starting event for this process; upon arrival, the ASR of the Event_Manager is entered. The pseudo code for the ASR of Event_manager could look like:

```
CREATE_RESPONSE:
  Begin
   CRTCB (Task_a);    Monitor is Event_manager; By doing this
                        some pipelining in the execution of
                         several processes is allowed.
   GTASQ (Task_a);
   Na:=Na-1;
   QEVNT(Task_a, Input_Event);
   START (Task_a);
  end;

TASK_COMPLETED:
  Begin
   Determine_Source_of_event (Source);
   case Source of
      Task_a :  Na:=Na+1;  Na is the number of tasks
                           of type Task_a that are active
                                 simultaneously.
      Task_c : Nc:=Nc+1;
   end;  {end of case}
  end;

START_PROCESS_RESPONSE:
  Begin
   If (Na≠Ø) then CREATE_RESPONSE;
      else OPTIONS; event may be queued for further
              processing or it may be disregarded, etc..
  end;

ASYNCHRONOUS SERVICE ROUTINE:
  Begin
   GTEVNT (event);
   If (event=$Ø5) then TASK_COMPLETED;
       else START_PROCESS_RESPONSE;
  end;
```

In this context, the Event_Manager is responsible for the determination of the identity of the incoming event and the designation of the type of response expected.

4.6 / Concluding remarks.

In order to obtain a responsive multitasking system from a PAD based definition, a number of characteristics of the operating system and of the underlying hardware of the system have to be considered. For this reason, real-time, multitasking operating systems, and multiprocessor environments were reviewed in the first three sections of the chapter. In section 4 some guidelines for using Process Activity Diagrams for the allocation of activities to tasks in a multitasking system was proposed. It was shown that the relevant features of the operating systems, and in particular its task management and intertask communication and synchronization mechanisms, affect the partitioning of activities. In section five two examples, Harmony and VersaDos, were used for this purpose.

The guidelines drawn in section 4.4 indicate that if the software engineer is a specialist in one of the existing general purpose multitasking operating systems, and in any of the interprocessor communication features involved in the executive, then he would be able to use their particularities to his advantage in the activity to task allocation. This is to say that experience and feeling of the system operation and system environment (software and hardware wise) are determinant features for a good and efficient task code generation and further

multitasking system implementation. Because there are very few tools available, the determination of tasks by partitioning interacting activities in a process can provide both a first approach and a deterministic solution to the problem.

If the designer is free to choose the operating system, some features that would be useful in a multitasking operating system that would be ideally suited to provide a simple mapping of the system's ASN's to tasks are: first, the communication mechanism should be non-blocking, but it does not matter whether messaging or mailboxes are used. The receive should be blocking however, and finally there is of course no reply necessary. Second, task creation should be dynamic; as such the user has the option of implementing a static or dynamic task creation policy for his system; this also allows for a more efficient handling of multiple occurrence of events. Finally, it would be useful to have generic tasks, such as gates and timer/counters, which would be used as server tasks by the different user tasks in the process.

CHAPTER V

CONCLUSION.

The purpose of this thesis was to show the applicability of Process Activity Diagrams for the design of real time multiprocessor systems by investigating the feasibility of developing design tools based on PAD's for the non-computer expert. Two aspects of the design process which are particularly relevant for the non-computer expert were considered in detail in this thesis:

The first one concerned the correct specification of the system requirements. In order to achieve this, a user friendly menu driven ASN matrix editor was developed. This editor allows the interactive entering of event driven systems defined in terms of PAD's, into a computer for checking and simulation purposes. Although limited in features like the maximum number of nodes and events per ASN, this semi-graphical editor serves to prove the possibility of developing such an editor.

To complement the editor, the verification of the correctness of the system representation in terms of ASN matrices was

considered in detail. Two main types of errors that can occur in this representation were identified. To detect simple 'mechanical' errors caused by incorrect and/or incomplete parameters (matrix entries, node and event definition and simple connectivity anomalies) a program called INSPECT was developed. For the detection of more complex connectivity problems, that may introduce deadlocks and/or multiple triggering situations in a process, three methods were proposed.

The problem of partitioning the system processes into tasks was the second design aspect considered in this thesis. This topic was selected because the proper partitioning of the system into tasks can have great effect on the responsiveness of the system. Different implementation environments, depending on the type of multitasking operating system and type of hardware configuration used, were considered and a set of guidelines were drawn for the activity to task allocation. To show the applicability of these guidelines, two examples, one for Harmony in a multiprocessor based system and the second for VersaDos in a uniprocessor based system were presented.

As previously stated, only the feasibility of developing user friendly tools was considered; a number of improvements and additional tools are needed to provide a complete toolset.

1 - A full graphical editor would be friendlier than a matrix editor. It would be relatively straight forward to integrate this

editor with the symbolic editor (SYMED) of the ARTT project.

2 - In addition, the matrix editor should be improved by eliminating some of its limitations:

i - The limitation on the number of nodes and events per process, which was imposed by screen restrictions.

ii - The possibility of having successive refinement of the activities of the system; this means the definition of the activities as lower level ASN's. For this purpose and to complete the editor, an activity descriptor tool (DESCRIPTOR) becomes necessary.

iii - The limited checking capability of the editor should be enhanced and some checking capability should be included during the editing process. In line with this, a syntax checker to verify the correctness of the connectivity of the ASN's describing the system process is required.

3 - Furthermore, various aids for the efficient partitioning of the system processes into tasks should be developed:

i - The activity to task allocation problem can be solved by an expert system like tool, which would include different sets of rules depending on the environment (both hardware and software), and statements describing the processes of the

system (ASN's).

ii - To analyze and test the activity to task allocation chosen, a simulator should be developed. This has to be a simulator flexible enough to allow the definition of parameters such as task and activity management (task creation and termination, dynamic or static task allocation to processors), activity execution times, process information flow requirements, intertask communication overhead, task priorities and queue management among a few would have to be considered. The results could be as well compared in terms of performance and processor and response time requirements with the multiactivity simulator developed in [JOAN 86].

REFERENCES and BIBLIOGRAPHY.

ALLW 80 S.T. Allworth, "Introduction to Real Time Software Design.", Springer-Verlag, New York Inc., 1980.

BENA 82 M. Ben-Ari. "Principles of Concurrent Programming.", Prentice/ Hall International, Englewood Cliffs, New Jersey, 1982.

BOWE 80 B.A. Bowen & R.J.A. Buhr, "The Logical Design of multiple-microprocessor systems." Prentice-Hall, Englewood Cliffs, New Jersey 1980.

BUHR 84 R.J.A. Buhr, "System Design with Ada.", Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1984.

CASH 80 P. Cashin, "Inter-Process Communication", BNR report, Ottawa, Ontario, 1980.

CHRI 75 Nicos Christofides, "Graph Theory: an algorithmic approach", Academic Press 1975.

FISH 86 David A. Fisher and Richard M. Weatherly, "Issues in the design of a distributed Operating System for Ada" IEEE Computer Magazine, May 1986.

GENT 81 W. Morven Gentleman, "Message passing between Sequential Processes: the Reply primitive and the Administrator concept", Software-Practice and Experience, vol 11 p 435-466, John Wiley and Sons, Ltd. 1981.

GENT 83 W. Morven Gentleman, "Using the Harmony Operating System", National Research Council report, December 1983.

HANS 75 B. Hansen, "The Programming Language Cocurrent Pascal.", IEEE Transactions in Software Engineering, June 1975.

HARA 65 Frank Harary, Robert Z. Norman and Dorwin Cartwright, "Structural Models: an introductio to the Theory of Directed Graphs", John Wiley and Sons, Ltd, 1965.

HOAR 74 C. A. R. Hoare, "Monitors: an Operating System Structuring Concept.", Communications of the ACM, November 1978.

HOAR 78 C. A. R. Hoare, "Communicating Sequential Processes.",

Communications of the ACM, August 1978.

HOAR 85 C. A. R. Hoare, "Communicating Sequential Processes", C. A. R. Hoare Series Editor, Prentice/Hall International, Englewood Cliffs, New Jersey, 1985.

HOLL 80 Wesley Chu and Leslie J. Holloway et al, "Task allocation in distributed data processing", IEEE Computer Magazine, November 1980.

HWAN 84 Kai Hwang and Faye Briggs, "Computer Architecture and Parallel Processing", Mc Graw-Hill Computer Science Series, Mc Graw-Hill Book Company, 1984.

INTE 84 "iRMX 86 Operating System, Programmer's Reference Manual, Intel Corporation 1984.

JANS 85 Philippe A. Janson, "Operating Systems: Structures and Mechanisms" , Academic Press 1985.

JOAN 86 Robert Joannis, "Specification and Design of multiple microprocessor systems using Process Activity Diagrams", M. A. Sc. thesis, University of Ottawa, 1986.

KRIE 86 Moshe Krieger and Robert Joannis, "Process Activity Diagrams", Proceedings of the ISMM International

Symposium on Software and Hardware applications of microcomputers, Beverly Hills, California, 1986.

LIEB 85 Burt H. Liebowitz and John H. Carson, "Multiple Processor Systems for Real-Time Applications", Prentice/Hall International, Englewood Cliffs, New Jersey, 1985.

MOTO 84 "M-68000 Family Real-Time Multitasking Software User's Manual", Microsystems, 1984.

NUTT 73 J. D. Noe and Gary J. Nutt, "Macro E-nets for representation of Parallel Systems", IEEE Transactions on Computers, August 1973.

PENG 87 Dartzen Peng and Kang G. Shin, "Modeling of Concurrent Task Execution in a Distribute System for Real-Time Control." IEEE Transactions on Computers, April 1987, pages 500 - 516.

PERN 84 Richard Perng-Yi Ma, " A Model to solve Timing-Critical Application problems in Distributed Computer Systems", IEEE Computer Magazine, January 1984.

PETE 83 J. Peterson and A. Silberschatz, "Operating System Concepts", Addison-Wesley, 1983.

PETE 77 James L. Peterson, "Petri Nets", Computing Surveys, September 1977.

RAMA 84 C.V. Ramamoorthy et al.. "Software Enginnering: Problems and Perspectives", IEEE Computer Magazine, October 1984, p 191-209.

SAIB 85 Sabina Saib, "Ada: an Introduction", CBS College Publishing, Holt, Reinhart and Winston, New York, 1985.

SATY 80 M. Satyanarayanan, "Multiprocessing: an annotated Bibliography", IEEE Computer Magazine, May 1980.

SESH 61 Sundaram Seshu and B. Reed, "Linear Graphs and Electrical Networks", Addison-Wesley Publishing, 1981.

SHAW 74 Alan C. Shaw, "The logical design of Operating Systems", Prentice/Hall series in Automatic Computation, Prentice/Hall International, 1974.

TANE 81 Andrew S. Tanenbaum, "Computer Networks", Prentice/Hall International, 1981.

WEIT 80 Cay Weitzman, "Distributed micro/minicomputer Systems Structure, Implementation and Application", Prentice/Hall International, 1980.

APPENDIX A.

PROCESS ACTIVITY DIAGRAM EDITOR.

This appendix contains the program listing for the Process Activity Diagram Editor, written in TURBO PASCAL (c), for the IBM Personal Computer (c).

