IJPR, 1998

STATECHARTS TO SPECIFY THE CONTROL OF AUTOMATED MANUFACTURING SYSTEMS

J.C. MARTY(1,3), A.E.K SAHRAOUI(1), M. SARTOR (2)
(1) Laboratoire d'Analyse et d'Architecture des Systèmes du
Centre National de la Recherche Scientifique L.A.A.S.-C.N.R.S.
7, Avenue Colonel Roche, 31077 Toulouse, France.
and IUT-B Université de Toulouse II 31073 Blagnac
e-mail : sahraoui@laas.fr
(2) Département de Génie mécanique
Institut National des Sciences Appliquées
Avenue de Rangueil 31400 Toulouse
e-mail : sartor@gmm.insa-tlse.fr
(3) Actually at Dassault systemes Suresnes, France

This paper deals with the use of Statecharts formalism, a recent automata-based language, to specify the control of automated manufacturing systems (AMS). Statecharts has been developed for the description of complex reactive systems ; it provides high level constructs which allows to carry out large problems by a graphical structured approach.
The main features of the formalism are given as introduction. A way to design the most important constructs and mechanisms of control field is proposed . Then, these constructs are implemented in order to specify the control of a flexible manufacturing cell and to show the adequacy of the Statecharts for AMS.
Others useful constructs are given allowing to translate control specifications according to a structured analysis method. Generic specification is also presented and the graphical simplification they provide in case of generic behaviour handling is enlightened. Extensions made on Statecharts support are illustrated.

Keywords : FMS, control, specification, reactive systems, Statecharts, real-time, manufacturing.

1. Introduction to the specification of systems

We denote by "Automated Manufacturing System" (AMS) all kind of automated machine used in industrial factories to process parts or products. Fig. 1 gives the general structure of AMS.

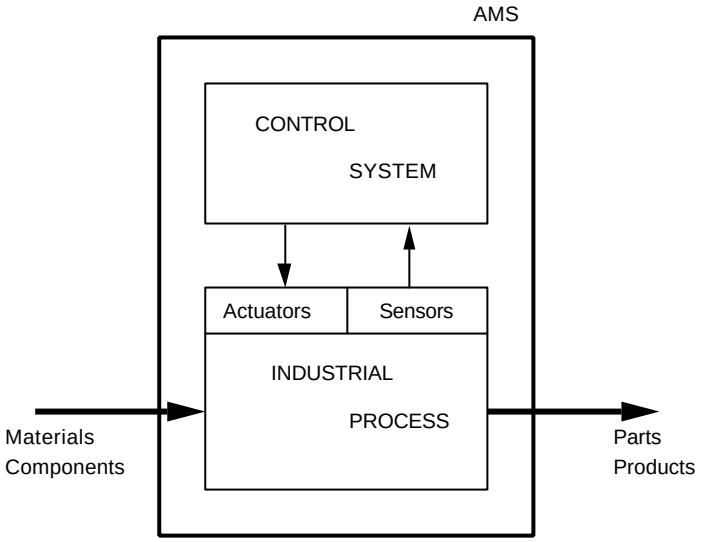

Fig. 1.

AMS are generally dedicated machines built in small quantities. Then, the development phase has to be done efficiently enough in order not to increase too much the system cost. It is actually well known that the specification stages have a great impact on the quality and the cost of the resulting system.

The complexity in specifying such systems is due mainly to the control structure and the various operation modes of the production system.

1.1 system architecture

The often adopted control structure is made of five levels of abstraction. This is illustrated by the following figure

