

**ON THE DEVELOPMENT OF REAL-TIME SOFTWARE  
BY GRAPHIC CAD/CAM METHODS**

by H.G. Mendelbaum, M. Reiff, D. Pisanti  
B.Z. Fridman, Y. Levian, Y. Maoz  
(Jerusalem College of Technology  
Industrial Programming Lab, 21 havaad haleumi Str.)

**A) INTRODUCTION :**

Highly sophisticated technology is a vital need for future industry. One important field is the "PROCESS CONTROL" software technology, which includes the programming of computerized industrial machines, robot-control programming, flexible manufacturing software systems etc...

An other important feature is the development of micro-computer interactive graphic systems which are very useful for cheap CAD/CAM and industrial programming design.

Generally in software development, there is a distinction between four activities: problem specification, software design, then programming and execution tests. Each of these activities has its own methods, languages or systems.

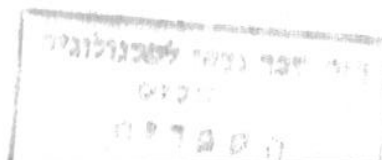
For instance, SADT or ISDS/HOS are methods for system requirements specification (1). In software design, we can use Petri-flow-charts (2)(3)(4) describing the flow of control (actions) of the application, or we can use state-diagrams (5)(6) describing the flow of events (state-transitions) in the system, or we can use data-flow techniques. There are also specific languages (7)(12).

In the programming activity we can use one of the numerous real time languages: from the oldest RT-Fortran (8) to the newest ADA (9).

These last years, there is a tendency to create big computer systems giving a programming environment which integrates: specification and design tools, programming languages and executive operating systems (10)(11)(13).

In this Paper, we study a very cheap and simple integrated environment to design and produce software for industrial real time control or robot command, using flow-charts on graphic micro-computers.

The main idea is to develop a technology which enables to design and describe graphically **Real Time Applications** avoiding the language level and enabling the design-graphs to be translated directly into executable code.



This can now be done with cheap micro-computer based on interactive graphic systems : In this paper we present such a simple integrated environment built for Macintosh computers including a **flow-chart graphic editor**, an **interpreter** which checks interactively the flow-chart building errors, and an **executor** to perform the application described in the chart.


## **B) FLOW-CHART DESIGN :**

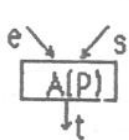
1) **CHOICE :** We choosed PETRI-like design-graphs for several reasons :

- Δ we think that a graphical representation of the logical flow of an application is simpler for the engineer than a language representation;
- Δ Petri-like graphs give the flow of control (actions) in the application, this is closer to the way of thinking of the automatician, who deals more often with actions and events (signals) than with flows of data;
- Δ Nevertheless, each action is viewed as a procedure with parameters and data, which enables to represent also the flow of data;
- Δ the Petri-like graphs enable to represent in a unified mode all types of actions (I/O, commands, computations, tests etc...);
- Δ the Petri-like graphs enable to represent synchronizations between events or actions, triggering conditions and timing;
- Δ the Petri-graphs have been studied in various R&D centers in the world and have a large theoretical background, this means that they can be validated and proved as "safe", "clean", "alife" etc...(2X3X4)
- Δ Numerous applications have been made in the world for hardware or software using Petri modelling : such as in telephone switching (14), space industry (15), mechanical industry (16) etc...

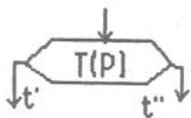
## 2) **DESIGNING RULES :**

The **Real-Time-Charts** that we use, are an interpreted form of PETRI-graphs and there are biunivoqual relations between them. The components of these charts are :

 is an arrow relating the various components ( also called internal event);



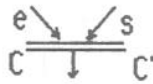
is an action A activated either by event e or s (inclusive OR) and giving event t at the end of its execution. This action is called procedure A with parameters P;



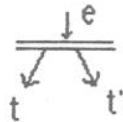
is a procedure T of tests (on variables or sensors) with two possible output-events t' or t'' (exclusive OR)



is only the expression of an " inclusive OR" between various events (it is considered as a null action);



is a transition-gate allowing to wait on condition C after the occurrence of events e "AND" s ; and then to continue the following actions. If we have the ill-condition C' when e and s have arrived , there is an error;



is a transition-gate allowing to make several actions in Parallel (in the same time)