TABLE of CONTENTS.

```
(Activity Sequence Nets System declarations)

CONST max_number_of_nets = 10;
      max_number_of_activities = 50;
      max_number_of_ext_events = 20;

      max_number_of_nodes_per_net = 16;
      max_number_of_events_per_net = 16;

      (Valid entries for ASN tables)
      not_connected  = '0';
      input_trigger  = 'I';
      enable         = 'E';
      disable        = 'D';

(*****************************************************************)

TYPE node_type = (activity_initiator, wait, merge, branch, gate,
                  counter, terminator);

     event_type = (int_event, ext_event); (internal event (node trigger) or
                                           external event)

     counter_status = (counter_expired, counter_disabled);

     node_definition = RECORD
                       CASE node: node_type of
                         activity_initiator: (activity_id:
                                              1..max_number_of_activities);
                         wait: ();
                         merge: ();
                         branch: (branch_variable_id: string[1]);
                         gate: ();
                         counter: (counter_variable_id: string[1]);
                         terminator: ();
                       end;

     event_definition = RECORD
                        CASE event: event_type of
                          ext_event: (event_number:
                                      1..max_number_of_ext_events);
                          int_event: (node_number:
                                      1..max_number_of_nodes_per_net;
                                      CASE node: node_type of
                                        activity_initiator: ();
                                        wait: ();
                                        merge: ();
                                        branch: (branch_condition:
                                                 boolean);
                                        gate: ();
                                        counter: ();
                                        terminator: ());
                        end;

     matrix_definition = RECORD
                         number_of_events: 1..max_number_of_events_per_net;
                         number_of_nodes:  1..max_number_of_nodes_per_net;
                         matrix: array [1..max_number_of_events_per_net,
                                        1..max_number_of_nodes_per_net]
                                 of string[1];
                         end;

(*****************************************************************)

{ Global variable declarations }

VAR system_name: string[8];

    number_of_nets: 1..max_number_of_nets;
```

```
    number_of_activities: 1..max_number_of_activities;
    number_of_ext_events: 1..max_number_of_ext_events;

    asn_starting_event: array [1..max_number_of_nets]
                        of 1..max_number_of_ext_events;

    asn_node_definition: array [1..max_number_of_nets,
                                1..max_number_of_nodes_per_net]
                         of node_definition;

    asn_event_definition: array [1..max_number_of_nets,
                                 1..max_number_of_events_per_net]
                          of event_definition;

    asn_matrix: array [1..max_number_of_nets] of matrix_definition;


{             VARIABLE DECLARATIONS USED IN THIS EDITOR          }
{   These are to complement the global variables defined in the  }
{ file DEFASN.DEC.                                               }

VAR entry : string[1];

    i_position, j_position, x_position, y_position : integer;
    net_number, activity_number: integer;
    number_of_nodes,number_of_events:integer;
    node_number,event_number,ext_event_number:integer;
    epsilon, present_node, present_event : integer;

    color_terminal, new_system, printing: boolean;
    last_node_flag, one_more_node: boolean;
    starting_event_entered, ext_event_description:boolean;
    IOerr, no_problem: boolean;

    command : char;

    activity_label:array [1..max_number_of_activities]
              of string[4];
    type_of_node: array [1..max_number_of_nets,
              1..max_number_of_nodes_per_net] of char;
    type_of_event: array [1..max_number_of_nets,
              1..max_number_of_events_per_net] of char;
    counter_var: array [1..max_number_of_nets,
              1..max_number_of_nodes_per_net] of string [1];
    branch_cond_var: array [1..max_number_of_nets,
              1..max_number_of_events_per_net] of string [1];
    activity_id: array [1..max_number_of_nets,
              1..max_number_of_nodes_per_net] of integer;
    ext_event_label: array [1..max_number_of_ext_events]
              of string [4];
    ASN_label: array [1.. max_number_of_nets]
              of string [4];
    external_event: array [1..max_number_of_nets,
              1..max_number_of_events_per_net] of string [4];
    starting_event_label: array [1..max_number_of_nets]
              of string [4];
    node :node_type;

type
    length = string [80];
    name_of_file = string [14];
    word = string [4];

{_____________________________________________________________}
```

```
PROCEDURE READ_SYSTEM_DEFINITION;

   VAR char: string[1];
       file_name: string[14];
       f: text;
       net_number, i, j, number : integer;

   BEGIN
      no_problem:=true;
      ClrScr;
      writeln;
      writeln (' Insert data disk in drive B.');
      write (' Press <CR> when ready.');
      readln;
      (Form file name using system name)
      file_name := 'b:' + system_name + '.asn';
      (Open the file)
    ($I-)
     repeat
      ASSIGN (f,file_name);
      reset (f);
      IOcheck (file_name);
      If IOerr then
       begin
        (Rough_window;)
        ClrScr;
        writeln;
        writeln(' System name is incorrect. ');
        writeln(' At least one of the three files necessary to ');
        writeln(' specify the system is missing, or system name');
        write  (' is incorrect. Press Return or Q to quit.');
        read (kbd,command);
        ClrScr;
        command:=upcase(command);
        If (command<>'Q') then
         begin
          writeln;
          writeln(' Reenter system name .');
          readln(system_name);
          file_name := 'b:' + system_name + '.asn';
          no_problem:=true;
         end
         else no_problem:=false;
       end;
      until not IOerr or not no_problem;
    ($I+)
    If no_problem then
     begin
     (Read the system definition)
      readln (f,number_of_nets);
      readln (f,number_of_activities);
      readln (f,number_of_ext_events);
      for net_number := 1 to number_of_nets do
        begin
          (read starting event)
         readln (f,asn_starting_event[net_number]);
          (read asn matrix)
         readln (f,asn_matrix[net_number].number_of_nodes);
         readln (f,asn_matrix[net_number].number_of_events);
         for i := 1 to asn_matrix[net_number].number_of_events do
          begin
           for j := 1 to asn_matrix[net_number].
                                 number_of_nodes do
              read (f,asn_matrix[net_number].matrix[i,j]);
          end;
         readln (f);
          (read asn node definition)
         for i := 1 to asn_matrix[net_number].number_of_nodes do
          begin
```

```
          readln (f,number);
          asn_node_definition[net_number,i].node :=
                                    node_type(number);
          case asn_node_definition[net_number,i].node of
               activity_initiator:
                    readln(f,asn_node_definition[net_number,i].
                                    activity_id);
              branch: readln(f,asn_node_definition[net_number,i].
                                    branch_variable_id);
              counter: readln(f,asn_node_definition[net_number,i].
                                    counter_variable_id);
            end;  { end of case}
          end;
           {read asn event definition}
          for i := 1 to asn_matrix[net_number].number_of_events do
           begin
            readln (f,number);
            asn_event_definition[net_number,i].
                              event := event_type(number);
            case asn_event_definition[net_number,i].event of
              ext_event: readln (f,asn_event_definition[net_number
                                    ,i].event_number);
              int_event:
                 begin
                  readln (f,asn_event_definition[net_number,i].
                                    node_number);
                  readln (f,number);
                  asn_event_definition[net_number,i].
                                node := node_type(number);
                  if asn_event_definition[net_number,i].
                              node = branch then
                    begin
                     readln (f,number);
                     asn_event_definition[net_number,i].
                          branch_condition := boolean(number);
                    end;
                 end;
            end; {end case}
           end;
          end; {end for loop on net_number}
         CLOSE (f);
        end;

   END;

{_____________________________________________________________}

PROCEDURE Read_System_labels;

var char: string [1];
    file_name: string [14];
    f :text;
    net_number,i,j:integer;

Begin

no_problem:=true;
{ Form file name.                                             }
 file_name:='b:'+system_name+'.nam';
{ Open the label file.                                        }
{$I-}
repeat
 Assign(f,file_name);
 reset(f);
 IOcheck (file_name);
 If IOerr then
  begin
   window(1,1,80,25);
   GotoXY(2,2);
   ClrScr;
```

```
    writeln;
    writeln(' System name is incorrect. ');
    writeln;
    writeln(' At least one of the three files necessary to ');
    writeln(' specify the system is missing, or system name');
    write  (' is incorrect. Press Return or Q to quit.');
    read (kbd,command);
    ClrScr;
    command:=upcase(command);
    If (command()'Q') then
     begin
      writeln;
      write(' Reenter system name . ');
      readln(system_name);
      file_name:='b:'+system_name+'.nam';
      no_problem:=true;
     end
     else no_problem:=false;
   end;
 Until not IOerr or not no_problem;
{$I+}
 If no_problem then
  begin
{ read system label definition.                                    }
    readln(f,number_of_nets);
    readln(f,number_of_activities);
    readln(f,number_of_ext_events);
    For net_number:=1 to number_of_nets do
     begin
{ read starting event label.                                       }
      readln(f,starting_event_label[net_number]);
{ read ASN labels.                                                 }
      readln(f,ASN_label[net_number]);
     end;
    For i:=1 to number_of_ext_events do
     begin
{ read all external events.                                        }
      readln(f,ext_event_label[i]);
     end;
    For i:=1 to number_of_activities do
     begin
{ read all activities in the system.                               }
      readln(f,activity_label[i]);
     end;
     Close(f);
  end;

End;

{_____________________________________________________________}

PROCEDURE Read_System_characters;

var i,j :integer;
    file_name : string[14];
    f : text;

Begin
 no_problem:=true;
 file_name:= 'b:'+system_name+'.char';
{$I-}
repeat
 ASSIGN (f,file_name);
 reset(f);
 IOcheck (file_name);
 If IOerr then
  begin
   window(1,1,80,25);
   GotoXY(2,2);
   ClrScr;
```

```
   writeln;
   writeln(' System name is incorrect. ');
   writeln(' At least one of the three files necessary to ');
   writeln(' specify the system is missing, or system name');
   write  (' is incorrect. Press Return or Q to quit.');
   read (kbd,command);
   ClrScr;
   command:=upcase(command);
   If (command<>'Q') then
    begin
     writeln;
     writeln(' Reenter system name. ');
     readln(system_name);
     file_name:= 'D:'+system_name+'.char';
     no_problem:=true;
    end
    else no_problem:=false;
   end;
  until not IOerr or not no_problem;
{$I+}
If no_problem then
 begin
  readln(f,number_of_nets);
  readln(f,number_of_activities);
  readln(f,number_of_ext_events);

{ We now save the characters associated with each node          }

 For net_number:=1 to number_of_nets do
  begin
   For i:=1 to asn_matrix[net_number].number_of_nodes do
    begin
     readln(f,type_of_node[net_number,i]);
     If (type_of_node[net_number,i] = 'C') then
       readln(f,counter_var[net_number,i]);
     If (type_of_node[net_number,i] = 'A') then
       readln(f,activity_id[net_number,i]);
    end;
  end;

{ Now it is the turn of the events to be saved.                 }

 For net_number:=1 to number_of_nets do
  begin
   For j:=1 to asn_matrix[net_number].number_of_events do
    begin
     readln(f,type_of_event[net_number,j]);
     If (type_of_event[net_number,j] = 'B') then
       readln(f,branch_cond_var[net_number,j]);
    end;
  end;

 Close(f);
end;

End;

{_____________________________________________________________}
```

```
PROCEDURE SAVE_SYSTEM_DEFINITION;

   VAR char: string[1];
       file_name: string[14];
       f: text;
       net_number,i,j: integer;

  BEGIN
   Clrscr;
   writeln;
   writeln (' Insert data disk in drive B.');
   write (' Do you want to save the system definition on disk (Y/N)? ');
   read(kbd,char);
   if (char = 'Y') or (char = 'y') then
     begin
          {Form file name using system name}
      file_name := 'b:' + system_name + '.asn';
          {Open the file}
      ASSIGN (f,file_name);
      rewrite (f);
          {Save the system definition}
      writeln(f,number_of_nets);
      writeln(f,number_of_activities);
      writeln(f,number_of_ext_events);
      for net_number:=1 to number_of_nets do
       begin
            {write starting event}
        writeln (f,asn_starting_event[net_number]);
            {write asn matrix}
        writeln (f,asn_matrix[net_number].number_of_nodes);
        writeln (f,asn_matrix[net_number].number_of_events);
        for i := 1 to asn_matrix[net_number].number_of_events do
          begin
           for j := 1 to asn_matrix[net_number].
                                 number_of_nodes do
             write (f,asn_matrix[net_number].matrix[i,j]);
          end;
          writeln (f);
            {write asn node definition}
          for i := 1 to asn_matrix[net_number].number_of_nodes do
           begin
             writeln (f,integer(asn_node_definition[net_number,i].
                                          node));
           case asn_node_definition[net_number,i].node of
                activity_initiator:
                 writeln (f,asn_node_definition[net_number,i].
                                          activity_id);
               branch: writeln(f,asn_node_definition[net_number,i].
                                          branch_variable_id);
                 counter: writeln(f,asn_node_definition[net_number,i].
                                           counter_variable_id);
                 end;
           end;
            {write asn event definition}
           for i := 1 to asn_matrix[net_number].number_of_events do
            begin
              writeln(f,integer(asn_event_definition[net_number,i].
                                        event));
              case asn_event_definition[net_number,i].event of
                  ext_event: writeln (f,asn_event_definition[net_number
                                          ,i].event_number);
                  int_event:
                     begin
                       writeln (f,asn_event_definition[net_number,i].
                                          node_number);
                       writeln (f,integer(asn_event_definition
                                          [net_number,i].node));
                       if asn_event_definition[net_number,i].
                                          node = branch then
```

```
                        writeln (f,integer(asn_event_definition
                             (net_number,il.branch_condition));
                    end;
                end; (end case)
              end;
          end; (end for loop on net_number)
          CLOSE (f);
          writeln;
          writeln (' The system definition has been saved in the file: ',
                   file_name);
          writeln;
        end;
  END;

{_____________________________________________________________}

{ These procedures are used to save in memory the system labels }
{ and to be able to restore these labels afterwards.           }

PROCEDURE Save_System_labels;

var char: string [1];
    file_name: string [14];
    f :text;
    net_number,i,j:integer;

Begin

{ Form file name.                                              }
 file_name:='b:'+system_name+'.nam';
{ Open the label file.                                         }
 Assign(f,file_name);
 rewrite(f);
{ save system label definition.                                }
 writeln(f,number_of_nets);
 writeln(f,number_of_activities);
 writeln(f,number_of_ext_events);
 For net_number:=1 to number_of_nets do
 begin
{ write starting event label.                                  }
  writeln(f,starting_event_label[net_number]);
{ write ASN labels.                                            }
  writeln(f,ASN_label[net_number]);
 end;
 For i:=1 to number_of_ext_events do
 begin
{ write all external events.                                   }
  writeln(f,ext_event_label[i]);
 end;
 For i:=1 to number_of_activities do
 begin
{ write all activities in the system.                          }
  writeln(f,activity_label[i]);
 end;
 Close(f);

End;

{_____________________________________________________________}

PROCEDURE Save_System_characters;

var i, j : integer;
    file_name :string [14];
    f : text;

{    The purpose of this procedure, and of the following       }
{ READ_SYSTEM_CHARACTERS is to maintain the characters describing}
{ the differnt nodes and events of the ASNs involved in the given}
{ system. The purpose of these characters is to be able to verify}
```

```
{ what the ASN contains, without having to go out to the PRTASN  }
{ program. The characters for nodes and events correspond exactly}
{ except in the case of the END (terminator) node and of the     }
{ external event. In both cases, the character associated with it}
{ is 'E'. There is no confusion, as a terminator node will not   }
{ have any triggers coming from it, and that an external event   }
{ does not involve any node.                                     }

Begin

 file_name:= 'b:'+system_name+'.char';
 ASSIGN (f,file_name);
 rewrite(f);
 writeln(f,number_of_nets);
 writeln(f,number_of_activities);
 writeln(f,number_of_ext_events);
{ We now save the characters associated with each node           }
 For net_number:=1 to number_of_nets do
  begin
   For i:=1 to asn_matrix[net_number].number_of_nodes do
    begin
     writeln(f,type_of_node[net_number,i]);
     If (type_of_node[net_number,i] = 'C') then
       writeln(f,counter_var[net_number,i]);
     If (type_of_node[net_number,i] = 'A') then
       writeln(f,activity_id[net_number,i]);
    end;
  end;
{ Now it is the turn of the events to be saved.                  }
 For net_number:=1 to number_of_nets do
  begin
   For j:=1 to asn_matrix[net_number].number_of_events do
    begin
     writeln(f,type_of_event[net_number,j]);
     If (type_of_event[net_number,j] = 'B') then
       writeln(f,branch_cond_var[net_number,j]);
    end;
  end;
 Close(f);

End;

{_________________________________________________________________}
```

```
PROCEDURE Error (Msg : length);

Begin

 Clrscr;
 writeln;
 write(Msg);

End;

{_____}

PROCEDURE IOCHECK (FILE_NAME : name_of_file);

VAR IOcode : integer;
    ch :char;

 Begin

  ClrScr;
  IOcode:=IOresult;
  IOerr:=(IOcode<>0);
  If IOerr then
   begin
    case IOcode of
     $01 :Error(' File of system does not exist.');
     $02 :Error(' File not open for input.');
     $F0 :Error(' Disk write error.');
     $05 :Error(' Can"t read from this file.');
     else
     Error(' I/O error has ocurred.');
    end;    ( end of case)
    writeln;
    writeln('    Press <CR>.');
    read(kbd,ch);
  end;
  ClrScr;

End;

{_____}

FUNCTION READ_NUMBER  : integer;

var IOcode : integer;
    IOerror : boolean;
    value : integer;

Begin
{$I-}
  readln(value);
  IOcode:=IOresult;
  IOerror:=(IOcode<>0);

  If IOerror then
   begin
    case IOcode of
      $10 :begin
            Error(' Error in numeric Format of entry; try again.');
            write(' Press <CR>. ');
            readln;
           end;
      else begin
            Error(' Incorrect Entry; try again.');
            write(' Press <CR>. ');
            readln;
           end;
    end; { end of case }
   read_number:=0;
```

```
   end
   else
    read_number:=value;
{$I+}

End;

{_____________________________________________________________}

FUNCTION READ_NAME : word;

var length_of_name: integer;
    who: word;

Begin

 readln(who);
 length_of_name:=length(who);
 case length_of_name of
  0:read_name:=who+'----';
  1:read_name:=who+'---';
  2:read_name:=who+'--';
  3:read_name:=who+'-';
  4:read_name:=who;
  else
   begin
    read_name:=copy(who,1,4);
   end;
  end;  { end of case }

 End;

{_____________________________________________________________}
```

```
{ The purpose of this matrix is to show, in a node by node basis }
{ the contents of the matrix. The user can only see what the     }
{ contents are. If any modification is desired, then he will     }
{ have to go to the editing mode.                                }

PROCEDURE Check_Matrix_Parameters;

var e :string [1];
    value,i ,j : integer;
    answer,n : string[1];
    continue, number_ok : boolean;
    w : char;

Begin

 If ( color_terminal = true ) then
  begin
   TextColor(15);
   TextBackground(1);
  end;
 Window(1,1,80,25);
 ClrScr;
 n:=' ';
 writeln;
 writeln(' You can see now the different "columns", of the ');
 writeln(' matrix, defined on a per node basis.');
 writeln;
 write(' Press <CR> or Q to quit.');
 while not keypressed do;
 read(kbd,w);
 ClrScr;
 w:=upcase(w);
 If ( w <> 'Q') then
   continue:=true;
 while continue do
  begin
   repeat
    writeln;
    ClrScr;
    writeln;
    writeln;
    writeln(' These are the ASN''s of system ',system_name,' .');
    writeln;
    For i:=1 to number_of_nets do