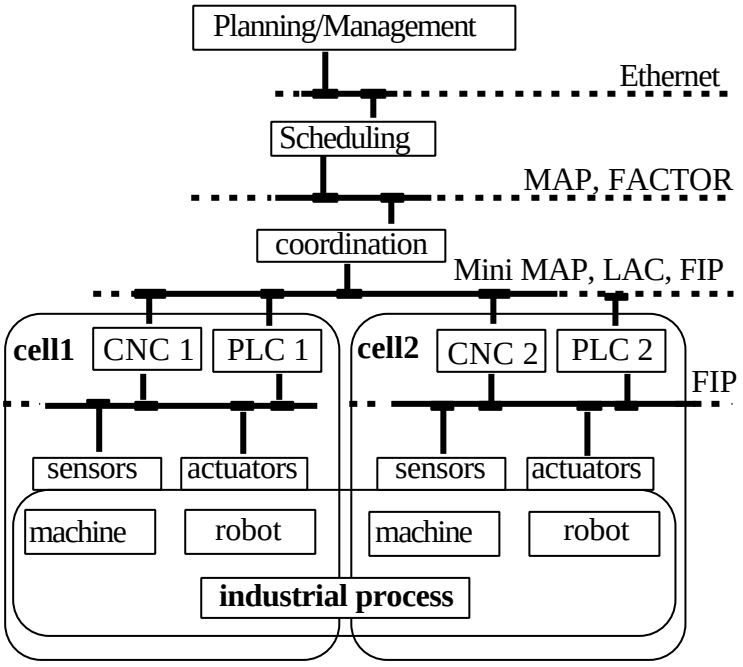

Figure 2

- Level 4 (planning): dealing with all aspects of management functions,long term and mid term planning,
- Level 3 (scheduling): function dealing with allocation of machine. This function is at first triggered off-line and then behaves as a dynamic task, the results are function of all parameters and available resources in the system,
- Level 2: coordination of all manufacturing cells,
- Level 1: machines real -time control,
- Level 0: intelligent sensors and actuators

1.2 Specification aspects

In any system development approach, a great importance is given at the specification phase. The most common used in industrial application are the semi-formal specification tools : Transformation schema, SREM, Ward-Mellor approach etc ...There is actually a great interest in using the formal approach : Specification languages with mathematical basis (formal semantics); the great advantage is the verification aspect which is not available in other approaches.

The specification and implementation of reactive systems are becoming an important issue in various research field such as parallelism and concurrency, distributed systems. When deciding to build a program to control and to monitor industrial processes, the first difficulty to appear is what software development approach to use; and for each development , there are many steps to follow. These steps can be

- Requirement or design specification
- System specification
- Design
- Implementation and coding

In a requirement specification of real-time systems, it is important to take into account their timing and functionnal behaviour.

At each step of the development cycle, there are many proposed formalisms to deal with them. However, there are a lot of misunderstandings about the system specification; it is necessary to define the difference in between by classifying the formalisms with respect to their descriptive power at each step.

When dealing with system development for real-time systems at the specification phase, we rarely in most cases use criteria for selecting a particular formalism. The most common criteria is either fast prototyping, verification or easy of use; building critical software may require other criteria. Hence, we can state without any loss of generality that the purpose of software determines the choice of development method and also the associated specification tools. That is why it is tried to present in this paper a taxonomy of specification tools and a comparative study through their descriptive power.

It becomes cumbersome to deal with all proposed approaches without classifying them with some criteria. The criteria adopted is the degree of formalism for each specification language. The formalism determines the verification step. Hence that the reason we deal with formal tools. It is worthwhile to mention that semi-formal methods are well adapted to specification in the large where there is a higher level of abstraction; on the other side, transition system based methods are appropriate for specification in the small. Discret event control is considered at the lower levels of abstraction.

In this paper, we deal with specification tools for AMS control systems. AMS become more and more complex, and specifying it is an hard problem. To succeed, the specification work has to begin by the design of a functional general architecture. Many multi-layer/multi-level architectures are proposed. Then, each function has to be specified in details.

In software development field, functions are often transformational dominant ; the most common methods used in industrial application are based on semi-formal specification tools : SA *(DeMarco,T. 1979)*, SADT...

In reactive systems development field, a lot of functions are indeed control dominant. Real-time extensions of the previous semi-formal methods have been developed : Transformation schema (Ward,P. 1986), SA-RT *(Hartley,D and Pirbhai,M. 1987)*. But there is presently a great interest in using methods based on formal tools : specification languages with formal semantics ; their major advantage is the verification aspect which is not available in other approaches. Among these, there are :

- Petri nets, the mostly used tool for discrete event systems (Valette, R 1983, Krogh,B 1990) ;
- Temporal logic, used in the verification of sequential and parallel computation [Pnueli, A 1988, 1991] ;
- Synchronous languages *(Berry,J.P 1987, Benveniste,A 1990)* ;
- Statecharts, a recent tool *(Harel,D 1987a, 1987b)*.