According to Petri rules, there must be always a transition-gate between two actions (or tests); this means that there must be always an explicit condition to pass from one action to another;

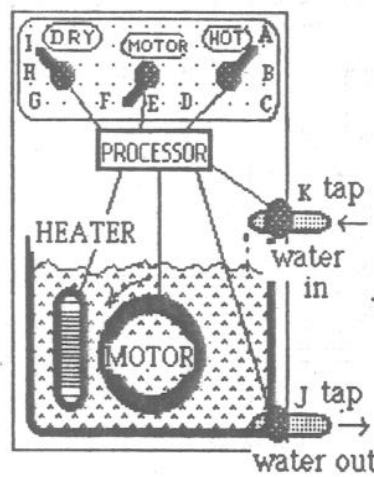
Another Petri rule is that you cannot put two transition-gates one after the other, this means that it is not necessary to split a condition into two following parts, since if there is 2 conditions occurring at the same moment they can be expressed in the same equation;



is an external event (signal) which can be connected only to a transition-gate . This signal can come from a timer or from any external device (interrupts);

### 3) APPLICATION :

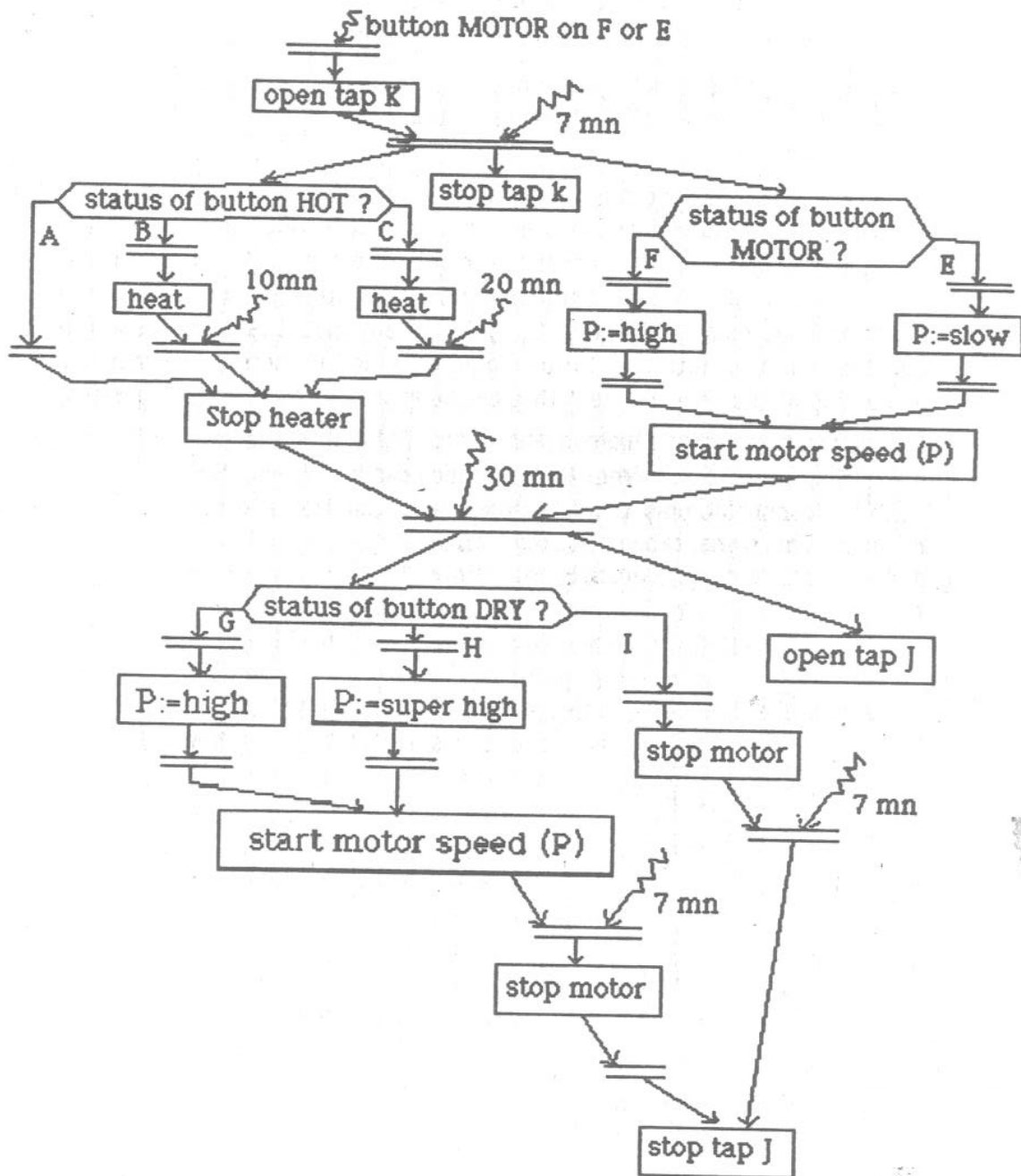
A simplified example of designing an application with these real-time-charts. Let's take a washing-machine with only 3 buttons :