     begin
      writeln('  Net number ',i:2,' is : ',ASN_label[i]);
      If ( i = 18 ) then
       begin
        writeln;
        write('      Press <CR> to continue.');
        readln;
        ClrScr;
       end;
     end;
    writeln;
    write(' Enter number of matrix to check . ');
    value:=read_number;
    net_number:=value;
    ClrScr;
    If ((net_number>0) and (net_number<=number_of_nets))
     then number_ok:=true
     else number_ok:=false;
    writeln;
    ClrScr;
   until number_ok;
   number_of_events:=asn_matrix[net_number].number_of_events;
   number_of_nodes:=asn_matrix[net_number].number_of_nodes;
 If ( color_terminal = true ) then
  begin
```

```
    TextColor(0);
    TextBackground(6);
   end;
  Window(56,1,80,25);
  ClrScr;
  GotoXY(56,1);
  writeln;
  writeln('    NODE DESCRIPTION.');
  writeln('  --------------------');
  writeln;
  writeln(' A - Activity Initiator.');
  writeln;
  writeln(' B - Branch.     ');
  writeln;
  writeln(' C - Timer / Counter.');
  writeln;
  writeln(' M - Merge.      ');
  writeln;
  writeln(' W - Wait.       ');
  writeln;
  writeln(' G - Gate.       ');
  writeln;
  writeln(' T - Terminator / End. ');
  writeln;
  writeln;
  writeln('    EVENT DESCRIPTION.');
  writeln('  --------------------');
  writeln;
  writeln(' E - External Event.');
  writeln;
  If ( color_terminal = true ) then
   begin
    TextColor(15);
    TextBackground(1);
   end;
  Window(1,1,55,25);
  ClrScr;
{ The actual description is achieved by means of the following   }
{ FOR loop.                                                      }
  For node_number:=1 to number_of_nodes  do
   begin
    n:=type_of_node[net_number,node_number];
    writeln;
    write (' NODE ',node_number:2,' has these triggers:');
    writeln ('   NODE ',node_number:2,' is a ',n,' node.');
    writeln;
    j:=node_number;
    For event_number:=1 to number_of_events  do
      begin
       i:=event_number;
       e:=type_of_event[net_number,event_number];
       write ('  Event ',event_number:2,'is ',asn_matrix[net_number].
                      matrix [i,j]);
       writeln('; -- Event ',event_number:2,' is a ',e,' trigger.');
      end;
      readln;
      ClrScr;
   end;
   writeln;
   writeln('   Press (CR).');
   readln;
   ClrScr;
   writeln;
   write(' Do you want to check an other matrix ? (Y/N)');
   read(kbd,answer);
   answer:=upcase(answer);
   If answer = 'N' then continue:=false;
  end;

End;
```

```
{_____________________________________________________________}

PROCEDURE List_System_Parameters;

var i : integer;
    answer : string[1];

Begin

{ We want to list the parameters specific to this system. That   }
{ includes the activities, the external events and the Activity  }
{ Sequence Nets. In this procedure we can also request to see    }
{ the matrix parameters of a particular ASN. This is done by the }
{ procedure CHECK_MATRIX_PARAMETERS.                             }
{ We first show the ASN's.                                       }
  If (color_terminal = true ) then
   begin
    TextColor(15);
    TextBackground(1);
   end;

  Window(1,1,80,25);
  ClrScr;
  Read_system_labels;
  Read_system_definition;
  Read_system_characters;
  ClrScr;
  writeln;
  writeln;
  writeln(' These are the ASN''s of system ',system_name,' .');
  writeln;

  For i:=1 to number_of_nets do
   begin
    writeln('  Net number ',i:2,' is : ',ASN_label[i]);
    If ( i = 18 ) then
     begin
      writeln;
      write('      Press <CR> to continue.');
      readln;
      ClrScr;
     end;
   end;
   writeln;
   write('                 Press <CR>.');
   readln;
{ We then continue to see the external events of the system.     }
   ClrScr;
   writeln;
   writeln;
   writeln(' These are the external events of this system. ');
   writeln;
   For i:=1 to number_of_ext_events do
    begin
     writeln('  External event ',i:2,' is : ',ext_event_label[i]);
     If ( i = 18 ) then
      begin
       writeln;
       write('      Press <CR> to continue.');
       readln;
       ClrScr;
      end;
    end;
    writeln;
    write('                 Press <CR>.');
    readln;

{ Finally, we list the activity set of the system.               }
    ClrScr;
```

```
    writeln;
    writeln;
    writeln(' These are the activities of system ',system_name,' .');
    writeln;
    For i:=1 to number_of_activities do
     begin
      writeln('  Activity ',i:2,' is : ',activity_label[i]);
      If ( i = 18 ) then
       begin
        writeln;
        write('      Press <CR> to continue.');
        readln;
        ClrScr;
       end;
     end;

{ At this point we provide the possibility of showing the        }
{ contents of the last ASN, on a per node basis.                 }
     writeln;
     writeln('     At this point you can:');
     writeln;
     writeln('      - Check the parameters of a matrix.');
     writeln('      - Return to main Menu.');
     writeln;
     write(' Do you want to check the parameters of a matrix (Y/N) ? ');
     read(kbd,answer);
     if (( answer = 'Y') or ( answer ='y')) then
        Check_matrix_parameters;

End;

{_____________________________________________________________}
```

```
    { Node Definition }

PROCEDURE Adjust_nodes_down(node_number_to_delete:integer);

var j: integer;

Begin
 number_of_nodes:= asn_matrix[net_number].number_of_nodes;
 For j:=node_number_to_delete to number_of_nodes-2 do
  begin
    asn_node_definition[net_number,j].node
        := asn_node_definition[net_number,j+1].node;
    type_of_node[net_number,j]:=
         type_of_node[net_number,j+1];
    node:=asn_node_definition[net_number,j].node;
{ At this point we must also make sure that the  information     }
{ regarding all types of nodes is incremented. }
    case node of
      merge:;
      wait:;
      gate:;
      branch:
       begin
        asn_node_definition[net_number,j].branch_variable_id
             :=asn_node_definition[net_number,j+1].branch_variable_id;
       end;
      counter:
       begin
        asn_node_definition[net_number,j].counter_variable_id
             :=asn_node_definition[net_number,j+1].counter_variable_id;
       end;
       activity_initiator:
        begin
         activity_id[net_number,j]:=activity_id[net_number,j+1];
         asn_node_definition[net_number,j].activity_id
             :=asn_node_definition[net_number,j+1].activity_id;
        end;
       terminator:;
     end;   { end of case }
  end;

End;

{_____________________________________________________________}

PROCEDURE Adjust_nodes_up(node_position:integer);

var j, number : integer;

Begin

 j:=(asn_matrix[net_number].number_of_nodes);
 while (j > node_position) do
  begin
   asn_node_definition[net_number,j].node
       := asn_node_definition[net_number,j-1].node;
   type_of_node[net_number,j]:=type_of_node[net_number,j-1];
   node:=asn_node_definition[net_number,j-1].node;
{ At this point we must also make sure that the  information     }
{ regarding all types of nodes is incremented. }

   case node of
     merge:;
     wait:;
     gate:;
     branch:
      begin
       asn_node_definition[net_number,j].branch_variable_id
            :=asn_node_definition[net_number,j-1].branch_variable_id;
```

```
      end;
     counter:
      begin
       asn_node_definition[net_number,j].counter_variable_id
            :=asn_node_definition[net_number,j-1].counter_variable_id;
      end;
      activity_initiator:
       begin
        activity_id[net_number,j]:=activity_id[net_number,j-1];
        asn_node_definition[net_number,j].activity_id
            :=asn_node_definition[net_number,j-1].activity_id;
       end;
      terminator:;
    end;   { end of case }
{ We must now get ready to move the next node up.                }
   j:=j-1;
 end;

End;

{_____________________________________________________________}

PROCEDURE Delete_node;

var k, value, number, i ,j ,node_number_to_delete,m : integer;
    answer : string[1];
    who : string [4];

Begin
 Rough_Window;
 number_of_nodes:=asn_matrix[net_number].number_of_nodes;
 number_of_events:=asn_matrix[net_number].number_of_events;
 ClrScr;
 writeln;
 writeln(' There are ',number_of_nodes-1:2,' nodes in this net.');
 write (' Enter number of node to delete: ');
 value:=read_number;
 number:=value;
 while ( number > number_of_nodes-1 ) or ( number <= 0 ) do
  begin
   writeln (' Invalid number.');
   write (' Reenter number of node to delete: ');
   value:=read_number;
   number:=value;
  end;
 node_number_to_delete:= number;

 { 1st. case: Node to delete is a branch node.                   }
 If (asn_node_definition[net_number,node_number_to_delete].
     node = Branch) then
   begin
    For k:=1 to number_of_events do
     begin
      If (asn_event_definition[net_number,k].
          node_number = node_number_to_delete) then
        event_number:=k;
     end;
    i:=event_number;
    Adjust_events_down_by_one(i);
    If ( (i+2) < number_of_events ) then
     begin
      For k:=1 to number_of_events do
       begin
        If (asn_event_definition[net_number,k].
           node_number = node_number_to_delete) then
         event_number:=k;
       end;
      Adjust_events_down_by_one(i);
     end;
    ClrScr;
```

```
    Adjust_nodes_down(node_number_to_delete);
   end
   else
{ 2nd. case: Node to delete is not an activity initiator node    }
{             and not a terminator node.                         }
    If ((asn_node_definition[net_number,
       node_number_to_delete].node <> activity_initiator) and
        (asn_node_definition[net_number,node_number_to_delete].
         node <> terminator )) then
     begin
      For k:=1 to number_of_events do
       begin
        If (asn_event_definition[net_number,k].node_number
              = node_number_to_delete) then
         event_number:=k;
       end;
      i:=event_number;
      If ( (i+1) <= number_of_events) then
            Adjust_events_down_by_one(i);
      ClrScr;
      Adjust_nodes_down(node_number_to_delete);
     end
     else
{ 3rd.case: Node to delete is an activity initiator node.        }
      If (asn_node_definition[net_number,node_number_to_delete].
            node = activity_initiator) then
       begin
        For k:=1 to number_of_events do
         begin
          If (asn_event_definition[net_number,k].node_number
               = node_number_to_delete) then
            event_number:=k;
         end;
        i:=event_number;
        ClrScr;
        number:=asn_node_definition[net_number,
              node_number_to_delete].activity_id;
        who:= activity_label[number];
        If ((i+1)<=number_of_events) then
              Adjust_events_down_by_one(i);
        Adjust_nodes_down(node_number_to_delete);
        ClrScr;
       end
      else
{ 4th case  : node to delete is an end node. ( terminator )     }
        Adjust_nodes_down(node_number_to_delete);

      asn_matrix[net_number].number_of_nodes:=
            asn_matrix[net_number].number_of_nodes-2;
      If (asn_node_definition[net_number,node_number_to_delete].
            node = branch) then
        asn_matrix[net_number].number_of_events:=
              asn_matrix[net_number].number_of_events-1;
      If (asn_node_definition[net_number,node_number_to_delete].
            node <> terminator ) then
        asn_matrix[net_number].number_of_events:=
              asn_matrix[net_number].number_of_events-2;

      GotoXY(1,1);
      Display_matrix;
      GotoXY(58,1);
      Description_window;

      asn_matrix[net_number].number_of_nodes:=
            asn_matrix[net_number].number_of_nodes+1;
      If (asn_node_definition[net_number,node_number_to_delete].
            node <> terminator ) then
        asn_matrix[net_number].number_of_events:=
            asn_matrix[net_number].number_of_events+1;
```

```
    End;

{_____________________________________________________________}

{ This file contains the procedures to allow the entering of     }
{ nodes in the ASN's. One procedure for each of the possible     }
{ types of nodes available .                                     }

PROCEDURE Enter_activity_initiator_node;

var node_position, event_position:integer;
    number, m, h, value : integer;
    who : string [4];
    new_activity, found: boolean;
    answer: string[1];

Begin
 Red_window;
 Rough_window;
 ClrScr;
 writeln;
 write (' Enter activity identity [4 characters].');
 who:=read_name;
 For h := 1 to number_of_activities do
  begin
   if (activity_label[h] =who) then
    begin
     ClrScr;
     writeln;
     writeln(' WARNING!... activity ',who,' already exists.');
     write(' Reenter activity identity.');
     who:=read_name;
    end;
  end;
 found:=false;
 new_activity:=true;
 For m:=1 to number_of_activities do
 If (activity_label[m] = who ) then
  begin
   found:=true;
   activity_number:=m;
   new_activity:=false;
  end;

 if not found then
    activity_number:=number_of_activities;
  number_of_nodes:=asn_matrix[net_number].number_of_nodes;
  number_of_events:=asn_matrix[net_number].number_of_events;
  Rough_window;
  ClrScr;
  writeln;
  write('  Do you want to add node at end of list ? (Y/N)');
  while not keypressed do;
  read(kbd,answer);
  answer:=upcase(answer);

  If (answer ='Y') then
   begin
    For h:=1 to number_of_events do
    begin
     number:=asn_event_definition[net_number,h].node_number;
     if (number =node_position) then
        event_position:=h;
    end;
  node_number:=number_of_nodes;
  asn_node_definition[net_number,node_number].
             node:=activity_initiator;
  asn_node_definition[net_number,node_number].
             activity_id:=activity_number;
  activity_id[net_number,node_number]:=
```

```
          activity_number;
activity_label[activity_number]:=who;
type_of_node[net_number,node_number]:='A';
event_number:=number_of_events;
asn_event_definition[net_number,event_number].
            event :=int_event;
asn_event_definition[net_number,event_number].
            node_number :=node_number;
asn_event_definition[net_number,event_number].
            node :=activity_initiator;
type_of_event[net_number,event_number]:='A';

GotoXY(58,1);
Description_window;
GotoXY(1,1);
Display_matrix;
Rough_window;
GotoXY(3,20);
ClrScr;

asn_matrix[net_number].number_of_nodes:=
     (asn_matrix[net_number].number_of_nodes)+1;
asn_matrix[net_number].number_of_events:=
     (asn_matrix[net_number].number_of_events)+1;
If new_activity then
   number_of_activities:=number_of_activities+1;

number_of_nodes:=asn_matrix[net_number].number_of_nodes;
If (number_of_nodes > max_number_of_nodes_per_net) then
 begin
  ClrScr;
  writeln;
  writeln('WARNING!... No more nodes in this net can be accepted.');
  readln;
 end;

If (number_of_activities > max_number_of_activities) then
 begin
  ClrScr;
  writeln;
  writeln ('WARNING!... No more activities can be accepted.');
  readln;
 end;

number_of_events:=asn_matrix[net_number].number_of_events;
If (number_of_events >= max_number_of_events_per_net) then
 begin
  ClrScr;
  writeln;
  writeln('WARNING!... No more events can be accepted.');
  readln;
 end;

ClrScr;
end
else
 begin
  ClrScr;
  writeln;
  writeln(' Node positions available where you can put');
  writeln(' a node in this ASN are 1 to ',number_of_nodes-2:2,' .');
  writeln;
  write(' Enter position where you want to put activity: ');
  value:=read_number;
  node_position:=value;

  while ( node_position <= 0) or
              (node_position > number_of_nodes-2) do
   begin
    writeln(' Invalid node position.');
```

```
    write (' Reenter position of activity node : ');
    value:=read_number;
    node_position:=value;
   end;
  For h:=1 to number_of_events do
   begin
    number:=asn_event_definition[net_number,h].node_number;
    If ( number=node_position) then
       event_position:=h;
   end;
  Adjust_nodes_up(node_position);
  node_number:=node_position;
  asn_node_definition[net_number,node_number].
                node:=activity_initiator;
  asn_node_definition[net_number,node_number].
              activity_id:=activity_number;
  activity_id[net_number,node_number]:=
                activity_number;
  activity_label[activity_number]:=who;
  type_of_node[net_number,node_number]:='A';

  Adjust_events_up_by_1(node_position,event_position);
  event_number:=event_position;
  asn_event_definition[net_number,event_number].
                node:=activity_initiator;
  asn_event_definition[net_number,event_number].
                event:=int_event;
  asn_event_definition[net_number,event_number].
              node_number:=node_number;
  type_of_event[net_number,event_number]:='A';

  If new_activity then
       number_of_activities:=number_of_activities+1;

GotoXY(50,1);
Description_window;
GotoXY(1,1);
Display_matrix;
Rough_window;
GotoXY(3,20);
ClrScr;
asn_matrix[net_number].number_of_nodes:=
        (asn_matrix[net_number].number_of_nodes)+1;
asn_matrix[net_number].number_of_events:=
        (asn_matrix[net_number].number_of_events)+1;

number_of_nodes:=asn_matrix[net_number].number_of_nodes;
If (number_of_nodes > max_number_of_nodes_per_net) then
 begin
  ClrScr;
  writeln;
  writeln('WARNING!... No more nodes in this net can be accepted.');
  readln;
 end;

If (number_of_activities > max_number_of_activities) then
 begin
  ClrScr;
  writeln;
  writeln ('WARNING!... No more activities can be accepted.');
  readln;
 end;

number_of_events:=asn_matrix[net_number].number_of_events;
If (number_of_events >= max_number_of_events_per_net) then
 begin
  ClrScr;
  writeln;
  writeln('WARNING!... No more events can be accepted.');
  readln;
```

```
 no_more_events:=true;
end
else no_more_events:=false;

If no_more_events = false then
begin
 event_number:=asn_matrix[net_number].number_of_events;
 asn_event_definition[net_number,event_number].
             node:=branch;
 asn_event_definition[net_number,event_number].
             event:=int_event;
 asn_event_definition[net_number,event_number].
             node_number:=node_number;
 type_of_event[net_number,event_number]:='B';
 branch_cond_var[net_number,event_number]:='F';
 asn_event_definition[net_number,event_number].
  branch_condition:=false;

 GotoXY(58,1);
 Description_window;
 GotoXY(1,1);
 Display_matrix;
 asn_matrix[net_number].number_of_events:=
         (asn_matrix[net_number].number_of_events)+1;
 asn_matrix[net_number].number_of_nodes:=
         (asn_matrix[net_number].number_of_nodes)+1;
 Rough window;
 GotoXY(3,20);
 ClrScr;

 number_of_nodes:=asn_matrix[net_number].number_of_nodes;
 If (number_of_nodes > max_number_of_nodes_per_net) then
  begin
   ClrScr;
   writeln;
   writeln('WARNING!... No more nodes in this net can be accepted.');
   readln;
  end;

 number_of_events:=asn_matrix[net_number].number_of_events;
  If (number_of_events >= max_number_of_events_per_net) then
   begin
    ClrScr;
    writeln;
    writeln('WARNING!... No more events can be accepted.');
    readln;
   end;
 end;
 end
 else
  begin
   ClrScr;
   writeln;
   writeln(' Node positions available where you can put');
   writeln(' a node in this ASN are 1 to ',number_of_nodes-2:2,' .');
   writeln;
   write (' Enter position where you want to put branch node: ');
   value:=read_number;
   node_position:=value;
   while (node_position <= 0) or
      (node_position > number_of_nodes-2) do
    begin
     writeln(' Invalid node position.');
     write (' Reenter node position of the branch node: ');
     value:=read_number;
     node_position:=value;
    end;
   For h:=1 to number_of_events do
    begin
     number:=asn_event_definition[net_number,h].node_number;
```

```
   end;

 end;

End;

{_____________________________________________________________}

PROCEDURE Enter_branch_node;

var node_position,event_position:integer;
    no_more_events : boolean;
    h, number, value : integer;
    answer: string[1];

Begin

 Red_window;
 Rough_window;
 ClrScr;
 writeln;
 number_of_nodes:=asn_matrix[net_number].number_of_nodes;
 number_of_events:=asn_matrix[net_number].number_of_events;

 writeln;
 answer:='y';
 write(' Do you want to add node at end of list of nodes (Y/N) ?');
 while not keypressed do ;
 read (kbd,answer);
 answer:=upcase(answer);
 If (answer ='Y') then
  begin
{ We now define the parameters describing the node.              }
   node_number:=number_of_nodes;
   asn_node_definition[net_number,node_number].
                node:=branch;
   type_of_node[net_number,node_number]:='B';
   event_number:=number_of_events;
   asn_event_definition[net_number,event_number].
                node:=branch;
   asn_event_definition[net_number,event_number].
                event:=int_event;
   asn_event_definition[net_number,event_number].
                node_number:=node_number;
   type_of_event[net_number,event_number]:='B';
   ClrScr;
   writeln;
   write(' Enter branch condition variable [1 character] : ');
   readln(entry);
{ Here, it is where the condition parameters are specified.      }
   asn_node_definition[net_number,node_number].
       branch_variable_id:=entry;
   branch_cond_var[net_number,event_number]:='T';
   asn_event_definition[net_number,event_number].
     branch_condition:=true;

{ We get ready to enter the definition of the second event       }
{ derived from this branch event.                                }
   asn_matrix[net_number].number_of_events:=
    (asn_matrix[net_number].number_of_events)+1;
   Rough_window;
   GotoXY(3,20);
   ClrScr;
   number_of_events:=asn_matrix[net_number].number_of_events;
   If (number_of_events >= max_number_of_events_per_net) then
    begin
     writeln;
     writeln(' WARNING!... No more events can be accepted.');
     writeln('   The second event part of this branch node  ');
     writeln('         is not defined.');
```

```
        if (number = node_position) then
              event_position:=h;
       end;

      Adjust_nodes_up(node_position);
      node_number:=node_position;
      asn_node_definition[net_number,node_number].node:=branch;
      type_of_node[net_number,node_number]:='B';
      ClrScr;
      writeln;
      write(' Enter branch condition variable [1 character] : ');
      readln(entry);
{ Here, it is where the condition parameters are specified.      }
      asn_node_definition[net_number,node_number].branch_variable_id:=entry;
      Adjust_events_up_by_1(node_position,event_position);
      event_number:=event_position;
      asn_event_definition[net_number,event_number].node:=branch;