Statecharts has the benefits of all results obtained in the last two decades in the field of graphical tools. To our knowledge, Statecharts have not been used until now in AMS field. We intend to give some elements in order to show that this tool, which provides the classical state-transition diagram formalism plus extended mechanisms (depth, orthogonality, broadcast-communication), is a very interesting way for the design of AMS.

1.3 problem issues in manufacturing system specification

As presented in part 1.1, the structure consists of many abstraction levels; a specification language must offer this type of construct. A survey carried out *(Sahraoui, A and Gilhodes,L;. 1991, Sahraoui,A. 1992)* had the purpose in comparing several formal and semi-formal methods for manufacturing systems ; statecharts method appeared the most appropriate in specifying, analysing and prototyping control of manufacturing systems. Among these, we considered in this study :

- Petri nets the most used tool for discret event systems and mainly manufacturing systems. An experience has already been carried in the French ARA research project (Advances in Robotics and Automation). Sub-project SECOIA was focussed on the use of Petri nets and associated models to specify the the three lower levels presented in figure 2.

- Temporal logic, being used in the verification of sequential and parallel computation as a logic approach to manufacturing systems.
- State charts , the recent proposed approach with an automata based structure. The use of state charts for some powerful constructs as
  - History and recursive history
  - Hierarchy

It is well known among the academic community that none of the following formalisms : temporal logics, transition systems (automata based systems), process algebra and assertional methods completely solves the problems in real-time specification and design, hence each of them has its own deficiency. We try in this paper to contribute to this study from application oriented view point and specifically to design discret event controllers. These controllers are mainly located at the lower levels.

From this study, we carried a work proposed in the present paper in considering statecharts to specify manufacturing systems and applied to a real scale manufacturing cell.

2 - Statecharts and AMS specification

2.1 - Introduction

Five common constructs are fundamental in control design :

- sequence,
- alternative,
- loop,
- concurrency (and synchronisation),
- resource sharing.

We will base on an example to illustrate :

- how Statecharts can express these constructs,
- how these constructs can be implemented and combined in case of specification of a manufacturing system.

We choose a rather complex example in order to apply all the constructs and show that difficult cases can be solved.

2.2 - Presentation of the example : flexible manufacturing cell

Our running example will be a flexible manufacturing cell (FMC). As shown in Fig. 3, it consists of 9 stations and 4 pallets.

The stations are :

- station A = Resource RA (a tool centre) + post A ;
- station B = Resource RB_I and RB_{II} (two special machines) + post B ;
- station M = the Manager of the cell ;
- station O = Resource RO (an Operator) + post O + set of parts α + set of parts β ;