- HOT A = cold
- ☑ B = warm (10 mn heating)
- ☑ C = very warm (20 mn heating)
- MOTOR D = stop
- ☑ E = low speed
- ☑ F = High speed
- the motor works during heating time and then 30 mn
- DRY = tap J open 7 mn
- ☑ G = high speed
- ☑ H = super high speed
- I = without motor

The user of this washing-machine must first prepare the buttons 'HOT' and 'DRY', and afterwards the button 'MOTOR' in order to start the washing process. The machine stops alone. Then the user may turn the button 'MOTOR' to the position D.

A design Real-time-chart describing this process is as follows:



On this design real-time-chart, we can see that the processor of the washing-machine starts the washing process only when it receives a signal ' button MOTOR turned on F or E '. Then, it opens the tap K to pour in water, waits 7 mn, and starts 3 operations in parallel : it stops the tap K, and tests the status of the buttons MOTOR and HOT. According to the positions of these buttons, the processor will start the motor and the heater with the corresponding parameters of values and timings. Afterwards it waits 30 mn more, while still moving the motor, then it will start the drying process according to the position of the button DRY. Finally it will stop the motor and the tap J.

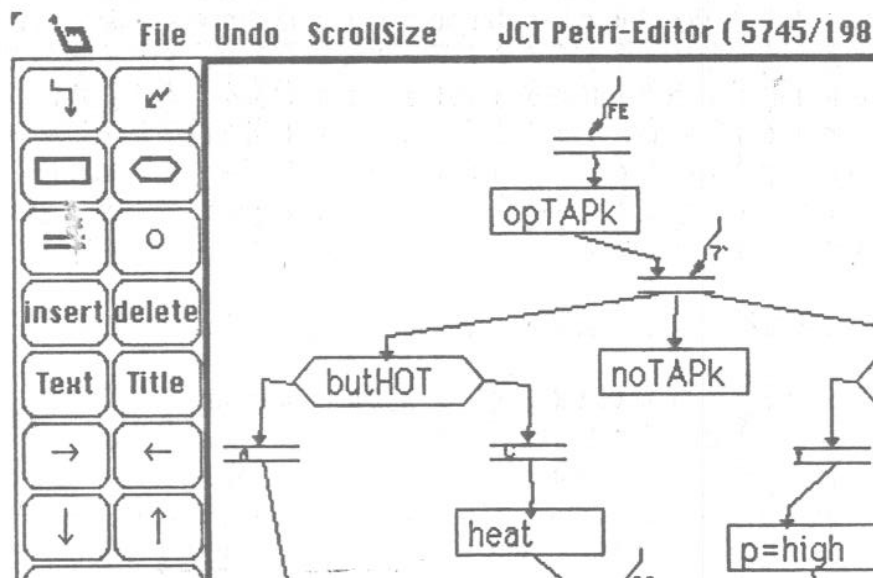
### C) GRAPHIC DEVELOPMENT SYSTEM :

We built a software-environment based on the object-oriented graphic-technology of the Macintosh Computer (17) with multiple windows on the screen, scrolling menus, interactive dialogs and mouse-pointing. Our developer's integrated environment is made of 3 pieces :

#### 1) A FLOW-CHART GRAPHIC EDITOR :

This complete interactive editor works with the elementary figures (Arrows, External Events, Actions, Tests, Gates, Ors) which are the components of our Petri-like graphs (see section B 2 above) . These figures appear on the left side of the screen. The engineer uses the mouse to choose them and build his graphic real-time-chart on the right side of the screen. The chart can be larger than the screen.