      asn_event_definition[net_number,event_number].event:=int_event;
      asn_event_definition[net_number,event_number].node_number:=node_number;
      type_of_event[net_number,event_number]:='B';
      branch_cond_var[net_number,event_number]:='T';
      asn_event_definition[net_number,event_number].branch_condition:=true;
      asn_matrix[net_number].number_of_events:=
             (asn_matrix[net_number].number_of_events)+1;
      number_of_nodes:=asn_matrix[net_number].number_of_nodes;
      If (number_of_nodes > max_number_of_nodes_per_net) then
       begin
        ClrScr;
        writeln;
        writeln('WARNING!... No more nodes in this net can be accepted.');
        readln;
       end;
      number_of_events:=asn_matrix[net_number].number_of_events;
      If (number_of_events >= max_number_of_events_per_net) then
       begin
        ClrScr;
        writeln;
        writeln('WARNING!... No more events can be accepted.');
        readln;
       end;

      Adjust_events_up_by_1(node_position,event_position);
      event_number:=event_number+1;
      asn_event_definition[net_number,event_number].node:=branch;
      asn_event_definition[net_number,event_number].event:=int_event;
      asn_event_definition[net_number,event_number].node_number:=node_number;
      type_of_event[net_number,event_number]:='B';
      branch_cond_var[net_number,event_number]:='F';
      asn_event_definition[net_number,event_number].branch_condition:=false;

      GotoXY(58,1);
      Description_window;
      GotoXY(1,1);
      Display_matrix;
      Rough_window;
      GotoXY(3,20);
      ClrScr;
      asn_matrix[net_number].number_of_nodes:=
            (asn_matrix[net_number].number_of_nodes)+1;
      asn_matrix[net_number].number_of_events:=
            (asn_matrix[net_number].number_of_events)+1;

      number_of_events:=asn_matrix[net_number].number_of_events;
      If (number_of_events >= max_number_of_events_per_net) then
       begin
        ClrScr;
        writeln;
        writeln('WARNING!... No more events can be accepted.');
        readln;
       end;
```

```
    end;

End;

{_____}

PROCEDURE Enter_merge_node;

var node_position,event_position:integer;
    h, number, value :integer;
    answer : string [1];

Begin

 Red_window;
 Rough_window;
 ClrScr;
 writeln;
 number_of_nodes:=asn_matrix[net_number].number_of_nodes;
 number_of_events:=asn_matrix[net_number].number_of_events;
 answer:='y';
 writeln;
 write(' Do you want to add merge node at end of list (Y/N) ?');
 while not keypressed do ;
 read (kbd,answer);

 answer:=upcase(answer);
 If (answer ='Y') then
  begin
   node_number:=number_of_nodes;
   asn_node_definition[net_number,node_number].node:=merge;
   type_of_node[net_number,node_number]:='M';
   event_number:=number_of_events;
   asn_event_definition[net_number,event_number].node:=merge;
   asn_event_definition[net_number,event_number].event:=int_event;
   asn_event_definition[net_number,event_number].node_number:=node_number;
   type_of_event[net_number,event_number]:='M';

   GotoXY(58,1);
   Description_window;
   GotoXY(1,1);
   Display_matrix;

   asn_matrix[net_number].number_of_nodes:=
        (asn_matrix[net_number].number_of_nodes)+1;
   asn_matrix[net_number].number_of_events:=
        (asn_matrix[net_number].number_of_events)+1;

   Rough_window;
   GotoXY(3,20);
   ClrScr;

   number_of_nodes:=asn_matrix[net_number].number_of_nodes;
    If (number_of_nodes > max_number_of_nodes_per_net) then
     begin
      ClrScr;
      writeln;
      writeln('WARNING!... No more nodes in this net can be accepted.');
      readln;
     end;

    number_of_events:=asn_matrix[net_number].number_of_events;
     If (number_of_events >= max_number_of_events_per_net) then
      begin
       ClrScr;
       writeln;
       writeln('WARNING!... No more events can be accepted.');
       readln;
      end;
 end
```

```
 else
  begin
   ClrScr;
   writeln;
   writeln(' Node positions available where you can put');
   writeln(' a node in this ASN are 1 to ',number_of_nodes-2:2,' .');
   writeln;
   write (' Enter position where you want to put merge node: ');
   value:=read_number;
   node_position:=value;

   while (node_position <= 0) or
      (node_position > number_of_nodes-2) do
      begin
       ClrScr;
       writeln;
       writeln(' Invalid node position.');
       write (' Reenter node position of the merge node: ');
       value:=read_number;
       node_position:=value;
      end;

For h:=1 to number_of_events do
  begin
   number:=asn_event_definition[net_number,h].node_number;
   if (number = node_position) then
    event_position:=h;
  end;

  Adjust_nodes_up(node_position);
  node_number:=node_position;
  asn_node_definition[net_number,node_number].node:=merge;
  type_of_node[net_number,node_number]:='M';

  Adjust_events_up_by_1(node_position,event_position);
  event_number:=event_position;
  asn_event_definition[net_number,event_number].node:=merge;
  asn_event_definition[net_number,event_number].event:=int_event;
  asn_event_definition[net_number,event_number].node_number:=node_number;
  type_of_event[net_number,event_number]:='M';

  GotoXY(58,1);
  Description_window;
  GotoXY(1,1);
  Display_matrix;
  Rough_window;
  GotoXY(3,20);
  ClrScr;

  asn_matrix[net_number].number_of_nodes:=
      (asn_matrix[net_number].number_of_nodes)+1;
  asn_matrix[net_number].number_of_events:=
      (asn_matrix[net_number].number_of_events)+1;

  number_of_nodes:=asn_matrix[net_number].number_of_nodes;
   If (number_of_nodes > max_number_of_nodes_per_net) then
    begin
     ClrScr;
     writeln;
     writeln('WARNING!... No more nodes in this net can be accepted.');
     readln;
    end;

   number_of_events:=asn_matrix[net_number].number_of_events;
    If (number_of_events >= max_number_of_events_per_net) then
     begin
      ClrScr;
      writeln;
      writeln('WARNING!... No more events can be accepted.');
      readln;
```

```
      end;

   end;

End;

{_____________________________________________________________}

PROCEDURE Enter_counter_node;

var node_position,event_position:integer;
    answer, character : string[1];
    h , number, value : integer;

Begin

 Red_window;
 Rough_window;
 ClrScr;
 writeln;
 writeln(' WARNING ! : MAKE SURE THE ASN DOES NOT HANG UP ');
 writeln(' ---------   BECAUSE OF THE END NODE NOT BEING  ');
 writeln('             REACHED WHEN COUNTER DISABLED.');
 write(' PRESS <CR>.');
 readln;
 Rough_window;
 ClrScr;
 number_of_nodes:=asn_matrix[net_number].number_of_nodes;
 number_of_events:=asn_matrix[net_number].number_of_events;

 writeln;
 answer:='y';
 write(' Do you want to add node at end of list of nodes (Y/N) ?');
 while not keypressed do ;
 read (kbd,answer);

 answer:=upcase(answer);
 If (answer ='Y') then
  begin
   node_number:=number_of_nodes;
   asn_node_definition[net_number,node_number].node:=counter;
   type_of_node[net_number,node_number]:='C';
   event_number:=number_of_events;
   asn_event_definition[net_number,event_number].node:=counter;
   asn_event_definition[net_number,event_number].event:=int_event;
   asn_event_definition[net_number,event_number].node_number:=node_number;
   type_of_event[net_number,event_number]:='C';

   ClrScr;
   writeln;
   write(' Enter counter variable id [1 character] : ');
   readln(character);
   counter_var[net_number,node_number]:=character;
   asn_node_definition[net_number,node_number].
        counter_variable_id:=character;

   GotoXY(50,1);
   Description_window;
   GotoXY(1,1);
   Display_matrix;

   asn_matrix[net_number].number_of_nodes:=
      (asn_matrix[net_number].number_of_nodes)+1;
   asn_matrix[net_number].number_of_events:=
      (asn_matrix[net_number].number_of_events)+1;

   Rough_window;
   GotoXY(3,20);
   ClrScr;
```

```
number_of_nodes:=asn_matrix[net_number].number_of_nodes;
 If (number_of_nodes > max_number_of_nodes_per_net) then
  begin
   ClrScr;
   writeln;
   writeln('WARNING!... No more nodes in this net can be accepted.');
   readln;
  end;

 number_of_events:=asn_matrix[net_number].number_of_events;
  If (number_of_events >= max_number_of_events_per_net) then
   begin
    ClrScr;
    writeln;
    writeln('WARNING!... No more events can be accepted.');
    readln;
   end;
end
else
 begin
  ClrScr;
  writeln;
  writeln(' Node positions available where you can put');
  writeln(' a node in this ASN are 1 to ',number_of_nodes-2:2,' .');
  writeln;
  write (' Enter position where you want to put counter node: ');
  value:=read_number;
  node_position:=value;
  while (node_position <= 0) or
     (node_position > number_of_nodes-2) do
    begin
     writeln(' Invalid node position.');
     write (' Reenter node position of the counter node: ');
     value:=read_number;
     node_position:=value;
    end;

 For h:=1 to number_of_events do
  begin
   number:=asn_event_definition[net_number,h].node_number;
   if (number = node_position) then
    event_position:=h;
  end;

 Adjust_nodes_up(node_position);
 node_number:=node_position;
 asn_node_definition[net_number,node_number].node:=counter;
 type_of_node[net_number,node_number]:='C';
 ClrScr;
 writeln;
 write(' Enter counter variable id [1 character] : ');
 readln(character);
 counter_var[net_number,node_number]:=character;
 asn_node_definition[net_number,node_number].
     counter_variable_id:=character;

 Adjust_events_up_by_1(node_position,event_position);
 event_number:=event_position;
 asn_event_definition[net_number,event_number].node:=counter;
 asn_event_definition[net_number,event_number].event:=int_event;
 asn_event_definition[net_number,event_number].node_number:=node_number;
 type_of_event[net_number,event_number]:='C';

 GotoXY(58,1);
 Description_window;
 GotoXY(1,1);
 Display_matrix;
 Rough_window;
 GotoXY(3,20);
 ClrScr;
```

```
  asn_matrix[net_number].number_of_nodes:=
      (asn_matrix[net_number].number_of_nodes)+1;
  asn_matrix[net_number].number_of_events:=
      (asn_matrix[net_number].number_of_events)+1;

  number_of_nodes:=asn_matrix[net_number].number_of_nodes;
   If (number_of_nodes > max_number_of_nodes_per_net) then
    begin
     ClrScr;
     writeln;
     writeln('WARNING!... No more nodes in this net can be accepted.');
     readln;
    end;

   number_of_events:=asn_matrix[net_number].number_of_events;
    If (number_of_events >= max_number_of_events_per_net) then
     begin
      ClrScr;
      writeln;
      writeln('WARNING!... No more events can be accepted.');
      readln;
     end;
  end;

End;

{_____________________________________________________________}

PROCEDURE Enter_gate_node;

var node_position,event_position:integer;
    h, number, value : integer;
    answer : string [1];

Begin
 Red_window;
 Rough_window;
 ClrScr;
 writeln;
 writeln(' WARNING ! : MAKE SURE THE ASN DOES NOT HANG UP ');
 writeln(' ---------   BECAUSE OF THE END NODE NOT BEING  ');
 writeln('             REACHED WHEN GATE DISABLED.');
 write(' PRESS <CR>.');
 readln;
 Rough_window;
 ClrScr;
 number_of_nodes:=asn_matrix[net_number].number_of_nodes;
 number_of_events:=asn_matrix[net_number].number_of_events;
 writeln;
 answer:='y';
 write(' Do you want to add node at end of list of nodes (Y/N) ?');
 while not keypressed do ;
 read (kbd,answer);
 answer:=upcase(answer);
 If (answer ='Y') then
  begin
   node_number:=number_of_nodes;
   asn_node_definition[net_number,node_number].node:=gate;
   type_of_node[net_number,node_number]:='G';
   event_number:=number_of_events;
   asn_event_definition[net_number,event_number].node:=gate;
   asn_event_definition[net_number,event_number].event:=int_event;
   asn_event_definition[net_number,event_number].node_number:=node_number;
   type_of_event[net_number,event_number]:='G';

   GotoXY(58,1);
   Description_window;
   GotoXY(1,1);
   Display_matrix;
```

```
 asn_matrix[net_number].number_of_nodes:=
     (asn_matrix[net_number].number_of_nodes)+1;
 asn_matrix[net_number].number_of_events:=
     (asn_matrix[net_number].number_of_events)+1;

 Rough_window;
 GotoXY(3,20);
 ClrScr;

 number_of_nodes:=asn_matrix[net_number].number_of_nodes;
  If (number_of_nodes > max_number_of_nodes_per_net) then
   begin
    ClrScr;
    writeln;
    writeln('WARNING!... No more nodes in this net can be accepted.');
    readln;
   end;

  number_of_events:=asn_matrix[net_number].number_of_events;
   If (number_of_events >= max_number_of_events_per_net) then
    begin
     ClrScr;
     writeln;
     writeln('WARNING!... No more events can be accepted.');
     readln;
    end;

end
else
 begin
  writeln;
  writeln(' Node positions available where you can put');
  writeln(' a node in this ASN are 1 to ',number_of_nodes-2:2,' .');
  writeln;
  write (' Enter position where you want to put gate node: ');
  value:=read_number;
  node_position:=value;
  while (node_position <= 0) or
     (node_position > number_of_nodes-2) do
     begin
      ClrScr;
      writeln;
      writeln(' Invalid node position.');
      write (' Reenter node position of the gate node: ');
      value:=read_number;
      node_position:=value;
     end;

 For h:=1 to number_of_events do
  begin
   number:=asn_event_definition[net_number,h].node_number;
   if (number = node_position) then
    event_position:=h;
  end;

 Adjust_nodes_up(node_position);
 node_number:=node_position;
 asn_node_definition[net_number,node_number].node:=gate;
 type_of_node[net_number,node_number]:='G';

 Adjust_events_up_by_1(node_position,event_position);
 event_number:=event_position;
 asn_event_definition[net_number,event_number].node:=gate;
 asn_event_definition[net_number,event_number].event:=int_event;
 asn_event_definition[net_number,event_number].node_number:=node_number;
 type_of_event[net_number,event_number]:='G';

 GotoXY(50,1);
 Description_window;
```

```
 GotoXY(1,1);
 Display_matrix;
 Rough_window;
 GotoXY(3,20);
 ClrScr;

 asn_matrix[net_number].number_of_nodes:=
        (asn_matrix[net_number].number_of_nodes)+1;
 asn_matrix[net_number].number_of_events:=
        (asn_matrix[net_number].number_of_events)+1;

 number_of_nodes:=asn_matrix[net_number].number_of_nodes;
  If (number_of_nodes > max_number_of_nodes_per_net) then
    begin
     ClrScr;
     writeln;
     writeln('WARNING!... No more nodes in this net can be accepted.');
     readln;
    end;

  number_of_events:=asn_matrix[net_number].number_of_events;
    If (number_of_events >= max_number_of_events_per_net) then
     begin
      ClrScr;
      writeln;
      writeln('WARNING!... No more events can be accepted.');
      readln;
     end;

 end;

End;

{_____________________________________________________________}

PROCEDURE Enter_wait_node;

var node_position,event_position:integer;
    answer: string[1];
    h, number, value : integer;

Begin

 Red_window;
 Rough_window;
 ClrScr;
 writeln;
 number_of_nodes:=asn_matrix[net_number].number_of_nodes;
 number_of_events:=asn_matrix[net_number].number_of_events;
 answer:='y';
 write(' Do you want to add node at end of list of nodes (Y/N) ?');
 while not keypressed do ;
 read (kbd,answer);
 answer:=upcase(answer);
 If (answer ='Y') then
  begin
   node_number:=number_of_nodes;
   asn_node_definition[net_number,node_number].node:=wait;
   type_of_node[net_number,node_number]:='W';
   event_number:=number_of_events;
   asn_event_definition[net_number,event_number].node:=wait;
   asn_event_definition[net_number,event_number].event:=int_event;
   asn_event_definition[net_number,event_number].node_number:=node_number;
   type_of_event[net_number,event_number]:='W';

   GotoXY(58,1);
   Description_window;
   GotoXY(1,1);
   Display_matrix;
```

```
 asn_matrix[net_number].number_of_nodes:=
       (asn_matrix[net_number].number_of_nodes)+1;
 asn_matrix[net_number].number_of_events:=
       (asn_matrix[net_number].number_of_events)+1;

 Rough_window;
 GotoXY(3,20);
 ClrScr;

number_of_nodes:=asn_matrix[net_number].number_of_nodes;
 If (number_of_nodes > max_number_of_nodes_per_net) then
   begin
    ClrScr;
    writeln;
    writeln('WARNING!... No more nodes in this net can be accepted.');
    readln;
   end;

 number_of_events:=asn_matrix[net_number].number_of_events;
   If (number_of_events >= max_number_of_events_per_net) then
    begin
     ClrScr;
     writeln;
     writeln('WARNING!... No more events can be accepted.');
     readln;
    end;

end
else
 begin
  writeln;
  writeln(' Node positions available where you can put');
  writeln(' a node in this ASN are 1 to ',number_of_nodes-2:2,' .');
  writeln;
  write (' Enter position where you want to put wait node: ');
  value:=read_number;
  node_position:=value;
  while (node_position <= 0) or
     (node_position > number_of_nodes-2) do
     begin
      ClrScr;
      writeln;
      writeln(' Invalid node position.');
      write (' Reenter node position of the wait node: ');
      value:=read_number;
      node_position:=value;
     end;

For h:=1 to number_of_events do
 begin
  number:=asn_event_definition[net_number,h].node_number;
  if (number = node_position) then
   event_position:=h;
 end;

 Adjust_nodes_up(node_position);
 node_number:=node_position;
 asn_node_definition[net_number,node_number].node:=wait;
 type_of_node[net_number,node_number]:='W';

 Adjust_events_up_by_1(node_position,event_position);
 event_number:=event_position;
 asn_event_definition[net_number,event_number].node:=wait;
 asn_event_definition[net_number,event_number].event:=int_event;
 asn_event_definition[net_number,event_number].node_number:=node_number;
 type_of_event[net_number,event_number]:='W';

 GotoXY(58,1);
 Description_window;
 GotoXY(1,1);
```

```
  Display_matrix;
  Rough_window;
  GotoXY(3,20);
  ClrScr;

  asn_matrix[net_number].number_of_nodes:=
      (asn_matrix[net_number].number_of_nodes)+1;
  asn_matrix[net_number].number_of_events:=
      (asn_matrix[net_number].number_of_events)+1;

  number_of_nodes:=asn_matrix[net_number].number_of_nodes;
  If (number_of_nodes > max_number_of_nodes_per_net) then
    begin
     ClrScr;
     writeln;
     writeln('WARNING!... No more nodes in this net can be accepted.');
     readln;
    end;

  number_of_events:=asn_matrix[net_number].number_of_events;
   If (number_of_events >= max_number_of_events_per_net) then
     begin
      ClrScr;
      writeln;
      writeln('WARNING!... No more events can be accepted.');
      readln;
     end;

 end;

End;

{_____________________________________________________________}

PROCEDURE Enter_terminator_node;

Begin

 asn_matrix[net_number].number_of_events:=
   (asn_matrix[net_number].number_of_events)-1;
 number_of_nodes:=asn_matrix[net_number].number_of_nodes;
 node_number:=number_of_nodes;
 asn_node_definition[net_number,node_number].node:=terminator;
 type_of_node[net_number,node_number]:='T';
 last_node_flag:=true;

 GotoXY(50,1);
 Description_window;
 GotoXY(1,1);
 Display_matrix;
 Rough_window;
 GotoXY(3,20);
 ClrScr;
 number_of_nodes:=number_of_nodes+1;
 asn_matrix[net_number].number_of_nodes:=number_of_nodes;
 asn_matrix[net_number].number_of_events:=
   (asn_matrix[net_number].number_of_events)+1;

End;

{_____________________________________________________________}

PROCEDURE Add_node;

var number, event_position :integer;
    n :string[1];

Begin
 ClrScr;
 number:=asn_matrix[net_number].number_of_nodes;
```

```
node_number:=number;
If (number = max_number_of_nodes_per_net) then
 begin
  writeln;
  writeln(' WARNING...last node of a network should be ');
  writeln( '  an END node. This is the last place to add a ');
  writeln('   node.');
  writeln('                                Press <CR>');
  readln;
 end;
if (number > max_number_of_nodes_per_net+1) then
  writeln('TOO MANY NODES IN THIS ASR NET!!!..')
  else
  begin
   Display_node_types_menu;
   Red_window;
   Reduced_rough_Window;
   ClrScr;
   writeln;
   write(' Enter type of node to add: ');
   readln (n);
   n:=upcase(n);
   If (one_more_node = true) then
    begin
     while (n <>'E') do
       begin
        write (' Invalid type. Reenter node type.');
        readln (n);
       end;
      type_of_node[net_number,node_number]:=n;
     end
     else
     begin
      While ( (n <>'A') and (n <>'B') and (n <>'C') and (n <>'G') and
        (n <>'M') and (n <>'W') and (n <>'E') ) do
       begin
        ClrScr;
        writeln;
        write (' Invalid type. Reenter node type.');
        readln (n);
        n:=upcase(n);
       end;
      type_of_node[net_number,node_number]:=n;
{ At this point, the node type has been accepted.                }
       CASE n of
        'A':Enter_Activity_initiator_node;
        'B':Enter_Branch_node;
        'C':Enter_Counter_node;
        'E':Enter_Terminator_node;
        'G':Enter_Gate_node;
        'M':Enter_Merge_node;
        'W':Enter_Wait_node;
       end; {end of case}
    end;
   end;

End;

{_____________________________________________________________}

PROCEDURE Show_nodes_and_events;

var answer, sentry:string[1];
    number, number_ext, h :integer;

Begin
 last_node_flag:=false;
 one_more_node:=false;
 entry:='0';
 Rough_window;
```

```
ClrScr;
sentry:='b';
While ((sentry()'x') and (sentry()'X')) do
 begin
  menu_window;
  Rough_window;
  If (number_of_nodes=max_number_of_nodes_per_net) then
   begin
    Rough_window;
    ClrScr;
    writeln;
    writeln('    WARNING.');
    writeln(' This should be the last node.');
    write('    PRESS  <CR>');
    readln;
   end;
  ClrScr;
  writeln;
  write(' WAITING FOR COMMAND FROM MENU... ');
  while not keypressed do;     ( wait for an entry )
  read (kbd,sentry);
  sentry:=upcase(sentry);
  If ((sentry='R') or (sentry='S') or (sentry='A')
        or (sentry='X') or (sentry='V') or (sentry='C') ) then
   begin
    case sentry of
     'R': Delete_node;
     'S': Add_node;
     'V': Add_ext_event;
     'C': Delete_ext_event;
     'X':begin
          number_ext:=0;
          For h:=1 to (asn_matrix[net_number].number_of_events)-1 do
           begin
            If (asn_event_definition[net_number,h].event
                     =ext_event) then number_ext:=number_ext+1;
           end;
          If (number_ext=0) then
           begin
            Rough_window;
            Clrscr;
            writeln;
            writeln(' There are no external events in this ASN.');
            writeln(' You must enter now the starting event.');
            writeln;
            write('                        Press <CR>.');
            readln;
            Clrscr;
            Add_ext_event;
            Rough_window;