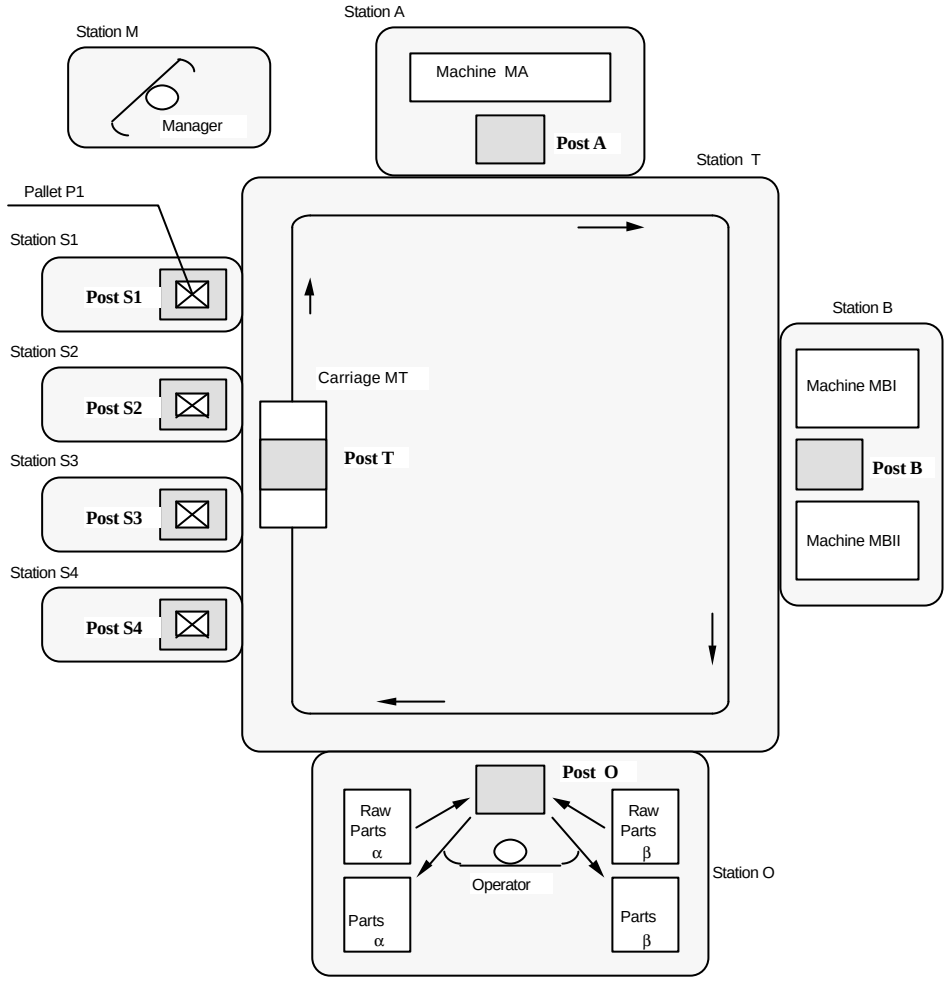

Fig. 3.

- stations S1, S2, S3, and S4 = posts S1, S2, S3, and S4 ;
- station T = Resource RT (an automated Transfer carriage) + post T + circulation net.

The four pallets P1, P2, P3 and P4 are identical ; each one is an universal support on which the operator can fix a part. Pallets never leave the cell. The places that each pallet Pi can occupy at any moment is one of the following posts :

- post A : in this post, the machine RA executes an operation OA on the part fixed on the pallet ;
- post B : in this post, machines RB_I and RB_{II} execute simultaneously operations OB_I and OB_{II} on the part fixed on the pallet ;
- post O : in this post, the operator RO takes the finished part off the pallet (unloading : operation OOu), and put a new part on the pallet (loading : operation OOl) ;
- post Si : in this post, the pallet is kept in a waiting state ; Si is reserved for Pi ;
- post T : it is a moving post ; the carriage is used to transfer the pallet from a static post to an another.

The cell has to produce two types of parts : α and β . The operating sequences are :

- for α : 001α --> OAα --> OB_I and OB_{II} --> OOuα
- for β : 001β --> OB_I and OB_{II} --> OAβ --> OOuβ

Remark. Only operations OB_I and OB_{II} are identical for both types of parts.

At starting, we consider that the cell is in its "reference state" : all four pallets Pi are empty and located at the waiting posts Si.

The manager's mission is to supervise the general operating modes. He triggers the system off from the "Stopped in Reference" mode to the "In Production" mode through giving the start order. He triggers the system off from the "In Production" mode to the "Return in Reference" mode through giving the stop order. When the system reaches the reference state, it enters automatically in the "Stopped in Reference " mode.

In IP mode, the pallets which are empty (at starting) or which carry a finished part are systematically driven in the station O in order that the operator loads a new part. The operator may choose either part α or β.

The four pallets must share the stations A, B, O and T. There will be conflicts for accessing these resources. We make some priorities to manage the conflicts : P1 > P2 > P3 > P4.

2.3 - Illustration of the sequence construct

It is a linear simple oriented graph, where nodes denote states and arrows denote transitions. The diagram easily specifies the order whereby the state of the system changes, or the state of a fragment of the system changes. In our case, we can describe by a sequence the evolution of the pallet P1 state when P1 leaves its initial state (empty, in the station S1), to go to the station O. See Fig. 4.

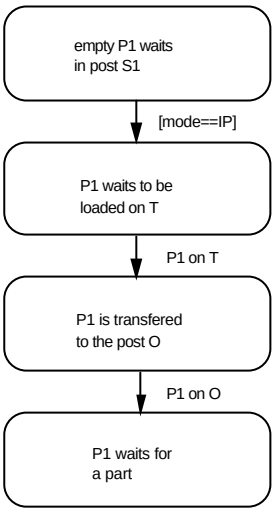

Fig. 4.