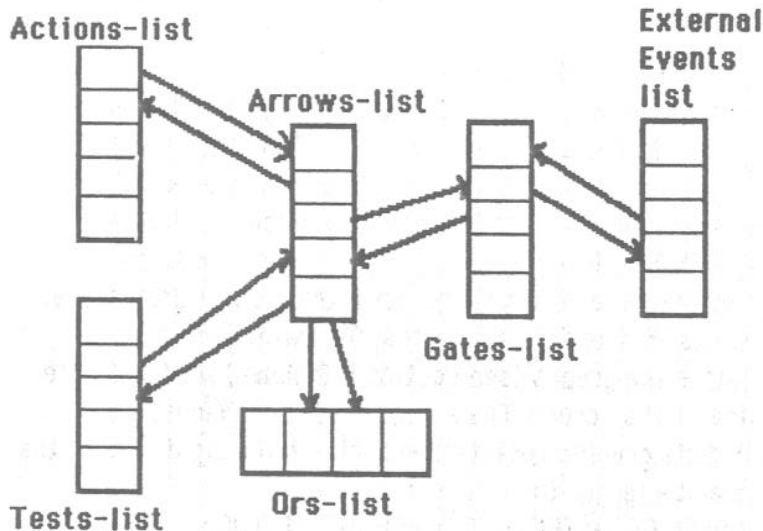
#### PART OF THE SCREEN OF THE REAL-TIME-CHART EDITOR :



While building his graph with the mouse , the user has the possibility to scroll the chart on the screen, to insert new figures in the graph, to delete figures, to modify them, to rename them, to save the chart on a disk- file, to recall it , to print it etc...

Furthermore, with the editor the engineer has also the possibility to write the text of new action-procedures and to give the waiting-conditions of the transition-gates.

The real-time-chart is not represented in memory as a bitmap, it is in a condensed form as a structure of 6 'figure-lists' which are interconnected through pointers, depending on the connections between the Petri-like figures on the screen:



For each type of elementary figures (Arrows, External Events, Actions, Tests, Gates, Ors) there is a list describing the figures existing in the edited real-time-chart.

For each figure there are: its coordinates on the screen, its name, its parameters and pointers connecting each figure with its neighbouring figures in the graph. The interpreter fills in these figure-lists each time a new figure is introduced and accepted. The texts of the procedures are also saved with the figure-lists.

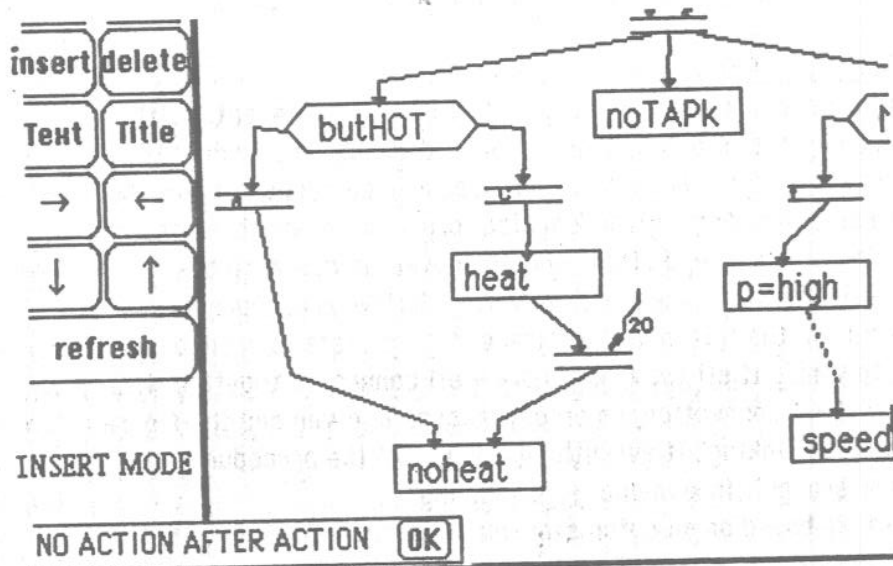
## 2) A FLOW-CHART GRAPHIC INTERACTIVE INTERPRETER :