            Clrscr;
            writeln;
            write(' External event has been entered; press <CR>.');
            readln;
            sentry:='X';
           end;
          end;
     end;   { end of case }
    end;
   end;
 Red_window;
 Green_window;
 ClrScr;
 Working_window;
 ClrScr;
 Rough_window;
 ClrScr;
 writeln;
 write(' Do you wish to save the ASN just entered (Y/N) ?');
 while not keypressed do;
```

```
 read (kbd,answer);
 answer:=upcase(answer);
 number:=net_number;
 If ((answer='Y') or (answer='y')) then
  begin
   Save_System_Definition;
   ClrScr;
   Save_system_labels;
   Save_system_characters;
   ClrScr;
  end;
 net_number:=number;

End;

{_____________________________________________________________}
```

```
    ( Event Definition )

PROCEDURE Adjust_events_down_by_one(i:integer);

var j : integer;

Begin
{ The purpose of this procedure is to, in the case of deleting a  }
{ node, shift all events in order to overwrite the event that     }
{ corresponds to the node that is being deleted.                  }
{ The value of i is that of the event position corresponding to   }
{ the position of the node being deleted.                         }
 j:=i;
 number_of_events:=asn_matrix[net_number].number_of_events;
 For j:=i to number_of_events-2  do
  begin
   type_of_event[net_number,j]:=type_of_event[net_number,j+1];
   asn_event_definition[net_number,j].event
           := asn_event_definition[net_number,j+1].event;
{ If the event being shifted is an internal event, then we have   }
{ to perform the following steps.                                 }
   If (asn_event_definition[net_number,j+1].event = int_event) then
    begin
     asn_event_definition[net_number,j].node_number
          := asn_event_definition[net_number,j+1].node_number-1;
     asn_event_definition[net_number,j].node
          := asn_event_definition[net_number,j+1].node;
{ In the case of a branch node we must not forget about the       }
{ condition variable.                                             }
     If (asn_event_definition[net_number,j+1].node = branch) then
      begin
       asn_event_definition[net_number,j].branch_condition:=
            asn_event_definition[net_number,j+1].branch_condition;
       branch_cond_var[net_number,j]:=
           branch_cond_var[net_number,j+1];
      end;
    end
   else
{ This is done in the case of an external event .                 }
     begin
      external_event[net_number,j]:=external_event[net_number,j+1];
      asn_event_definition[net_number,j].event_number:=
            asn_event_definition[net_number,j+1].event_number;
     end;
  end;

End;

{_____________________________________________________________________}

PROCEDURE Adjust_events_up_by_1(node_position,
                           event_position:integer);

var h :integer;

Begin
{ The purpose of this procedure is to shift all events in order   }
{ to leave a blank row in order to insert the corresponding       }
{ information there.                                              }
 h:=(asn_matrix[net_number].number_of_events);
 While ( h > event_position ) do
  begin
   asn_event_definition[net_number,h].event:=
           asn_event_definition[net_number,h-1].event;
   type_of_event[net_number,h]:=type_of_event[net_number,h-1];
   If (asn_event_definition[net_number,h].event = int_event)
      then
    begin
     asn_event_definition[net_number,h].node:=
```

```
            asn_event_definition[net_number,h-1].node;
     asn_event_definition[net_number,h].node_number:=
            (asn_event_definition[net_number,h-1].node_number)+1;
{ In the case of a branch node we must not forget about the      }
{ condition variable.                                            }
     If (asn_event_definition[net_number,h].node = branch) then
      begin
       asn_event_definition[net_number,h].branch_condition:=
            asn_event_definition[net_number,h-1].branch_condition;
       branch_cond_var[net_number,h]:=
            branch_cond_var[net_number,h-1];
      end;
     end
     else
     If ( asn_event_definition[net_number,h].event =
       ext_event ) then
      begin
       external_event[net_number,h]:=external_event[net_number,h-1];
       asn_event_definition[net_number,h].event_number:=
            asn_event_definition[net_number,h-1].event_number;
      end;
    h:=h-1;
  end;

End;

{_____________________________________________________________}

PROCEDURE Enter_Starting_event;

var p, m, i, h : integer;
    who :string [4];
    found :boolean;

Begin             ( Enter_starting_event )
 Rough_window;
 found:=false;
 While not found do
  begin
   ClrScr;
   p:=0;
   For h:=1 to asn_matrix[net_number].number_of_events do
    begin
     If (asn_event_definition[net_number,h].event = ext_event ) then
           p:=p+1;
    end;
   writeln;
   writeln(' There are ',p:2,' ext events in this net.');
   writeln(' These are :');
   write('  ');
   For h:=1 to number_of_events-1 do
    If (asn_event_definition[net_number,h].event = ext_event) then
     begin
      m:=asn_event_definition[net_number,h].event_number;
      write(ext_event_label[m],' ,');
     end;
   writeln;
   write(' Press <CR>.');
   readln;
   Clrscr;
   writeln;
   write (' Enter starting event for ASN ',
             ASN_label[net_number],': [4 characters] ');
   who:=read_name;
   ClrScr;
   For h:=1 to number_of_ext_events do
    begin
     For i:=1 to number_of_events do
        begin
         If (external_event[net_number,i]=who) then
```

```
        If (ext_event_label[h]=who) then
         begin
          asn_starting_event[net_number]:=h;
          starting_event_label[net_number]:=who;
          found:=true;
          starting_event_entered:=true;
          ClrScr;
         end;
       end;
    end;
  If (found = false) then
   begin
    writeln;
    writeln(' Starting event must have been included in the ');
    writeln(' System specification.');
    write(' Reenter starting event : [4 characters] ');
    who:=read_name;
   end;
end;
ClrScr;
writeln;
writeln(' Starting event for ASN ',ASN_label[net_number],
     ' is ',starting_event_label[net_number]);
writeln(' This starting event is ext event number ',
    asn_starting_event[net_number]:2);
writeln;
writeln('Press <CR>.');
readln;

End;

{_____________________________________________________________}

PROCEDURE Delete_ext_event;

var j, event_position, number ,h ,m ,i ,event_number_to_delete :integer;
    who : word;
    ext_event_found : boolean;

Begin
 number_of_events:=asn_matrix[net_number].number_of_events;
 Rough_window;
 ClrScr;
 writeln;
 ext_event_found:=false;
 write(' Enter external event to delete [4 characters] : ');
 who:=read_name;
 For j:=1 to number_of_events do
  If (asn_event_definition[net_number,j].event = ext_event) then
   If (external_event[net_number,j] = who) then
    begin
     ext_event_found:=true;
     number:=j;
    end;
 If (ext_event_found) then
  begin
   event_number_to_delete:=number;
   i:=number;
   Adjust_events_down_by_one(i);
   asn_matrix[net_number].number_of_events:=
        (asn_matrix[net_number].number_of_events)-1;
  end
 else
  begin
   Rough_window;
   ClrScr;
   writeln;
   writeln(' Sorry, input data incorrect. Event not deleted.');
   writeln;
   write('  Press <CR>.');
```

```
  readln;
 end;

 asn_matrix[net_number].number_of_nodes:=
      (asn_matrix[net_number].number_of_nodes)-1;
 asn_matrix[net_number].number_of_events:=
      (asn_matrix[net_number].number_of_events)-1;

 GotoXY(50,1);
 Description_window;
 GotoXY(1,1);
 Display_matrix;

 asn_matrix[net_number].number_of_nodes:=
      (asn_matrix[net_number].number_of_nodes)+1;

End;

{_____________________________________________________________}

PROCEDURE Add_ext_event;

var who :string [4];
    not_accepted, number_ok, found, new_ext_event : boolean;
    h, m : integer;

Begin
 Rough_window;
 ClrScr;
 asn_matrix[net_number].number_of_nodes:=
     (asn_matrix[net_number].number_of_nodes)-1;
 number_of_events:=asn_matrix[net_number].number_of_events;
 event_number:=number_of_events;
 If (asn_matrix[net_number].number_of_events
           < max_number_of_events_per_net) then
  begin
   writeln;
   If (number_of_ext_events = max_number_of_ext_events) then
    begin
     Clrscr;
     writeln;
     writeln(' WARNING...Too many external events in this ASN.');
     writeln;
     write(' Press <CR>. ');
     readln;
     Clrscr;
     number_ok:=false;
    end
    else
     begin
      not_accepted:=false;
      Clrscr;
      writeln;
      write(' Enter external event label [4 characters] ');
      who:=read_name;
      Clrscr;
      For m:=1 to number_of_events do
       If (external_event[net_number,m] = who) then
        begin
         not_accepted:=true;
         Clrscr;
         writeln;
         writeln(' Event ',who,' already exists in the net.');
         write(' Event is not accepted. Press <CR>. ');
         readln;
         number_ok:=false;
         Clrscr;
        end;
      If not not_accepted then
       begin
```

```
      found:=false;
      number_ok:=true;
      new_ext_event:=true;
      For h:=1 to number_of_ext_events do
       If (ext_event_label[h]=who) then
        begin
         ext_event_number:=h;
         found:=true;
         new_ext_event:=false;
        end;
      If found = false then
         ext_event_number:=number_of_ext_events;
      external_event[net_number,event_number]:=who;
      ext_event_label[ext_event_number]:=who;
      asn_event_definition[net_number,event_number].event:=ext_event;
      asn_event_definition[net_number,event_number].event_number:=ext_event_number;
      type_of_event[net_number,event_number]:='E';
      If new_ext_event then
       begin
        number_of_ext_events:=(number_of_ext_events)+1;
        ext_event_number:=(ext_event_number)+1;
       end;
     end;
   end;
 end
 else
  begin
   writeln;
   writeln(' There are already ',event_number-1,' events.');
   writeln(' No more can be accepted.');
   write(' Press <CR>.');
   readln;
  end;

 GotoXY(58,1);
 Description_window;
 GotoXY(1,1);
 If not_accepted then
  begin
   asn_matrix[net_number].number_of_events
        :=asn_matrix[net_number].number_of_events-1;
   Display_event_not_accepted_matrix;
   asn_matrix[net_number].number_of_events
        :=asn_matrix[net_number].number_of_events+1;
  end
   else Display_matrix;

 asn_matrix[net_number].number_of_nodes:=
  (asn_matrix[net_number].number_of_nodes)+1;
 If number_ok then
     asn_matrix[net_number].number_of_events
          :=asn_matrix[net_number].number_of_events+1;

End;

{_____________________________________________________________}
```

```
    { ASN Operations }

PROCEDURE Start;

var answer: string [1];
    file_name: string [14];

Begin
 If color_terminal=true then
  begin
   TextColor(15);
   TextBackground(1);
  end;
 ClrScr;
 writeln;
 writeln;
 writeln('   ------------------------------------------------------------');
 writeln('   |                                                          |');
 writeln('   |        ACTIVITY SEQUENCE NET MATRIX EDITOR.              |');
 writeln('   |        -----------------------------------               |');
 writeln('   |                                                          |');
 writeln('   |    This program will enter the different parameters      |');
 writeln('   |   of the ASN matrix of the system.                       |');
 writeln('   |                                                          |');
 writeln('   |    Follow the different menus appearing on the screen    |');
 writeln('   |   in order to correctly enter, position and save all     |');
 writeln('   |   pertinent information concerning your system.          |');
 writeln('   |                                                          |');
 writeln('   ------------------------------------------------------------');
 writeln;
 GotoXY(35,18);
 write('Press <CR>.');
 readln;
 ClrScr;
 GotoXY(5,3);
 answer:='y';
 write('    Do you want to work on a new system (Y/N) ?');
 read (kbd,answer);
 answer:=upcase(answer);
 writeln;
 writeln;
 If ((answer = 'Y') or (answer = 'y')) then new_system:= true
                 else new_system:= false;
 write('     Enter system definition name (8 characters): ');
 readln(system_name);
 ClrScr;
 If not new_system then
  begin
   writeln;
   writeln(' Insert data disk in drive B.');
   write(' Press <CR>.');
   readln;
   Read_system_definition;
   Read_system_labels;
   Read_system_characters;
  end;

End;

{_____________________________________________________________}

{ The purpose of this procedure is to allow the editing of    }
{ already created ASNs. In this editing one can manipulate the }
{ ASN in the same way as during the creation. Creation and    }
{ deletion of events and nodes.                               }

{ The following variables are used here:                      }
{  - ASN_found will indicate that the ASN has been located in }
{    the list of ASN labels.                                  }
```

```
{  - The rest are used as temporary variables for manipulation.  }

PROCEDURE Edit_ASN;

var n, answer :string [1];
    who,name : word;
    i ,number,ext_number : integer;
    ASN_found : boolean;

Begin
  Red_Window;
  Working_window;
  ClrScr;
  Green_window;
  ClrScr;
  Rough_Window;
  ClrScr;
  ASN_found:=false;
  writeln;
  write (' Enter name of ASN network to edit : ');
  who:=read_name;
  ClrScr;
{ We read the system labels in order to compare the label of the }
{ the ASN we want toedit with those existing in the system.      }
  Read_System_labels;
  Rough_window;
  ClrScr;
  For i:=1 to number_of_nets do
   If (ASN_label[i] = who) then
     begin
      net_number:=i;
      number:=i;
      writeln;
      writeln('  The ASN ',ASN_label[net_number],' is ',
         net_number:2);
      ASN_found:=true;
      writeln(' Press <CR> or Q.');
      read(kbd,command);
      ClrScr;
      command:=upcase(command);
     end;
{ At this point, if the entered label coincides with any of the  }
{ ASN labels then we set the condition to true. Otherwise we     }
{ signal that the ASN to edit does not exist. (See at the end.)  }
{ We also allow to go back to main menu even  though the ASN     }
{ label has been recognized.                                     }
  If ( command <> 'Q') then
   If ASN_found then
    begin
     ASN_label[net_number]:=who;
{ Once located, we read the system characteristics.              }
      Rough_window;
      ClrScr;
      Read_system_definition;
      Rough_window;
      ClrScr;
      Read_system_characters;
      Rough_window;
      ClrScr;
      net_number:=number;
      For i:=1 to asn_matrix[net_number].number_of_events do
       If (asn_event_definition[net_number,i].event = ext_event)
          then
        begin
         ext_number:=asn_event_definition[net_number,i].
               event_number;
         name:=ext_event_label[ext_number];
         external_event[net_number,i]:=name;
        end;
      Working_window;
```

```
     ClrScr;
     Red_window;
     Green_window;
     Rough_window;
     ClrScr;
     writeln;
     writeln(' Do you want to check or change any parameter of ');
     write('          the matrix ? (Y/N)');
     while not keypressed do;
     read(kbd,answer);
     ClrScr;
{ Given an affirmative answer, we will start  the editing mode   }
{ The first thing is to show the contents of the matrix to edit. }
{ We will then go to the mode with the adding and deleting menu. }
     If ((answer ='y') or (answer ='Y')) then
      begin
{ Although the matrix is restored, the value of entry is given   }
{ a non x value.                                                  }
       entry:='0';
{ These two variables are incremented in case we want to add any }
{ activities or external events to the ASN.                       }
       number_of_ext_events:=number_of_ext_events+1;
       number_of_activities:=number_of_activities+1;
       Description_window;
       asn_matrix[net_number].number_of_nodes:=
           (asn_matrix[net_number].number_of_nodes)+1;
       asn_matrix[net_number].number_of_events:=
           (asn_matrix[net_number].number_of_events)+1;
       Rough_window;
       ClrScr;
       Working_window;
       Menu_window;
       Restore_matrix;
{ At this point we start to edit. Call SHOW_NODES_AND_EVENTS.     }
       Show_nodes_and_events;
{ We now fill the defined matrix, if a node was added or deleted.}
{ Othervise the matrix remains unchanged.                         }
       Write_in_matrix;
       asn_matrix[net_number].number_of_nodes:=
           (asn_matrix[net_number].number_of_nodes)-1;
       asn_matrix[net_number].number_of_events:=
           (asn_matrix[net_number].number_of_events)-1;
       Red_window;
       working_window;
       ClrScr;
       Green_window;
       Rough_window;
       ClrScr;
       Enter_starting_event;
       Working_window;
       ClrScr;
       Red_window;
       number_of_ext_events:=number_of_ext_events-1;
       number_of_activities:=number_of_activities-1;
       Rough_Window;
{ Now we must save the modifications made to the ASN.             }
       ClrScr;
       writeln;
       writeln('   You have just finished entering the parameters');
       writeln(' on the ASN matrix ',ASN_label[net_number],'.');
       write('   Do you want to save these parameters (Y/N) ?');
       while not keypressed do;
       read (kbd,n);
       n:=upcase(n);
       ClrScr;
       Red_window;
       Rough_window;
       ClrScr;
{ We may or not want to save all the modifications we made.       }
       If (n = 'Y') then
```

```
       begin
        writeln;
        Save_system_Definition;
        Save_system_characters;
        Save_System_labels;
       end;
      ClrScr;
      writeln;
      writeln(' Your matrix for ASN ',net_number:2,' is now entered.');
      writeln(' ------------------------------------');
      writeln;
      writeln('  Press <CR>');
      readln;
     end;
    end
    else
{ As we mentionned before, if no match of labels for the ASN to  }
{ edit has been found, then we display the following message      }
{ before going back to the main menu.                            }
      begin
       ClrScr;
       writeln;
       writeln('   ASN ',who,' has not been created.');
       writeln('   It does not belong to system, or the name was ');
       writeln('   not correctly entered.');
       writeln;
       write('                              Press <CR>.');
       readln;
       ClrScr;
      end;

End;

{_____________________________________________________________}

{ This Procedure allows for the creation of an ASN, including    }
{ the definition of the different nodes and events, and the      }
{ contents of the matrix itself.                                 }

PROCEDURE Create_ASN;

var h, i : integer;
    who : word;
    n, return : string[1];
    new_ASN, first_ASN: boolean;

{ Boolean variables new_ASN and first_ASN are used to determine  }
{ if the ASN we are including belongs to a new system, or if not }
{ if it is the first ASN. If there is an ASN already, and we     }
{ declare that this is to be the first ASN, then we will be      }
{ writing over the ASN # 1.                                      }

Begin

 Red_Window;
 Green_Window;
 Working_window;
 ClrScr;
 Rough_Window;
 ClrScr;
 new_ASN:=true;
 first_ASN:=true;
 command:=' ';
 writeln;
{ We ask if it is the first ASN of system.                        }
 write(' Is this the first ASN of this system ? (Y/N)');
 while not keypressed do ;
 read(kbd,return);
 return:=upcase(return);
 If ( return = 'N' ) then
```

```
  begin
   first_ASN:=false;
   new_ASN:=true;
  end;
{ We now start the creating of an ASN.                              }
{ We first have to check if there is other nets belonging to        }
{ the same system. If not net_number = 1, else, we have to          }
{ get the existing number of nets in that particular system.        }
{ This is done by first reading the labels and then the system      }
{ definition.                                                       }
 If ( new_system and first_ASN )then
  begin
   number_of_activities:=1;
   number_of_ext_events:=1;
   number_of_nets:=1;
   net_number:=1;
   ext_event_number:=1;
   asn_matrix[net_number].number_of_nodes:=1;
   asn_matrix[net_number].number_of_events:=1;
  end
  else
   begin
{ At this point, we are working in an already created system, but   }
{ we are going to create a new ASN for this system.                 }
    Read_system_Labels;
    Read_system_definition;
    Read_system_characters;
    Rough_window;
    ClrScr;
    writeln;
    writeln(' System is not new.');
    writeln(' There are already:');
    writeln('      - ',number_of_nets:2,' networks.');
    writeln('      - ',number_of_activities:2,' activities.');
    write('      - ',number_of_ext_events:2,' external events.');
    readln;
    ClrScr;
    If new_ASN then
     begin
      net_number:=number_of_nets+1;
      number_of_nets:=number_of_nets+1;
      asn_matrix[net_number].number_of_nodes:=1;
      asn_matrix[net_number].number_of_events:=1;
      number_of_activities:=number_of_activities+1;
      number_of_ext_events:=number_of_ext_events+1;
      ext_event_number:=number_of_ext_events;
     end;
   end;
  ClrScr;
  writeln;
  writeln(' This is the ',net_number,' ASN matrix of this system.');
  writeln;
  write(' Enter label of ASN to create [4 characters] : ');
  who:=read_name;
{  We have to check and make sure there are no repetitions in the}
{ labels of the ASN's.                                              }
  For h:=1 to number_of_nets do
    while (who = ASN_label[h]) do
     begin
      ClrSCr;
      writeln;
      writeln(' Invalid label. ASN ',who,' already exists.');
      write(' Reenter ASN label [4 characters]: ');
      who:=read_name;
     end;
  ASN_label[net_number]:=who;
  ClrScr;
{ We remind the user that he must include the declaration of the    }
{ starting event for this ASN in hi event and node definition.      }
  writeln;
```

```
  writeln(' Do not forget to include the starting event of ASN');
  writeln(' ',ASN_label[net_number],' in your declaration.');
  writeln;
  write(' Press <CR> or Q.');
  read(kbd,command);
  command:=upcase(command);
  ClrScr;
  If ( command <> 'Q' ) then
   begin
    writeln;
    writeln('  You will be able, after the next <CR>, to start');