2.4 - Illustration of the alternative constructs

In Statecharts, each transition will be labelled with an expression. When one wants to specify two evolutions from the same state, the construct of the transitions may differ depending on events and conditions combination. Let consider, in our example, the two following cases :

- when the pallet P1 is receiving a part in the post O, according to the choice of the operator, the loading operation OO1 terminates in event "End of OO1α " or "End of OO1β ". Depending on the event, the pallet will go to either post A or post B. See Fig. 5.

- when the pallet P1 arrives on post A (event "P1 on A"), the machine RA has to execute either operation OAα or OAβ according to the type of the part fixed on P1. We have to introduce a variable pP1 which will indicate the part type. The evolution depends on the value of pP1. See Fig. 6.

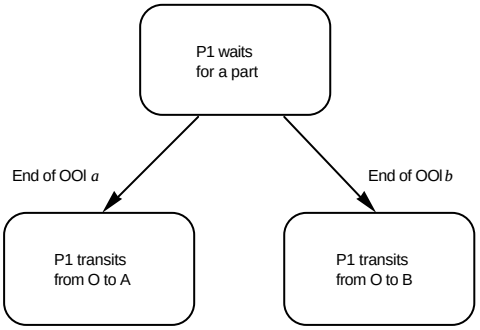

Fig. 5.

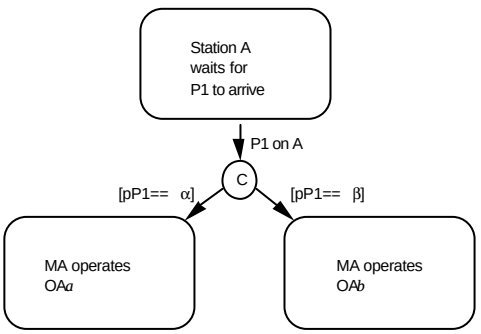

Fig. 6.

2.5 - Illustration of the loop construct

The loop construct is used whenever a sequence has to be repeated. The type of reasoning is the following : a condition is evaluate ; if it is true, a sequence is executed and the condition is re-evaluated ; this cycle starts again until the condition becomes false, at which point execution continues with the next part. Fig. 7 shows the corresponding construct using a conditional transition. In our example, the pallet P1 repeats the same production cycle while the mode selected by the manager is the IP mode. See Fig. 8.

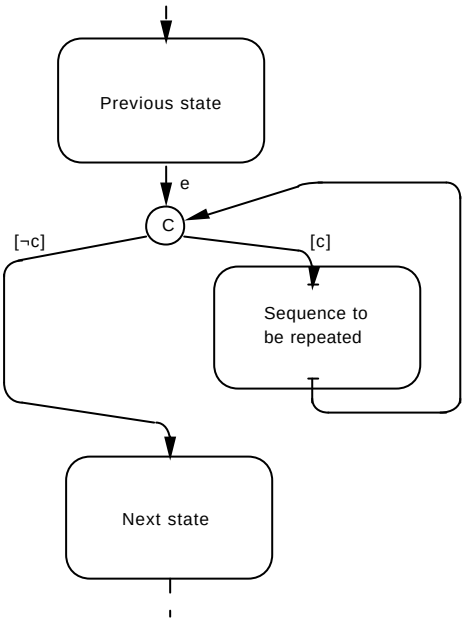

Fig. 7.

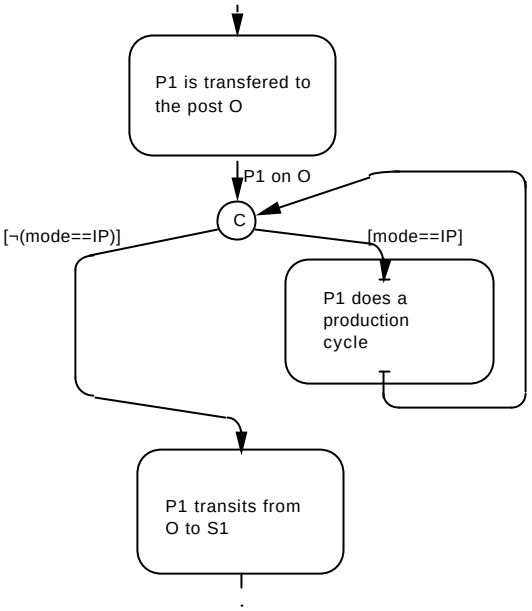

Fig. 8.