During the editing time of the graph, after each insertion of a new figure in the graph by the user, the interpreter tests the chart to see if it fits to the rules of Petri-like graphs (see section B 2 above). If it does not fit to these rules, it gives immediately an error message and deletes the new figure inserted in the graph. Typical error messages are of the form : NO ACTION DIRECTLY AFTER ACTION , NOT 2 ARROWS AFTER ACTION , NO EXTERNAL EVENT BEFORE TEST, NO GATE DIRECTLY AFTER GATE etc.. There are 20 types of such logical errors detected interactively at editing time.

In such a graphic editor, the user may arbitrarily design his graph as he wants on the 2-dimensional chart, for instance: he may first insert a gate, a following action and then connect them with an arrow, or he may first insert a gate, then the arrow and afterwards the action, or he may insert first the arrow and then the action and finally the initial gate etc....

In all the cases the interpreter must test and scan the figures actually present in the chart in order to connect this new figure to the already existing ones.

EXAMPLE OF AN INTERPRETER'S INTERACTIVE ERROR-MESSAGE:  
when the user tried to connect directly 'P=high' to 'speed'



This implies an interpreting technics using a special grammar describing both

- ◊ the correctness rules of the Petri-like graphs,
- ◊ the possible ways of building the graph.

For this, we found that the easiest method was to use ternary-succession-rules which describe the authorised flow of figures in the chart:

- #1 Action → Arrow → Gate
- #2 Test → Arrow → Gate
- #3 Or → Arrow → Gate
- #4 External Event → Gate
- #5 Gate → Arrow → Action
- #6 Gate → Arrow → Test
- #7 Gate → Arrow → Or

All other combinations of figures lead to an incorrect graph.

Each time a figure is introduced, the interpreter activates a special algorithm for scanning the 2D-figures and connect them together. For instance, if the user inserts a new Gate: the interpreter must seek what figures are already around the Gate on the screen (after it and before it) and what figures can be connected to it.

To find what figures are after the Gate, the interpreter tries to apply the ternary-succession-rules #5 #6 #7. It will seek the neighbouring Arrows and their connected Ors, Actions or Tests. If authorized figures are found, the connections can be set up as pointers between the corresponding figure-lists and the new Gate.

To find what figures are before the Gate, the interpreter makes the same kind of search using the rules #1 to #4, but backwards.

If the interpreter finds neighbouring figures which are not in the ternary-succession-rules, it edits an error message as those quoted above.

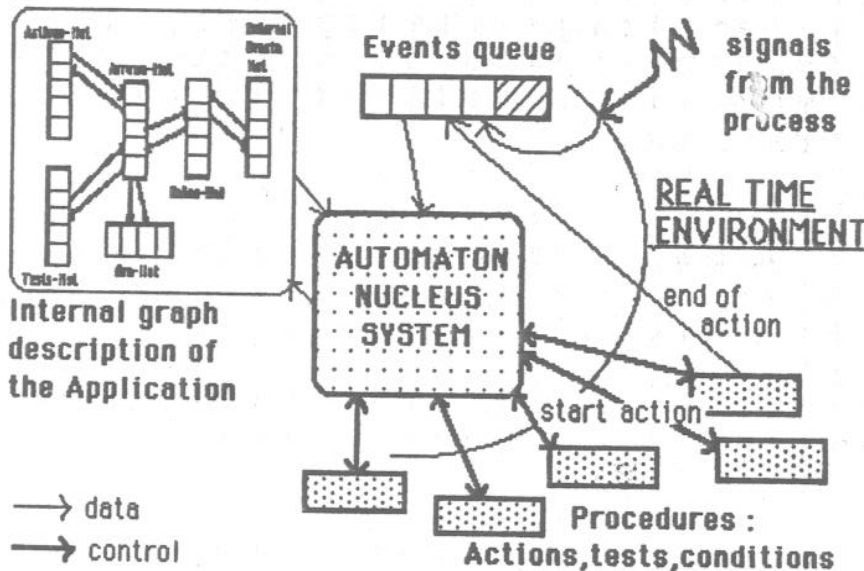
All these connections and these verifications are made each time the user inserts or deletes a figure in the graph, in a time shorter than the engineer reaction-time using the mouse (generally 0.5 to 1 sec).