    writeln(' entering the data of the matrix.');
    readln;
{ All the creating will be done now; we define the nodes and     }
{ external events, thus  building up  the ASN matrix. It will be }
{ filled up with zeros (0). In the second part, you will be able }
{ to enter the different parameters for the triggering.          }
    starting_event_entered:=false;
   while (starting_event_entered = false ) do
    begin
     Menu_window;
     Rough_window;
     ClrScr;
     Show_nodes_and_events;
     Write_in_Matrix;
     Green_window;
     Working_window;
     Clrscr;
     Red_window;
     Rough_window;
     ClrScr;
     Enter_Starting_Event;
{ starting_event_entered is set in this procedure.               }
     ClrScr;
     asn_matrix[net_number].number_of_nodes:=
          (asn_matrix[net_number].number_of_nodes)-1;
     asn_matrix[net_number].number_of_events:=
          (asn_matrix[net_number].number_of_events)-1;
    end;
   number_of_ext_events:= number_of_ext_events-1;
   number_of_activities:=number_of_activities-1;
 { This is done because we start already with 1 external event   }
 { and with 1 activity.                                          }
   If color_terminal = true then
    begin
     TextColor(15);
     TextBackground(1);
    end;
{ We now will save the description of the system that has been  }
{ entered; the matrix is saved as containing all 'zeros'.       }
   Working_window;
   ClrScr;
   Rough_Window;
   ClrScr;
   writeln;
   writeln('   You have just finished entering the parameters');
   writeln(' of the ASN matrix ',ASN_label[net_number],'.');
   write('   Do you want to save these parameters (Y/N) ?');
   read(kbd,n);
   n:=upcase(n);
   ClrScr;
   If (n = 'Y') then
    begin
     ClrScr;
     Save_system_Definition;
     ClrScr;
     Save_system_characters;
     ClrScr;
     Save_system_labels;
     Rough_window;
```

```
     ClrScr;
     writeln;
     write('  Press <CR>.');
     readln;
     new_system:=false;
    end
   else
    begin
     ClrScr;
     writeln;
     writeln('  System definition and labels not saved.');
     writeln('                                    Press <CR>. ');
     readln;
    end;
   ClrScr;
   writeln;
   writeln(' Your matrix for ASN ',net_number:2,' is now entered.');
   writeln(' ---------------------------------------');
   writeln;
   writeln('  Press <CR>');
   readln;
   ClrScr;
 end;

End;

{_____________________________________________________________}
```

```
    { Definition of the parameters of the ASN matrix. }

{ The following five procedures are to move the cursor inside    }
{ the matrix of the ASN.                                         }

PROCEDURE GoUp;

var x,i,y : integer;

begin
 i:=i_position;
 x:=x_position;
 y:=y_position;
 i:=i-1;
 y:=y-1;
 GotoXY (x,y);
 i_position:=i;
 y_position:=y;
end;

{_____________________________________________________________________}

PROCEDURE GoDown;

var x,i,y : integer;

Begin
 i:=i_position;
 x:=x_position;
 y:=y_position;
 i:=i+1;
 y:=y+1;
 GotoXY (x,y);
 i_position:=i;
 y_position:=y;

End;

{_____________________________________________________________________}

PROCEDURE Goright;

var x,j,y : integer;

Begin
 j:=j_position;
 x:=x_position;
 y:=y_position;
 j:=j+1;
 x:=x+3;
 GotoXY (x,y);
 j_position:=j;
 x_position:=x;

End;

{_____________________________________________________________________}

PROCEDURE GoLeft;

var x,j,y :integer;

Begin
 j:=j_position;
 x:=x_position;
 y:=y_position;
 j:=j-1;
 x:=x-3;
 GotoXY (x,y);
```

```
 j_position:=j;
 x_position:=x;

End;

{_____________________________________________________________}

PROCEDURE Movement;

Begin
 case (entry) of
  #72:Goup;
  #80:GoDown;
  #75:GoLeft;
  #77:Goright;
 end;      { end of case }

End;

{_____________________________________________________________}

PROCEDURE Display_Matrix;

{ This procedure's role is to update on the screen, particularly }
{ in the matrix, the results of the edit commands, that is, to   }
{ show the editing of the matrix.                                }

var i,j :integer;

Begin
 Working_Window;
 GotoXY(I,1);
 ClrScr;
 number_of_nodes:=asn_matrix[net_number].number_of_nodes;
 write (' ');
 For node_number:=1 to number_of_nodes do
  begin
     write (node_number:2,' ');
  end;
  writeln(' ');
 number_of_events:=asn_matrix[net_number].number_of_events;
 For event_number:=1 to number_of_events do
  begin
   write (' ',event_number:2);
   i:=event_number;
   for node_number:=1 to number_of_nodes do
    begin
{  We initialize everything in the matrix to "@"                 }
     write (' ','@',' ');
     j:=node_number;
     asn_matrix[net_number].matrix[i,j]:='@';
    end;
    writeln(' ');
   end;
End;

{_____________________________________________________________}

PROCEDURE Restore_Matrix;

{ This procedure's role is to show  on  the screen, particularly }
{ in the matrix, the results of the edit commands, that is, to   }
{ show the editing of the matrix. The matrix shown contains up   }
{ to four types of entries :@,I,D and E.                         }

var i,j :integer;

Begin
 Working_Window;
 GotoXY(I,1);
```

```
ClrScr;

number_of_nodes:=asn_matrix[net_number].number_of_nodes;
write ('   ');
For node_number:=1 to number_of_nodes-1 do
 begin
    write (node_number:2,' ');
 end;
 writeln(' ');
number_of_events:=asn_matrix[net_number].number_of_events;
For event_number:=1 to number_of_events-1 do
 begin
  write (' ',event_number:2);
  i:=event_number;
  For node_number:=1 to number_of_nodes-1 do
   begin
    j:=node_number;
    write (' ',asn_matrix[net_number].matrix[i,j],' ');
   end;
   writeln(' ');
 end;
End;

{_____________________________________________________________}

PROCEDURE Display_Event_not_accepted_Matrix;

var i,j :integer;

Begin
 Working_Window;
 GotoXY(I,1);
 ClrScr;
 number_of_nodes:=asn_matrix[net_number].number_of_nodes;
 write ('   ');
 For node_number:=1 to number_of_nodes do
  begin
     write (node_number:2,' ');
  end;
  writeln(' ');
 number_of_events:=asn_matrix[net_number].number_of_events;
 For event_number:=1 to number_of_events do
  begin
   write (' ',event_number:2);
   i:=event_number;
   for node_number:=1 to number_of_nodes do
    begin
     j:=node_number;
     write (' ',asn_matrix[net_number].matrix[i,j],' ');
    end;
    writeln(' ');
   end;
End;

{_____________________________________________________________}

{ This procedure job is to describe in the right hand side of    }
{ the screen, each node and each event appearing in the ASN      }
{ matrix. This table has thus two fields, which will be updated  }
{ whenever there is a change in the ASN.                         }

PROCEDURE Describe_nodes_and_events_Matrix(position:integer);

var nn:char;
    j, number,h,k :integer;
    who : string[4];

Begin
 j:=present_node;
 GotoXY(60,position);
```

```
 nn:=type_of_node[net_number,j];
 nn:=upcase(nn);
  case nn of
   'W':write(j:2,'.WAIT    !');
   'A':begin
        number:=activity_id[net_number,j];
        who:=activity_label[number];
        write(j:2,'.ACT ',who,'!');
       end;
   'B':write(j:2,'.BRAN  ',asn_node_definition[net_number,j].branch_variable_id,' !');
   'C':write(j:2,'.COUNT ',asn_node_definition[net_number,j].counter_variable_id,' !');
   'G':write(j:2,'.GATE    !');
   'T':write(j:2,'.END     !          ');
   'M':write(j:2,'.MERGE   !');
  end; { end of case }
{ Now we must describe the events.                              }
 GotoXY(72,position);

{-----------------------------------------------------------------}
{ At this point we are going to find the event number           }
{ corresponding to this node.                                   }
{ We will then give the description of the particular event     }
{ we are dealing with; it can basically be either external or   }
{ internal.                                                     }

h:=present_event;
If ( asn_event_definition[net_number,h].event =
                int_event ) then
  begin
   nn:=type_of_node[net_number,j];
   nn:=upcase(nn);
   h:=present_event;

   case nn of

    'W':write(h:2,'.WAIT   ');
    'A':begin
         number:=activity_id[net_number,j];
         who:=activity_label[number];
         write(h:2,'.AC ',who);
        end;
    'B':begin
         write(h-1:2,'.BRA ',branch_cond_var[net_number,
               h-1],' ',asn_node_definition[net_number,j].branch_variable_id);
         write('          !');
         write((h):2,'.BRA ',branch_cond_var[net_number,
               h],' ',asn_node_definition[net_number,j].branch_variable_id);
        end;
    'C':write(h:2,'.COUNT ',asn_node_definition[net_number,j].counter_variable_id);
    'G':write(h:2,'.GATE   ');
    'M':write(h:2,'.MERGE  ');

   end;              { end of case }
  end;
End;

{_________________________________________________________________}

PROCEDURE Description_Window;

{ This window contains the description of the nodes and events   }
{ of the matrix. Nodes in the left, events in the right.         }
{ It uses the above procedure DESCRIBE_NODES_AND_EVENTS_MATRIX.  }

var i,j, h, number,position : integer;

Begin

 If ( color_terminal = true ) then
  begin
```

```
  TextColor(0);
  TextBackground(2);
 end;
GotoXY(59,1);
Window (59,1,80,19);
ClrScr;
writeln ('--------------------');
writeln ('   NODES   :  EVENTS ');
writeln ('--------------------');
number_of_nodes:=asn_matrix[net_number].number_of_nodes;
number_of_events:=asn_matrix[net_number].number_of_events;
position:=4;
If (number_of_nodes >= 1) then
 begin
  For j:=1 to number_of_nodes do
   If (number_of_events >= 1) then
    begin
     present_node:=j;
     For h:=1 to number_of_events do
      If (asn_event_definition[net_number,h].event = int_event ) then
      begin
       number:=asn_event_definition[net_number,h].node_number;
       If ( number= present_node ) then present_event:=h;
      end;
     Describe_nodes_and_events_Matrix(position);
     position:=position+1;
   end;
  end;
For i:=1 to number_of_events do
 If (asn_event_definition[net_number,i].event = ext_event ) then
  begin
   event_number:=i;
   GotoXY(60,position);
   write('          !');
   write(i:2,'.EX ',external_event[net_number,event_number]);
   position:=position+1;
  end;

End;

{_____________________________________________________________________}

PROCEDURE Write_in_Matrix;

var x,y,i,j,xx,yy : integer;

Begin

 Working_Window;
 ClrScr;
 Rough_window;
 ClrScr;
 writeln;
 writeln(' CHARACTERS ALLOWED ARE : E, D ,0 and I.');
 writeln;
 writeln(' Nota Bene: Press "i" or "I" to enter I.');
 Cursor_menu_window;
 Description_window;
 Restore_matrix;
 number_of_events:=asn_matrix[net_number].number_of_events;
 number_of_nodes:=asn_matrix[net_number].number_of_nodes;
 node_number:=1;
 event_number:=1;
 x:=5;
 y:=2;
 GotoXY (x,y);
 entry:='0';
 j:=1;
 i:=1;
 while ((entry <>'x') and (entry<>'X')) do
```

```
 if KeyPressed then
 begin
  read(kbd,entry);
  xx:=x;
  yy:=y;
  node_number:=j;
  event_number:=i;
   If (entry=#27) then
    begin
     read (Kbd,entry);
    i_position:=i;
    j_position:=j;
    x_position:=x;
    y_position:=y;
     if ( entry=#72 ) then movement
      else
      if ( entry=#80 ) then movement
       else
       if ( entry=#77 ) then movement
        else
        if ( entry=#75 ) then movement;
      i:=i_position;
      j:=j_position;
      x:=x_position;
      y:=y_position;
   end
    else
    begin
     entry:=upcase(entry);
     If ((entry='I') or (entry='B') or (entry='D') or (entry='E')) then
      begin
       write (entry);
       GotoXY (xx,yy);
       asn_matrix[net_number].matrix [i,j]:=entry;
      end;
    end;
 end;

End;

{_____________________________________________________________}
```

```
    { Definition of the Windows for the Editor. }

PROCEDURE Working_Window;

{ This window will contain the matrix of the ASN; in this matrix }
{ only cursor movement, and entering specific values is allowed. }

Begin
 If ( color_terminal = true ) then
 begin
  TextColor(15);
  TextBackground(1);
 end;
 Window (1,1,58,19);
 GotoXY(5,5);
End;

{_____________________________________________________________}

PROCEDURE Menu_Window;

{ This menu will contain the command menu of the editing mode.   }

Begin
 If ( color_terminal = true ) then
 begin
  TextColor(15);
  TextBackground(4);
 end;
 Window (59,20,80,25);
 ClrScr;
 writeln;
 writeln ('      - MENU -      ');
 writeln (' R.- Node! C.-ExEvent');
 writeln (' S.+ Node! V.+ExEvent');
 writeln ('      X. Exit ');

End;

{_____________________________________________________________}

PROCEDURE Red_Window;

Begin
 If ( color_terminal = true ) then
 begin
  TextColor(15);
  TextBackground(4);
 end;
 Window (59,20,80,25);
 ClrScr;
End;

{_____________________________________________________________}

PROCEDURE Cursor_Menu_Window;

{ This menu will contain the command menu of the editing mode,   }
{ that will allow the filling of the ASN matrix with the         }
{ corresponding parameters.                                      }

Begin
 If ( color_terminal = true ) then
 begin
  TextColor(15);
  TextBackground(4);
 end;
 Window (59,20,80,25);
 ClrScr;
```

```
 writeln;
 writeln ('      - MENU -      ');
 writeln (' Move in matrix with ');
 writeln (' cursor arrow keys.  ');
 writeln ('   Press X to Exit   ');

End;

{_____________________________________________________________}

PROCEDURE Rough_Window;

{ This window will show the questions asked, some temporary     }
{menus, and part of the information entered, that will be put   }
{on the matrix screen above.                                    }

Begin
 If ( color_terminal = true ) then
  begin
   TextColor(0);
   TextBackground(5);
  end;
 Window (1,20,58,25);
 GotoXY(3,20);

End;

{_____________________________________________________________}

PROCEDURE Reduced_rough_window;

{ This is the working window in the case of entering an activity }
{ initiator node.                                               }

Begin
 If ( color_terminal = true) then
  begin
   TextColor(0);
   TexTBackground(5);
  end;
 Window(1,22,58,25);
 GotoXY(2,22);

End;

{_____________________________________________________________}

PROCEDURE Green_window;

Begin

 If ( color_terminal = true ) then
  begin
   TextColor(0);
   TextBackground(2);
  end;

 GotoXY(59,1);
 Window (59,1,80,19);
 ClrScr;

End;

{_____________________________________________________________}

{ This procedure will show the different types of nodes  that   }
{ are available. This menu will appear, together with the       }
{ procedure REDUCED_ROUGH_WINDOW, when we want to add a node    }
{ to the edited ASN.                                            }
```

```
PROCEDURE Display_node_types_menu;

Begin
 If ( color_terminal = true ) then
  begin
   TextColor(0);
   TextBackground(6);
  end;
 Window(1,20,58,22);
 GotoXY(1,20);
 write('  activity initiator:A;    branch:B;   timer/counter:C;   ');
 write('     merge:M;   terminator:E;   wait:W;    gate:G;       ');

End;

{_____________________________________________________________}
```

```
    { Procedures for copying and deleting ASN's. }

{ The purpose of this procedure is to take an existing ASN and   }
{ make a new ASN, identical to it, but with a new ASN label.     }
{ The source ASN must have been created beforehand in order to   }
{ be able to copy from it. The ASN net number assigned to this   }
{ new ASN is going to be (number_of_nets+1); after the number    }
{ will be incremented.                                           }

PROCEDURE COPY_ASN;

var number,i ,j ,source_net_number:  integer;
    who , destination, name : word;
    ASN_found : boolean;
    answer, go_ahead : char;

Begin
 If (color_terminal = true ) then
  begin
   TextColor(15);
   TextBackground(1);
  end;
 Window(1,1,80,25);
 ClrScr;
 GotoXY(8,3);
{ If the system is new,then there is no ASN to copy from.        }
 If not new_system then
  begin
   writeln;
   write(' Enter name of ASN to copy from : [4 characters] ');
   who:=read_name;
{ We have to make sure that source of copying does exist.        }
   Read_system_labels;
   For j:=1 to number_of_nets do
    begin
     If (ASN_label[j] = who) then
      begin
       number:=j;
       ASN_found:=true;
      end;
    end;
{ We will be copying only if the source has been found.          }
   If ASN_found then
    begin
     source_net_number:=number;
     ASN_label[source_net_number]:=who;
     writeln;
     writeln(' Source ASN, ',who,', is ASN number ',number:2);
(At this point we are going to make all the copying of parameters)
     writeln;
     write(' Enter name of new ASN : [4 characters] ');
{ We read the name of the destination ASN.                       }
     destination:=read_name;
     ClrScr;
{ We have to make sure that there are no repetitions in the      }
{ names of the already existing ASNs.                            }

     For j:=1 to number_of_nets do
      while (ASN_label[j] = destination) do
       begin
        writeln;
        writeln(' Invalid new label for new ASN to copy to.');
        writeln;
        write(' Reenter name of new ASN. [4 characters] ');
        destination:=read_name;
       end;
     number:=number_of_nets+1;
     writeln;
     writeln(' The new ASN ',destination,' is net number ',number:2);
```

```
    writeln;
    write(' Press<CR>.');
    readln;
    Read_system_definition;
    Read_system_characters;
    ClrScr;
    GotoXY(5,3);
    net_number:=number_of_nets+1;
    ASN_label[net_number]:=destination;
    writeln;
    write(' ASN source is ',ASN_label[source_net_number]);
    writeln('. It is net number ',source_net_number:2);
    writeln;
    write(' ASN destination is ',ASN_label[net_number]);
    writeln('. It will be net number ',net_number:2);
    writeln;
    write(' Is it all right ? (Y/N) ');
    read(kbd,go_ahead);
    go_ahead:=upcase(go_ahead);

    If go_ahead = 'Y'  then
     begin
      asn_matrix[net_number].number_of_nodes:=
         asn_matrix[source_net_number].number_of_nodes;
      asn_matrix[net_number].number_of_events:=
         asn_matrix[source_net_number].number_of_events;
      number_of_nodes:=asn_matrix[net_number].number_of_nodes;
      number_of_events:=asn_matrix[net_number].number_of_events;
      asn_starting_event[net_number]:=
          asn_starting_event[source_net_number];
      starting_event_label[net_number]:=
          starting_event_label[source_net_number];
{ We start with the matrix itself.                                    }
      For j:=1 to number_of_nodes do
       begin
        For i:=1 to number_of_events do
         begin
          asn_matrix[net_number].matrix[i,j]:=
            asn_matrix[source_net_number].matrix[i,j];
         end;
       end;
{ We then continue with the definition of the different nodes of      }
{ the ASN.                                                            }
      For j:=1 to number_of_nodes do
       begin
        type_of_node[net_number,j]:=
           type_of_node[source_net_number,j];
        asn_node_definition[net_number,j].node:=
           asn_node_definition[source_net_number,j].node;
        case type_of_node[net_number,j] of
{ Depending of the type of node that we have we have to get           }
{ different types of information.                                     }
         'A':begin
              asn_node_definition[net_number,j].activity_id:=
                  asn_node_definition[source_net_number,j].activity_id;
              activity_id[net_number,j]:=activity_id[source_net_number,j];
             end;
         'B':begin
              asn_node_definition[net_number,j].branch_variable_id:=asn_node_definition[source_net_number,j].
                    branch_variable_id;
             end;
         'C':begin
              counter_var[net_number,j]:=
                  counter_var[source_net_number,j];
              asn_node_definition[net_number,j].counter_variable_id:=
                 asn_node_definition[source_net_number,j].counter_variable_id;
             end;
         'G':;
         'M':;
         'W':;
```

```
          'T':;

         end; ( end of case )
        end;
{ At this point we have copied the node characteristics of the   }
{ ASN. We are going to get the event information now.            }
       For i:=1 to number_of_events do
        begin
         type_of_event[net_number,i]:=
              type_of_event[source_net_number,i];
         asn_event_definition[net_number,i].event:=
              asn_event_definition[source_net_number,i].event;
         If asn_event_definition[net_number,i].event = int_event
             then
          begin
           asn_event_definition[net_number,i].node:=
              asn_event_definition[source_net_number,i].node;
           asn_event_definition[net_number,i].node_number:=
              asn_event_definition[source_net_number,i].node_number;
           case type_of_event[net_number,i] of
            'A':;
            'B':begin
                 asn_event_definition[net_number,i].branch_condition:=
                     asn_event_definition[source_net_number,i].
                         branch_condition;
                 branch_cond_var[net_number,i]:=
                      branch_cond_var[source_net_number,i];
                end;
            'C':;
            'G':;
            'M':;
            'W':;
           end;   ( end of case )
          end
         else
          begin
           If (asn_event_definition[net_number,i].event = ext_event)
                       then
            begin
             asn_event_definition[net_number,i].event_number:=
                  asn_event_definition[source_net_number,i].event_number;
             external_event[net_number,i]:=
                  external_event[source_net_number,i];
            end;
           end;
        end;
       ClrScr;
       number_of_nets:=number_of_nets+1;
       writeln;
       writeln(' ASN ',ASN_label[net_number],' has been created.');
       writeln;
       writeln(' This ASN is net number ',net_number:2);
       writeln;
       writeln(' The starting event for ',who,' is: ',starting_event_label
                 [source_net_number]);
       writeln;
       writeln('                                   Press <CR>.');
       readln;
       ClrSCr;
       writeln;
       write(' Do you wish to save this ASN ? (Y/N)');
       read(kbd,answer);
       answer:=upcase(answer);
       If (answer = 'Y') then
        begin
         Save_system_labels;