2.6 - Illustration of concurrency

Concurrency, noted orthogonality in Statecharts, is a powerful construct for specifying simultaneous execution of more than one process. In Statecharts, concurrency is introduced as communicating state-machines called AND-states. When decomposing a system, we have sub-systems which evolve independently and, in some cases, synchronise with each other. In specifying manufacturing systems, this construct is very important. In our example, we will consider the following sub-systems : pallets P1, P2, P3 and P4 and stations A, B, M, O, S1, S2, S3, S4 and T. Hence, we will have, at the top level, thirteen parallel diagrams, each corresponding to one specific equipment. See Fig. 9.

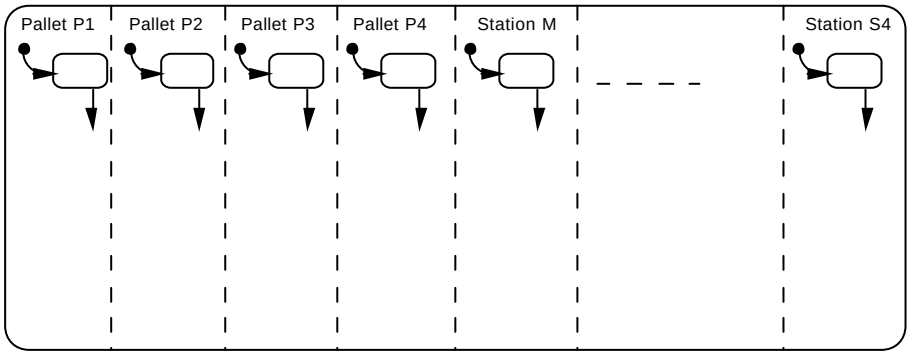

Fig. 9.

By refinement of the station B, Fig.10 shows how we can detail the station state. States of both MB_I and MB_{II} machines become visible.

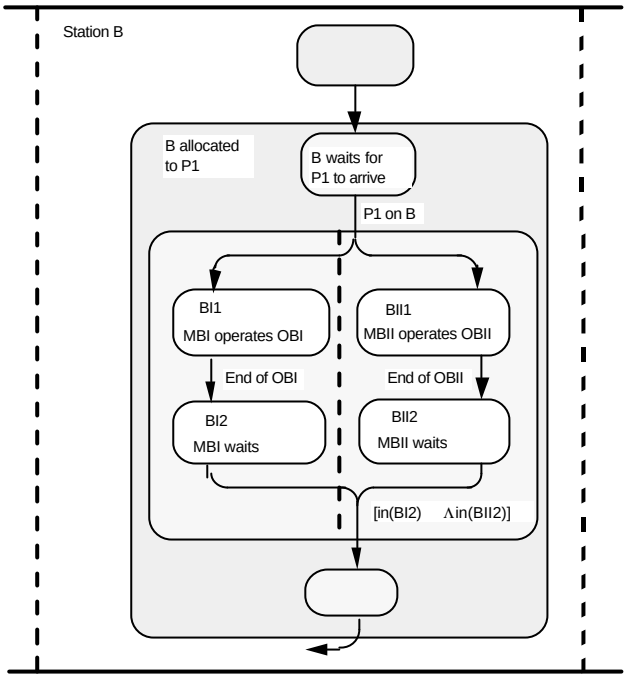

Fig. 10.

2.7 - Resource sharing specification

Among the sub-systems composing a manufacturing system, we have shared resources. So there must be a mechanism to allocate resources to the process users. We can specify this by Fig. 11.