### 3) FLOW-CHART EXECUTOR :

The executor uses 3 types of library : a list of external events with the corresponding physical signals, a list of procedures corresponding to the gates waiting-conditions, and a list of procedures corresponding to the various actions and tests. For a given 'physical process' these libraries are quite standard since they depend strongly on it, nevertheless with the editor it is possible to extend them and to write additive procedures.

Before execution, the internal structure of figure-lists is scanned finally in order to verify if all the figures are well connected together. If a figure remains without connection, an error message is given and the figure in question is shown 'blinking'. If everything is O.K, all the procedures and the signal list are brought into memory for linking.

The execution is based on an automaton-nucleus-system (10) which makes the interface between the physical process and the internal description of the real-time-chart which represents the application to perform :





The automaton-nucleus-system scans continuously the physical process and the internal condensed form of the graph. This automaton-nucleus system receives, waits or tests the conditions or signals of the process according to the graph description, and it activates the corresponding action-procedures which are to be performed at each moment.

#### D) CONCLUSION :

At the JCT, in our Industrial Programming Laboratory, our graphic development system is running on Macintoshes connected to Eshed-Robotec robots, for which we have built the external events and procedures libraries (move shoulder, roll wrist, test elbow switch etc...). The system allows to build graphically various robot applications using this hardware and these elementary actions. Computational procedures may be added. The system is easily adaptable to another type of process, in this case we have only to write a new library of actions (depending on the physical features of this new process hardware), all the design system will remain the same.

#### BIBLIOGRAPHY :

- (1) SADT, Report Softech Inc., 460 Totten Pond Rd, Waltham MA 02154  
ISDS, Report HOS Inc., 843 Massachusetts av., Cambridge MA 02139
- (2) C.A. PETRI "Introduction to a general Net Theory"  
Lecture Notes in Computer Sc. N°84, Springer Verlag Ed., 1979, p.1-19
- (3) G.W. BRAMS "Réseaux de Petri", Masson Ed., Paris, 1982
- (4) J.L. PETERSON, "Petri nets", Computing surveys, vol. 9, N°3, Sept. 1977
- (5) Bruce TAYLOR, "Expressing functional requirements", Report 1980  
GTE Lab., 40 Sylvan Rd, Waltham MA 02154
- (6) David HAREL, "statecharts", Dept. Applied Maths, Weizmann Inst., 1984
- (7) H.G. MENDELBAUM. "GAELIC, a real-time global description language"  
IFAC/IFIP real time programming workshop, Eindhoven, 1979
- (8) Industrial FORTRAN standard, ISO, TC 97/SC 5/ WG 1, revised dec 1976
- (9) ADA reference manual, DOD, DARPA, 1980
- (10) L.S. ORZECH "PSL/PSA: a computer aided tool for High level design"  
IBM/FSD Software Engineering Exchange (Jan. 1980)
- (11) K.L. HENINGER, "Software specification", IEEE trans. Soft. Eng., Jan. 1980
- (12) P.E. LAUER et al. "Cosy: paths specification" Acta Informatica 12, 1979
- (13) M. BEAUDOIN-LAFON, C. GRESSE "CATY: Construction graphique  
interactive de programmes" Revue T.S.I., Paris, 1984
- (14) M. YOELI, Z. BARZILAI, "Extended Petri Nets for switching systems"  
Digital Processes vol 3, 1977, Georgi Publ., CH-1813 St-Saphorin
- (15) J. ALBUKERQUE, "Specification d'automatismes interconnectés",  
Doctoral Thesis 3°C, Univ. Sabatier, Toulouse, dec 1982
- (16) J.C. BOSSY et al, "le Grafcet", Educative Ed., 1979, Paris 7°
- (17) B.J. COX, "Object-oriented programming for software craftsmen"  
Unix Review, Febr/march 1984
- (18) H.G. MENDELBAUM, "ABOS, automate based realtime Operating-System"  
IFAC Symposium on Control Software, Tallin, 1977