         Save_system_definition;
         Save_system_characters;
        end;
       end;
```

```
      end
     else
      begin
       ClrScr;
       GotoXY(5,4);
       writeln;
       writeln(' The source ASN, ',who,' has not been found.');
       writeln;
       writeln(' Either :');
       writeln('   - it does not exist in this system.');
       writeln('   - there is an spelling error in the ASN name.');
       writeln;
       writeln;
       write('    Press <CR).');
       readln;
      end;
    end;

End;

{_____________________________________________________________}

PROCEDURE MOVE_ASNs_DOWN(net_number_to_delete: integer);

{ This procedure is called by the procedure DELETE_ASN, and will }
{ shift the ASNs down by one, overwriting, starting from the net }
{ to be deleted, by the net immediately after it; that is ASN 3  }
{ will become ASN 2, if the original ASN 2 was to be deleted.    }

var source_net_number, m, c, i, j: integer;

Begin
 m:=net_number_to_delete;
 For c:=1 to number_of_nets-net_number_to_delete do
  begin
   Read_system_definition;
   Read_system_labels;
   Read_system_characters;
   net_number:=m;
   source_net_number:=m+1;
   ASN_label[net_number]:=ASN_label[source_net_number];
   asn_matrix[net_number].number_of_nodes:=
            asn_matrix[source_net_number].number_of_nodes;
   asn_matrix[net_number].number_of_events:=
            asn_matrix[source_net_number].number_of_events;
   number_of_nodes:=asn_matrix[net_number].number_of_nodes;
   number_of_events:=asn_matrix[net_number].number_of_events;
   asn_starting_event[net_number]:=
            asn_starting_event[source_net_number];
   starting_event_label[net_number]:=
            starting_event_label[source_net_number];
  { We now must start adjusting the characteristics of the Matrix. }
   For j:=1 to number_of_nodes do
    begin
     For i:=1 to number_of_events do
      begin
       asn_matrix[net_number].matrix[i,j]:=
           asn_matrix[source_net_number].matrix[i,j];
      end;
    end;
  { We must now continue with the description of the nodes.        }
   For j:=1 to number_of_nodes do
    begin
     type_of_node[net_number,j]:=
            type_of_node[source_net_number,j];
     asn_node_definition[net_number,j].node:=
         asn_node_definition[source_net_number,j].node;
     case type_of_node[net_number,j] of
      'A':begin
           asn_node_definition[net_number,j].activity_id:=
```

```
            asn_node_definition[source_net_number,j].activity_id;
          activity_id[net_number,j]:=
            activity_id[source_net_number,j];
        end;
      'B':begin
          asn_node_definition[net_number,j].branch_variable_id:=
            asn_node_definition[source_net_number,j].
              branch_variable_id;
        end;
      'C':begin
          counter_var[net_number,j]:=
            counter_var[source_net_number,j];
          asn_node_definition[net_number,j].counter_variable_id:=
            asn_node_definition[source_net_number,j].
              counter_variable_id;
        end;
      'G':;
      'M':;
      'W':;
      'T':;

    end;  { end of case }
   end;
{ We have completed the adjustement of the nodes and their       }
{ description. We now turn to the events.                        }
  For i:=1 to number_of_events do
   begin
    type_of_event[net_number,i]:=
        type_of_event[source_net_number,i];
    asn_event_definition[net_number,i].event:=
        asn_event_definition[source_net_number,i].event;
    If ( asn_event_definition[net_number,i].event = int_event ) then
     begin
      asn_event_definition[net_number,i].node:=
          asn_event_definition[source_net_number,i].node;
      asn_event_definition[net_number,i].node_number:=
          asn_event_definition[source_net_number,i].node_number;
      case type_of_event[net_number,i] of
       'B':begin
            asn_event_definition[net_number,i].branch_condition:=
              asn_event_definition[source_net_number,i].
                branch_condition;
           end;
       'A':;
       'C':;
       'G':;
       'M':;
       'W':;
       'T':;
      end;   { end of case }
     end
    else
     begin
      If ( asn_event_definition[net_number,i].event = ext_event )
           then
       begin
        asn_event_definition[net_number,i].event_number:=
          asn_event_definition[source_net_number,i].
            event_number;
        external_event[net_number,i]:=
          external_event[source_net_number,i];
       end;
     end;
   end;
{ We have to go and get the next ASN to shift down.              }
   m:=m+1;
  end;
  number_of_nets:=number_of_nets-1;

End;
```

```
{_____________________________________________________________}

PROCEDURE Delete_ASN;

{ This procedure is used to delete an ASN. The actual operation  }
{ is as follows: the contents of the ASN to delete are destroyed.}
{ The purpose of this is to allow the inclusion of a new ASN at  }
{ that particular ASN number. The net number of the other ASNs   }
{ is not altered. Further on, the external events, and activities}
{ that were included in that ASN are preserved. That is, they    }
{ still have their assigned number. The numbering is transparent }
{ to the user.                                                   }
{ However we include a provision to allow the user to completely }
{ destroy the ASN and to adjust the net number of the remaining  }
{ ASNs affected by this deletion.                                }

var h,i,j,k,number, net_number_to_delete:integer;
    who : word;
    n, nn: char;
    ASN_found : boolean;

Begin
 Window(1,1,80,25);
 ClrScr;
 writeln;
 writeln(' You have now to choices:');
 writeln;
 writeln('  - 1 / Delete ASN and leave the blank to be filled, or');
 writeln('  - 2 / Delete ASN and adjust the remaining ASNs.');
 writeln('  - 3 / Quit.');
 writeln;
 write('            Enter "1","2" or "3" as a choice.');
 read (kbd,n);
 ClrScr;
 nn:=n;

 case n of
  '1' : begin
        GotoXY(4,2);
        writeln;
        write(' Enter name of ASN to delete [4 characters] :');
        who:=read_name;
 { We have to make sure that the ASN to deleted exists.          }
        Read_system_labels;
        For j:=1 to number_of_nets do
         begin
          If ASN_label[j] = who then
           begin
            number:=j;
            ASN_found:=true;
           end;
         end;
        GotoXY(4,2);
        writeln(' ASN to delete is ',who,', net number '
                  ,number:2,'.');
        write(' Press <CR> or Q.');
        read(kbd,n);
        n:=upcase(n);
        If (n<>'Q') then
         begin
          net_number_to_delete:=number;
          For j:=1 to number_of_nodes do
           begin
            For i:=1 to number_of_events do
             begin
              asn_matrix[net_number_to_delete].matrix[i,j]:='0';
             end;
           end;
          asn_matrix[net_number_to_delete].number_of_nodes:=1;
```

```
         asn_matrix[net_number_to_delete].number_of_events:=1;
         asn_starting_event[net_number_to_delete]:=1;
        end;
       Clrscr;
       GotoXY(3,3);
       writeln(' Do not forget to fill the gap created by the');
       writeln(' ASN that has been deleted. Remember that the');
       writeln(' remaining ASNs have not been adjusted down.');
       writeln;
       writeln('                                    Press <CR>.');
       readln;
      end;

  '2' : begin
        GotoXY(3,2);
        writeln;
        write(' Enter name of ASN to delete [4 characters] :');
        who:=read_name;
{ We have to make sure that the ASN to deleted exists.          }
        Read_system_labels;
        For j:=1 to number_of_nets do
         begin
          If ASN_label[j] = who then
           begin
            number:=j;
            ASN_found:=true;
           end;
         end;
        GotoXY(4,2);
        writeln(' ASN to delete is ',who,', net number '
                 ,number:2,'.');
        write('  Press <CR> or Q.');
        read(kbd,n);
        n:=upcase(n);
        If (n<>'Q') then
         begin
          net_number_to_delete:=number;
          Move_ASNs_down(net_number_to_delete);
         end;
       end;
  else
     begin
      ClrScr;
      writeln;
      writeln(' Incorrect Entry. ASN not deleted. ');
      writeln;
      readln;
     end;
    end; { end of case }

   If (( nn ='1') or (nn ='2')) then
    begin
     ClrScr;
     GotoXY(5,3);
     write('  Do you want to save the last changes (Y/N) ?');
     read(kbd,n);
     n:=upcase(n);
     If (n='Y') then
      begin
       Save_system_definition;
       Save_system_characters;
       Save_system_labels;
      end;
    end;

End;

{_____________________________________________________________}
```

```
    ( Procedures to rename events, activities and ASN's. )

PROCEDURE RENAME_ASN;

{ The purpose of this procedure is to change the label associated }
{ with an ASN. We will list the labels of the existing ASNs, ask  }
{ for which ASN to rename, check if label exists and change the   }
{ label of the corresponding ASN.                                 }

var who, new_who : word;
    count, i : integer;
    ASN_found1, ASN_found2 : boolean;
    answer : string[1];

Begin
 Window(1,1,80,25);
 Read_system_labels;
 ClrScr;
 writeln;
 writeln(' These are the ASN''s in this system :');
 writeln;
 count:=0;
 For i:=1 to number_of_nets do
  begin
   If (count<4) then
    begin
     write(' ',ASN_label[i],' , ');
     count:=count+1;
    end
   else
    begin
     writeln(' ',ASN_label[i]);
     count:=0;
    end;
  end;
 writeln;
 writeln;
 write(' Enter the name of the ASN you want to rename : ');
 who:=read_name;
 ASN_found1:=false;
 For i:=1 to number_of_nets do
  If (ASN_label[i]=who) then
   begin
    net_number:=i;
    ASN_found1:=true;
   end;
 If not ASN_found1 then
  begin
   ClrScr;
   writeln;
   writeln(' ASN ',who,' does not exist in this system.');
   writeln;
   write(' Press <CR>. ');
   readln;
  end
 else
  begin
   ClrScr;
   writeln;
   writeln(' ASN to rename is net number ',net_number:2,' .');
   writeln;
   write(' Enter new name for this ASN: [4 characters] ');
   new_who:=read_name;
   ASN_found2:=true;
   while ASN_found2 do
    begin
     ASN_found2:=false;
     For i:=1 to number_of_nets do
      If (ASN_label[i]=new_who) then
```

```
       begin
        ASN_found2:=true;
       end;
      If ASN_found2 then
       begin
        writeln;
        writeln('  Invalid name. Label ',new_who,' already exists.');
        writeln('  Reenter new name :');
        new_who:=read_name;
       end;
    end;
   Clrscr;
   writeln;
   writeln(' Old name is ',ASN_label[net_number],' .');
   writeln;
   writeln(' New name is ',new_who,' .');
   writeln;
   write('  Is this all right ? (Y/N) ');
   read(kbd,answer);
   answer:=upcase(answer);
   If ( answer ='Y') then
    begin
     ASN_label[net_number]:=new_who;
     Save_system_labels;
    end
   else
    begin
     writeln;
     writeln(' Labels not changed.');
     write('  Press <CR>.');
     readln;
    end;
  end;

End;

{_____________________________________________________________}

PROCEDURE RENAME_Activity;

{ In this procedure we will allow the renaming of activities.   }
{ The operation is similar to that of RENAME_ASN.               }

var who, new_who : word;
    count, i : integer;
    activity_found1, activity_found2 : boolean;
    answer : string[1];

Begin
 Window(1,1,80,25);
 Read_system_labels;
 ClrScr;
 writeln;
 writeln(' These are the activities in this system :');
 writeln;
 count:=0;
 For i:=1 to number_of_activities do
  begin
   If (count<4) then
    begin
     write(' ',activity_label[i],' , ');
     count:=count+1;
    end
   else
    begin
     writeln(' ',activity_label[i]);
     count:=0;
    end;
  end;
 writeln;
```

```
writeln;
write('  Enter the name of the activity you want to rename : ');
who:=read_name;
activity_found1:=false;
For i:=1 to number_of_activities do
 If (activity_label[i]=who) then
  begin
   activity_number:=i;
   activity_found1:=true;
  end;
If not activity_found1 then
 begin
  ClrScr;
  writeln;
  writeln(' Activity ',who,' does not exist in this system.');
  writeln;
  write('  Press <CR>. ');
  readln;
 end
else
 begin
  ClrScr;
  writeln;
  writeln(' Activity to rename is activity number ',
                                  activity_number:2,' .');
  writeln;
  write('  Enter new name for this activity: [4 characters] ');
  new_who:=read_name;
  activity_found2:=true;
  while activity_found2 do
   begin
    activity_found2:=false;
    For i:=1 to number_of_activities do
     If (activity_label[i]=new_who) then
      begin
       activity_found2:=true;
      end;
     If activity_found2 then
      begin
       writeln;
       writeln('  Invalid name. Label ',new_who,' already exists.');
       writeln('  Reenter new name :');
       new_who:=read_name;
      end;
   end;
  Clrscr;
  writeln;
  writeln(' Old name is ',activity_label[activity_number],' .');
  writeln;
  writeln(' New name is ',new_who,' .');
  writeln;
  write('  Is this all right ? (Y/N) ');
  read(kbd,answer);
  answer:=upcase(answer);
  If ( answer ='Y') then
   begin
    activity_label[activity_number]:=new_who;
    Save_system_labels;
   end
  else
   begin
    writeln;
    writeln(' Labels not changed.');
    write('  Press <CR>.');
    readln;
   end;
 end;

End;
```

```
{_____________________________________________________________}

PROCEDURE RENAME_ext_event;

{ In this procedure we will allow the renaming of external      }
{ events.                                                       }
{ The operation is similar to that of RENAME_ASN.               }

var who, new_who : word;
    count, i : integer;
    ext_event_found1, ext_event_found2 : boolean;
    answer : string[1];

Begin
 Window(1,1,80,25);
 Read_system_labels;
 ClrScr;
 writeln;
 writeln(' These are the external events in this system :');
 writeln;
 count:=0;
 For i:=1 to number_of_ext_events do
  begin
   If (count<4) then
    begin
     write(' ',ext_event_label[i],' , ');
     count:=count+1;
    end
   else
    begin
     writeln(' ',ext_event_label[i]);
     count:=0;
    end;
  end;
 writeln;
 writeln;
 write('  Enter the name of the external event you want to rename : ');
 who:=read_name;
 ext_event_found1:=false;
 For i:=1 to number_of_ext_events do
  If (ext_event_label[i]=who) then
   begin
    ext_event_number:=i;
    ext_event_found1:=true;
   end;
 If not ext_event_found1 then
  begin
   ClrScr;
   writeln;
   writeln(' External event ',who,' does not exist in this system.');
   writeln;
   write('  Press <CR>. ');
   readln;
  end
 else
  begin
   ClrScr;
   writeln;
   writeln(' External Event to rename is external event number ',
                                  ext_event_number:2,' .');
   writeln;
   write('  Enter new name for this external event: [4 characters] ');
   new_who:=read_name;
   ext_event_found2:=true;
   while ext_event_found2 do
    begin
     ext_event_found2:=false;
     For i:=1 to number_of_ext_events do
      If (ext_event_label[i]=new_who) then
       begin
```

```
      ext_event_found2:=true;
     end;
    If ext_event_found2 then
     begin
      writeln;
      writeln('  Invalid name. Label ',new_who,' already exists.');
      writeln('  Reenter new name :');
      new_who:=read_name;
     end;
   end;
  Clrscr;
  writeln;
  writeln(' Old name is ',ext_event_label[ext_event_number],' .');
  writeln;
  writeln(' New name is ',new_who,' .');
  writeln;
  write('  Is this all right ? (Y/N) ');
  read(kbd,answer);
  answer:=upcase(answer);
  If ( answer ='Y') then
   begin
    ext_event_label[ext_event_number]:=new_who;
    Save_system_labels;
   end
  else
   begin
    writeln;
    writeln(' Labels not changed.');
    write('  Press <CR>.');
    readln;
   end;
 end;

End;

{_____________________________________________________________}

PROCEDURE RENAME;

var number, choice : integer;

Begin
 ClrScr;
 writeln;
 writeln(' You can rename :');
 writeln;
 writeln('   1- an ASN,');
 writeln('   2- an Activity,');
 writeln('   3- an external event.');
 writeln;
 write(' Enter number of choice : ');
 readln(choice);
 case choice of
  1:Rename_ASN;
  2:Rename_Activity;
  3:Rename_ext_event;
 end;  { end of case }
 Clrscr;

End;

{_____________________________________________________________}
```

```
    ( MAIN program for Editor. )

PROGRAM ASN_Matrix_Editor (input,output);

{$R+}
{$I DECLAR.GLO}
{$I FILECHEC.GLO}
{$I READSYS.GLO}
{$I SAVESYS.GLO}
{$I SYSLIST.GLO}
{$I WINDOWS.EDT}
{$I MATDEST.EDT}
{$I EVENTIN.EDT}
{$I NODEIN.EDT}
{$I ASNOPER.EDT}

{-------------------------------------------------------------}

{ Some of the variables here will be used mainly for displaying  }
{ purposes; such is  the case  of  type_of_node, type_of_event,  }
{ branch_cond_var, activity_label and counter_var.               }
{ They have in some case corresponding variable  types included  }
{ in the declaration included in this program.                   }

{_____________________________________________________________}

{ MAIN PROGRAM }

var choice_number, choice:integer;
    answer : string[1];
    value : integer;

BEGIN

 ClrScr;
 writeln;
 write('    Do you have a color monitor ? ( Y / N )');
 read (kbd,answer);
 If ((answer = 'Y') or ( answer = 'y')) then color_terminal:= true
              else color_terminal:= false;
 choice:=0;
 If ( color_terminal = true ) then
  begin
   TextColor(15);
   TextBackground(1);
  end;
 ClrScr;
 writeln;
 writeln('              PROCESS ACTIVITY DIAGRAM EDITOR.');
 writeln('              --------------------------------');
 writeln;
 writeln('    This EDITOR is part of a toolset package that can be used ');
 writeln('  for the definition and specification of a system. ');
 writeln('    This package consists also of MATEDIT and INSPECT.');
 writeln;
 writeln('    Once the system to study has been defined using PAD''s, then');
 writeln('  the user will be able to enter all this information using both');
 writeln('  programs, EDITOR and MATEDIT.');
 writeln;
 writeln('    All this data will be saved in three files, defined by the ');
 writeln('  system name; these will be system name.asn, .nam and .cha.');
 writeln;
 writeln('    The user does not need to worry about which file to read or ');
 writeln('  to save. Files .nam contain all the labels of activities,');
 writeln('  ASN''s and external events present in the system; these labels');
 writeln('  are 4 characters long every time.');
 writeln;
 writeln('     This program was written by Jose M. DURAN.');
 writeln;
```

```
write('                                        Press (CR).');
readln;
while (choice <> 5) do
 Begin
  Start;
  choice_number:=0;
  While (choice_number <> 4) do
  begin
   If (color_terminal = true) then
    begin
     TextColor(15);
     TextBackground(1);
    end;
   Window(1,1,80,25);
   ClrScr;
   GotoXY(12,4);
   write('MENU OF COMMANDS');
   GotoXY(15,7);
   writeln('1. Create an ASN.');
   GotoXY(15,9);
   writeln('2. Edit an ASN.');
   GotoXY(15,11);
   writeln('3. List system parameters.');
   GotoXY(15,13);
   writeln('4. Change System.');
   GotoXY(15,15);
   writeln('5. Quit.');
   GotoXY(1,20);
   write('              Enter choice number : ');
   value:=read_number;
   choice_number:=value;

   case choice_number of

    1:Create_ASN;
    2:Edit_ASN;
    3:List_System_Parameters;
    5:begin
       choice:=5;
       choice_number:=4;
      end;
   end; { end of case }
  end;
 End;
 ClrScr;

END.

{_____________________________________________________________}
```

```
    ( MAIN Program for MATEDIT. )

PROGRAM MATRIX_MANIPULATOR (input,output);

{$I DECLAR.GLO}
{$I FILECHEC.GLO}
{$I READSYS.GLO}
{$I SAVESYS.GLO}
{$I SYSLIST.GLO}
{$I RENAME.MAT}
{$I COPYDEL.MAT}

{_____________________________________________________________}

{ The purpose of this matrix is to show, in a node by node basis }
{ the contents of the matrix. The user can only see what the     }
{ contents are. If any modification is desired, then he will     }
{ have to go to the editing mode.                                }

{_____________________________________________________________}

{ MAIN PROGRAM }

var choice_number, choice:integer;
    answer : string[1];
    value : integer;

BEGIN

 ClrScr;
 writeln;
 write('    Do you have a color monitor ? ( Y / N )');
 read (kbd,answer);
 If ((answer = 'Y') or ( answer = 'y')) then color_terminal:= true
              else color_terminal:= false;
 If ( color_terminal = true ) then
  begin
   TextColor(15);
   TextBackground(1);
  end;
 ClrScr;
 writeln;
 writeln('             MATRIX MANIPULATOR FOR PAD''S. ');
 writeln('             ----------------------------');
 writeln;
 writeln('      This program is part of the PAD''s editing facility');
 writeln('   tool.');
 writeln;
 writeln('      This program will allow the user the following:');
 writeln;
 writeln('      - to make an identical copy of an already existing ASN.');
 writeln('      - to delete an ASN from a system definition.');
 writeln('      - to rename an ASN, an external event or an activity.');
 writeln;
 writeln;
 writeln('      This program was written by Jose M. DURAN.');
 writeln;
 writeln;
 write('                                      Press <CR>.');
 readln;
 choice:=0;
 while (choice <> 6) do
  Begin
   ClrScr;
   GotoXY(4,3);
   write('  Enter system name [8 characters] : ');
   readln(system_name);
   Read_system_labels;
   ClrScr;
```

```
 choice_number:=0;
 While (choice_number <> 5) do
 begin
  If (color_terminal = true) then
   begin
    TextColor(15);
    TextBackground(1);
   end;
  Window(1,1,80,25);
  ClrScr;
  GotoXY(19,4);
  write('MENU OF COMMANDS');
  GotoXY(15,8);
  writeln('1. List system parameters.');
  GotoXY(15,10);
  writeln('2. Make a copy of an ASN.');
  GotoXY(15,12);
  writeln('3. Delete an ASN.');
  GotoXY(15,14);
  writeln('4. Rename an ASN, an activity or an external event.');
  GotoXY(15,16);
  writeln('5. Change System.');
  GotoXY(15,18);
  writeln('6. Quit.');
  GotoXY(1,22);
  write('              Enter choice number : ');
  value:=read_number;
  choice_number:=value;
  case choice_number of

   1:List_System_Parameters;
   2:Copy_ASN;
   3:Delete_ASN;
   4:Rename;
   6:begin
      choice:=6;
      choice_number:=5;
     end;
  end;     { end of case }
 end;
End;
ClrScr;

END.

{_____________________________________________________________}
```

APPENDIX B.

This appendix contains a program for the verification of the 'mechanical' errors found in an ASN, whose definition has been entered by means of the ASN matrix EDITOR.

The program includes global procedures listed in appendix A.

TABLE OF CONTENTS

```
    ( Menus for INSPECT. )

PROCEDURE CHECKING_MENU;

Begin

 Window(1,1,80,25);
 ClrScr;
 If (color_terminal) then
  begin
   Textcolor(0);
   TextBackground(6);
  end;
 Window(5,1,80,12);
 ClrScr;
 writeln;
 writeln('    Choice of checking to do on ASNs.');
 writeln('    ---------------------------------');
 writeln;
 writeln(' 1 - Check triggering of nodes and events.');
 writeln(' 2 - Check for basic anomalies in the connectivity.');
 writeln(' 3 - Check for lack of essential nodes and events.');
 writeln(' 4 - Complete ASN check up. (includes 1, 2 and 3).');
 writeln(' 5 - Check the Epsilon rule.');
 writeln(' 6 - Change ASN to check.');
 writeln(' 7 - Quit.');

End;

{_____________________________________________________________}

PROCEDURE EVENT_NODE_TRIGGERING_MENU;

Begin

 writeln;
 writeln('¦-----------------------------------------------------¦');
 writeln('¦   The purpose of this process is to verify that     ¦');
 writeln('¦  the different nodes in this ASN are correctly      ¦');
 writeln('¦  triggered.                                         ¦');
 writeln('¦-----------------------------------------------------¦');
 writeln('¦                                                     ¦');
 writeln('¦   We are going to verify the following :           ¦');
 writeln('¦                                                     ¦');
 writeln('¦   - if any node in the net is not being triggered.  ¦');
 writeln('¦     This causes a break in the execution sequence.  ¦');
 writeln('¦                                                     ¦');
 writeln('¦   - if any node in the net is being incorrectly     ¦');
 writeln('¦     triggered; that is like an activity initiator   ¦');
 writeln('¦     node having an "D" or an "E" trigger.           ¦');
 writeln('¦                                                     ¦');
 writeln('¦   - if any special node in the net, that is of type ¦');
 writeln('¦     gate or timer/counter is missing one trigger    ¦');
 writeln('¦     type; these nodes must have all three trigger   ¦');
 writeln('¦     types, "D", "I" and  "E".                       ¦');
 writeln('¦                                                     ¦');
 writeln('¦-----------------------------------------------------¦');
 writeln;

End;

{_____________________________________________________________}

PROCEDURE CONNECTIVITY_ANOMALIES_MENU;

Begin

 writeln('¦-----------------------------------------------------¦');
 writeln('¦   The purpose of this process is to verify that     ¦');
```

```
 writeln('¦  there are no basic anomalies in the connectivity          ¦');
 writeln('¦  of the ASN. These basic anomalies are:                     ¦');
 writeln('¦                                                           ¦');
 writeln('¦   We are going to look for two types of anomalies:         ¦');
 writeln('¦                                                           ¦');
 writeln('¦     - a node self triggering; this means, that except     ¦');
 writeln('¦       in the case of the "merge" node, a DEADLOCK          ¦');
 writeln('¦       situation will occur; this is because all the       ¦');
 writeln('¦       nodes are of the wait type, and they require all    ¦');
 writeln('¦       inputs to be present to be acitvated. However       ¦');
 writeln('¦       the output trigger is only issued once the node     ¦');
 writeln('¦       has been executed.                                  ¦');
 writeln('¦       In the case of a "merge" type node, we will have    ¦');
 writeln('¦       as soon as the ASN is awakened a series of triggers ¦');
 writeln('¦       coming from the node.                               ¦');
 writeln('¦     - an internal event not triggering anything at all,   ¦');
 writeln('¦       this having the effect of a cut in the execution    ¦');
 writeln('¦       sequence.                                           ¦');
 writeln('¦                                                           ¦');
 writeln('¦-----------------------------------------------------------¦');

End;

{_____________________________________________________________}

PROCEDURE LACK_OF_NODES_MENU;

Begin
 writeln('¦-----------------------------------------------------------¦');
 writeln('¦    In this case, we have to check if the current ASN       ¦');
 writeln('¦      contains the necessary nodes in a given ASN.          ¦');
 writeln('¦                                                           ¦');
 writeln('¦-----------------------------------------------------------¦');
 writeln('¦   Basically, this verification applies to only two types   ¦');
 writeln('¦ of nodes, activity initiator and terminator (end) nodes.   ¦');
 writeln('¦                                                           ¦');
 writeln('¦   - In the case of no activity node being present in the ASN, ¦');
 writeln('¦ we will issue a warning statement; fundamently it is not   ¦');
 writeln('¦ incorrect but it does not make too much sense to have a    ¦');
 writeln('¦ complete ASN with only the reamining types of "active" nodes. ¦');
 writeln('¦                                                           ¦');
 writeln('¦   - In the case of a missing terminator node, an ERROR message ¦');
 writeln('¦  will be issued. The END node is necessary in the ASN.     ¦');
 writeln('¦  One can visualize this node as an activity notifying the ASN ¦');
 writeln('¦  manager that the ASN has been completed.                 ¦');
 writeln('¦  Otherwise this ASN will remain awakened for ever.        ¦');
 writeln('¦                                                           ¦');
 writeln('¦    - Lack of external events will also be checked here.   ¦');
 writeln('¦                                                           ¦');
 writeln('¦-----------------------------------------------------------¦');
 writeln;

End;

{_____________________________________________________________}
```

```
     { Checking of the node-event triggering. }

PROCEDURE CHECK_EVENT_NODE_TRIGGERING;

VAR node_triggered, node_enabled, node_disabled : boolean;
    i, j, ev_number : integer;
    who : word;
    no_problem, ASN_found, incorrect_trigger : boolean;
    node : node_type;

Begin

 If (color_terminal) then
  begin
   TextColor(15);
   TextBackground(1);
  end;
 window(1,1,80,25);
 Clrscr;
 Window(9,1,80,25);
 ClrScr;
 Event_node_triggering_menu;
 write('                                          Press <CR>.');
 readln;
 window(1,1,80,25);
 number_of_nodes:=asn_matrix[net_number].number_of_nodes;
 number_of_events:=asn_matrix[net_number].number_of_events;
{ We have to see, depending on the type of node if the           }
{ triggering is correct or not.                                  }
 ClrScr;
 For j:=1 to number_of_nodes do
  begin
   no_problem:=true;
   node_triggered:=false;
   node_enabled:=false;
   node_disabled:=false;
   incorrect_trigger:=false;
{ Depending on the type of node that we have, there can be       }
{ different types of triggers associated with it.                }
{ For instance, only gates and timer/counters must be enabled    }
{ and disabled. The rest can only be triggered.                  }
   node:=asn_node_definition[net_number,j].node;
   case node of
    counter, gate:
     begin
      For i:=1 to number_of_events do
       begin
        If (asn_matrix[net_number].matrix[i,j]='E') then
            node_enabled:=true
        else
        If (asn_matrix[net_number].matrix[i,j]='D') then
            node_disabled:=true
        else
        If (asn_matrix[net_number].matrix[i,j]='I') then
            node_triggered:=true;
       end;
      If not node_triggered then
       begin
        no_problem:=false;
        writeln;
        writeln(' Node ',j:2,' is not triggered.');
       end;
      If not node_enabled then
       begin
        no_problem:=false;
        writeln;
        writeln(' Node ',j:2,' is not enabled.');
       end;
      If not node_disabled then
```

```
      begin
       no_problem:=false;
       writeln;
       writeln(' Node ',j:2,' is not  disabled.');
      end;
     end
    else
     begin
      For i:=1 to number_of_events do
       begin
        If (asn_matrix[net_number].matrix[i,j]='I') then
            node_triggered:=true
        else
        If ((asn_matrix[net_number].matrix[i,j]='D')or
               (asn_matrix[net_number].matrix[i,j]='E')) then
          begin
           no_problem:=false;
           writeln;
           writeln(' Node ',j:2,' has a non acceptable trigger ;');
           writeln('                    in event number ',i:2,'.');
          end;
       end;
      If not node_triggered then
       begin
        no_problem:=false;
        writeln;
        writeln(' Node ',j:2,' is not triggered.');
       end;
     end;
   end; { end of case }
  end;
  writeln;
  If no_problem then
   begin
    writeln;
    writeln(' No problem in the event-node triggering.');
   end;
  writeln;
  write('                  Press <CR>.');
  readln;

End;

{_____________________________________________________________}
```

```
    { Checking of the Basic connectivity anomalies. }

PROCEDURE CHECK_BASIC_CONNECTIVITY_ANOMALIES;

VAR no_problem1, no_problem2, event_triggers, self_trigger : boolean;
    i, j : integer;

Begin
 If (color_terminal) then
  begin
   TextColor(15);
   TextBackground(1);
  end;
 window(1,1,80,25);
 Clrscr;
 Window(8,1,80,25);
 ClrScr;
 Connectivity_anomalies_menu;
 write('                                                Press <CR>.');
 readln;
 window(1,1,80,25);
 ClrScr;
 no_problem:=true;
 no_problem1:=true;
 no_problem2:=true;
 event_triggers:=true;
 self_trigger:=false;
 number_of_nodes:=asn_matrix[net_number].number_of_nodes;
 number_of_events:=asn_matrix[net_number].number_of_events;
 For j:=1 to number_of_nodes do
  begin
   For i:=1 to number_of_events do
    begin
     If (asn_event_definition[net_number,i].node_number = j ) then
      begin
       node_number:=j;
       event_number:=i;
       If ( asn_matrix[net_number].matrix[event_number,
            node_number] = '1') then
        If (asn_event_definition[net_number,i].event = int_event )
             then
         begin
          no_problem:=false;
          If (asn_event_definition[net_number,i].node = merge ) then
           begin
            writeln;
            writeln(' Merge node ',node_number:2,' is self triggering.');
            writeln('    Multiple triggers will result.');
           end
          else
           begin
            writeln;
            writeln(' Node ',node_number:2,' is self triggering.');
            writeln('    Deadlock situation will occur.');
           end;
         end;
       end;
    end;
   end;
  If no_problem then
   begin
    writeln;
    writeln(' No connectivity anomalies have been found in this ASN.');
   end;
  writeln;
  write('                                          Press <CR>.');
  readln;
  Clrscr;
  For i:=1 to number_of_events do
```

```
  begin
   event_triggers:=false;
   For j:=1 to number_of_nodes do
    begin
     If ((asn_matrix[net_number].matrix[i,j]='I')or
         (asn_matrix[net_number].matrix[i,j]='O')or
          (asn_matrix[net_number].matrix[i,j]='E')) then
            event_triggers:=true;
    end;
   If not event_triggers then
    begin
     no_problem2:=false;
     writeln;
     writeln(' Event ',i:2,' does not trigger anything.');
    end;
  end;
 If no_problem2 then
  begin
   writeln;
   writeln(' No problem as all events trigger some node.');
  end;
 writeln;
 write('                                    Press <CR>.');
 readln;

End;
```

```
    { Checking the lack of nodes. }

PROCEDURE CHECK_LACK_OF_NODES;

Var no_activity, no_end, no_external_event : boolean;
    i, j : integer;

Begin
 If (color_terminal) then
  begin
   TextColor(15);
   TextBackground(1);
  end;
 window(1,1,80,25);
 Clrscr;
 Window(6,1,80,25);
 ClrScr;
 Lack_of_nodes_menu;
 write('                                              Press <CR>.');
 readln;
 window(1,1,80,25);
 ClrScr;
 no_external_event:=true;
 no_activity:=true;
 no_end:=true;
 number_of_nodes:=asn_matrix[net_number].number_of_nodes;
 number_of_events:=asn_matrix[net_number].number_of_events;
 For j:=1 to number_of_nodes do
  begin
   If (asn_node_definition[net_number,j].node =
        activity_initiator) then no_activity:=false;
   If (asn_node_definition[net_number,j].node =
        terminator) then no_end:=false;
  end;
 ClrScr;
 writeln;
 If no_activity then
  writeln(' WARNING! This ASN does not contain any activity nodes.');
 writeln;
 If no_end then
  writeln(' ERROR!!!   This ASN is missing its END node.');
 If ((not no_end) and (not no_activity)) then
  writeln(' No important node (activity or end) is missing.');
 writeln;
 write('      Press <CR>.');
 readln;
 Clrscr;
 For i:=1 to number_of_events do
  begin
   If (asn_event_definition[net_number,i].event = ext_event) then
    no_external_event:=false;
  end;
 writeln;
 If no_external_event then
  begin
   writeln(' ERROR!!!  This ASN does not have any external event.');
   writeln;
   writeln('                    Press <CR>.');
   readln;
  end;
 writeln;
 Clrscr;

End;

{_____________________________________________________________}
```

```
        ( Checking of the matrix. )

{ The purpose of this file is to allow the basic verification    }
{ of the matrix parameters.                                      }
{ We are checking the following :                                }
{ - that all nodes are triggered;                                }
{ - that all events trigger something;                           }
{ - that the matrix contains at least an external event;         }
{ - that the matrix contains the starting event;                 }
{ - that the triggering of nodes is correct, that is that special}
{   nodes such as counters and gates are enabled and disabled;   }
{ - Similarly, that the other nodes, wait, activity initiators,  }
{   branch end and merge nodes are either triggered or not       }
{   triggered;                                                   }

PROCEDURE MATRIX_CHECKING;

var choice,i, number, h, choice_number : integer;
    who :word;
    ASN_found: boolean;

Begin

 If (color_terminal) then
  begin
   Textcolor(15);
   TextBackground(1);
  end;
 choice_number:=0;
 choice:=0;
 window(1,1,80,25);
 ClrScr;
 while (choice <>7) do
  begin
   choice_number:=0;
   ClrScr;
   writeln(' These are the ASN''s of system ',system_name,' .');
   writeln;
   For i:=1 to number_of_nets do
    begin
     writeln('  Net number ',i:2,' is : ',ASN_label[i]);
     If ( i = 18 ) then
      begin
       writeln;
       write('      Press <CR> to continue.');
       readln;
       ClrScr;
      end;
    end;
   writeln;
   write(' Enter the NAME of the ASN to check : [4 characters] ');
   who:=read_name;
   ASN_found:=false;
   ClrScr;
   For h:=1 to number_of_nets do
    begin
     If (ASN_label[h] = who) then
      begin
       ASN_found:=true;
       number:=h;
      end;
    end;
   If ASN_found then
    begin
     writeln(' ASN being checked is ASN number ',number:2,' .');
     writeln;
     write(' Press <CR>.');
     readln;
     ClrScr;
```

```
    net_number:=number;
    Checking_menu;
    If (color_terminal) then
     begin
      Textcolor(15);
      TextBackground(1);
     end;
    window(1,13,80,25);
    ClrScr;
    writeln;
    write(' Enter choice number : ');
    choice_number:=read_number;
    while ((choice_number <>6) and (choice <>7)) do
     begin
      case choice_number of
       1:Check_event_node_triggering;
       2:Check_basic_connectivity_anomalies;
       3:Check_lack_of_nodes;
       4:
        begin
         Check_event_node_triggering;
         Check_basic_connectivity_anomalies;
         Check_lack_of_nodes;
        end;
       5:begin
          window(1,1,80,25);
          ClrScr;
          Epsilon_verification;
          Clrscr;
          window(1,13,80,25);
         end;
        7:begin
           choice_number:=6;
           choice:=7;
          end;
      end; { end of case }
     If (choice <>7) then
      begin
       Clrscr;
       Checking_menu;
       If (color_terminal) then
        begin
         Textcolor(15);
         TextBackground(1);
        end;
       window(1,13,80,25);
       ClrScr;
       writeln;
       write(' Enter choice number : ');
       readln(choice_number);
      end;
     end;
    end
   else
    begin
     ClrScr;
     writeln;
     writeln(' ASN to check has not been created.');
     writeln(' It can not thus be checked.');
     writeln;
     writeln('  Press <CR>.');
     readln;
    end;
   end; {end of while}
   window(1,1,80,25);
   ClrScr;

End;

{_____________________________________________________________}
```

```
PROGRAM ASN_MATRIX_INSPECTOR (input,output);

{ MAIN PROGRAM }

{ The purpose of this program is to verify the correction of     }
{ matrix entered. It is part of the PAD software tool. Once the  }
{ system has been defined and entered by means of the EDITOR,    }
{ the user will call the checker to verify if any errors have    }
{ been inserted in the inadvertantly in the matrix definition.   }
{ Once the correctness has been proven, the definition can be    }
{ tested on the simulator.                                       }

{_________________________________________________________________}

{$R+}
{$I DECLAR.GLO}
{$I FILECHEC.GLO}
{$I READSYS.GLO}
{$I SYSLIST.GLO}
{$I INSMENU.INS}
{$I BRANCHEC.INS}
{$I MECACHEC.INS}

{_________________________________________________________________}

{ MAIN PROGRAM }

VAR answer, response : string[1];
    choice, choice_number : integer;

BEGIN

 ClrScr;
 writeln;
 writeln;
 write('    Do you have a color monitor ? (Y/N) ');
 read(kbd,answer);
 answer:=upcase(answer);
 If answer='Y' then color_terminal:=true
  else color_terminal:=false;
 ClrScr;
 GotoXY(5,3);
 choice:=0;
 while (choice <>4 ) do
  begin
  writeln;
  write('   Enter system definition name [ 8 characters ] : ');
  readln(system_name);
  Read_system_labels;
  Read_system_definition;
  Read_system_characters;
  choice_number:=0;
  window(1,1,80,25);
  ClrScr;
  writeln;
  writeln('                    ASN ERROR DETECTOR.');
  writeln('                    -------------------');
  writeln;
  writeln('  INSPECT is an error detection program designed to locate');
  writeln(' any wrong entry into the definition of the ASN.');
  writeln;
  writeln('  The types of errors that can be found are :');
  writeln;
  writeln('   1- Incorrect entries in the matrix itself;');
  writeln('   2- Incorrect connections in the matrix;');
  writeln('      This is a source of deadlock or ');
  writeln('        multiple triggering.');
  writeln('   3- Incorrect use of branch and merge nodes;');
  writeln('      This is a source of deadlock or ');
```

```
writeln('       multiple triggering.');
writeln('   4- Lack of basic features of a given ASN;');
writeln('   5- Combinations of errors.');
writeln;
writeln('   This program was written by Jose M. DURAN.');
writeln;
write('                                          Press (CR).');
readln;
ClrScr;
While ( choice_number<>3) do
 Begin
  Window(1,1,80,25);
  If Color_terminal then
   begin
    TextColor(15);
    TextBackground(1);
   end;
  ClrScr;
  GotoXY(17,5);
  writeln('    MENU OF COMMANDS.');
  GotoXY(17,10);
  writeln('1. List System Parameters.');
  GotoXY(17,12);
  writeln('2. Start System Verification.');
  GotoXY(17,14);
  writeln('3. Change System.');
  GotoXY(17,16);
  writeln('4. Quit.');
  GotoXY(17,22);
  write('  Enter choice number : ');
  readln(choice_number);
  printing:=false;
  case choice_number of
   1: List_system_parameters;
   2: begin
       ClrScr;
       gotoXY(3,3);
       write(' Do you want a printed format of the errors ? (Y/N) ');
       read(kbd,response);
       response:=upcase(response);
       If response='Y' then printing:=true;
       ClrScr;
       If not printing then
         Matrix_checking
        else
          Print_Matrix_checking;
      end;
   4: begin
       choice:=4;
       choice_number:=3;
      end;
  end; { end of case }
 end;

End;
ClrScr;

END.

{_____________________________________________________________}
```

UNIVERSITÉ D'OTTAWA
UNIVERSITY OF OTTAWA