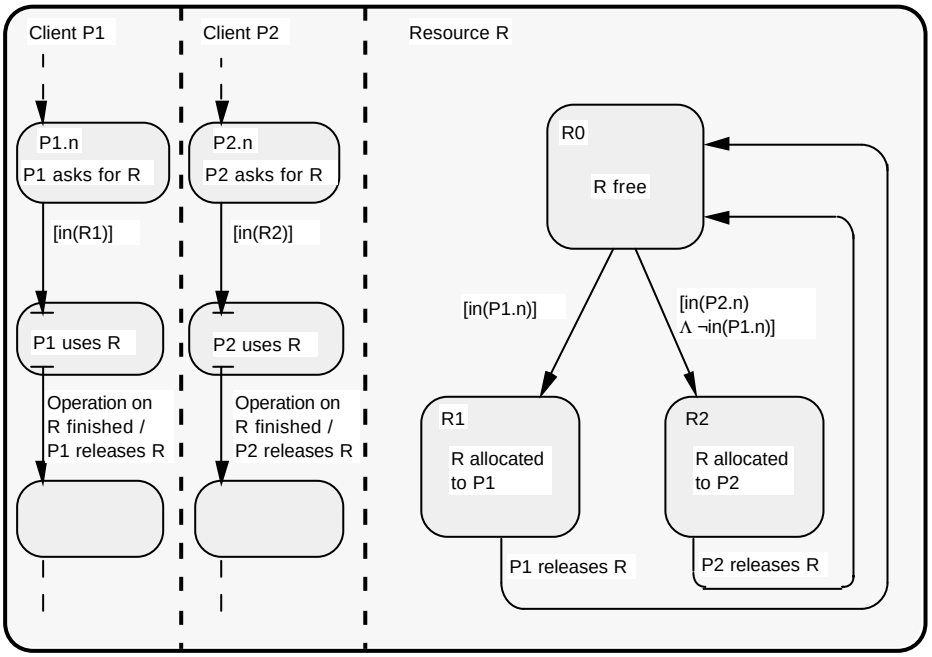

Fig. 11.

Interpretation :

- We consider that each user Pi and the resource R are described as sub-systems using concurrency ;
- Each user diagram contains a state, noted Pi.n, in which the user asks and waits for the resource R ;
- The resource diagram contains :

. a state, noted R0, which denotes that the resource is free and waits for an user ;

. as much states Ri as users ; Ri denotes the situation in which the resource is allocated to the client i. The transitions between R0 and all the Ri states expresses the resource sharing. The conditions supported by these transitions formalise the assignment logic. In Fig. 11, P1 has priority on P2.

Each user Pi has to release the resource when exploitation stops. The event "Pi release R" must be generated to produce the resource evolution to state "R free".

In our example, the four pallets compete, initially, for post O attribution. In order to overcome this problem, we have just to extend the Fig. 11 to the case of 4 users.

2.8 - Specification of the global system

Fig. 16 gives a possible specification of the cell control system. Note, that, graphically, this specification uses only the five basic constructs. The communication leans on the following items :

- events
  - . orders sent by the manager : start, stop
  - . reports sent by the process (end of operation) : End of OAα, End of OAβ,..., Choice α, Choice β, P1 on A..., P2 on A...
  - . generated events : P1 releases A...
- conditions
  - . simple conditions : [in (M1)], [pP1 == α]...
  - . compound conditions :

. [Ref] = [in(P1.0)Λin(P2.0)Λ...Λin(S4.0)] which evaluates the reference state situation
. [(P1?A)] = [in(P1.411)Vin(P1.414)] which evaluates if there is at least a state in which the pallet P1 asks and waits for the post A
Same with other pallets and resources

- data
  - . pP1 : type of part fixed on P1
  - . 1P1 : location that P1 occupies (value used for the carriage transfer control)
  - . dP1 : post destination for P1
  - . same for P2, P3 and P4

3 - Hierarchy and abstract specification

Hierarchy constructs in statecharts are well adapted mainly for specifying by abstraction. This top down specification approach is supported by two main constracts : state abstraction and event abstraction.

3.1 Interest of Hierarchy

To be useful, a state-event approach must be modular, hierarchical and well structured. So, one can intend a specification to be an abstraction process which ensures to represent the essential aspects of a system before giving more details. For instance, in the FMS domain, the combination of flow of all different products leads quickly to a great complexity of design. According to that fact, it seems that the easier way to carry out such a study consists, for a first step, in keeping a rough degree of precision. Next, we are able to detect sub-systems having a real influence on global performances. Those systems are analysed with more attention and finally take the designer to build the model of each ones in an inferior hierarchical level. This procedure can reoccur any time the specification wants to tell apart secondary states from ones to be refined again. Consequently, one must have a design tool which allows a hierarchical analysis to handle the case of complex workshops. These associated figures are shown in annex 2.

3.2 - Possibilities offered by Statecharts

3.2.1 - Introduction

The concept of Hierarchy enables abstraction and refinement possibilities and so improves readability of a specification. Those two notions take part in systems description in a complementary way :

- abstraction is characteristic of a "Bottom-Up" approach and permits to draw a state without taking account of its detailed composition ;
- refinement describes a "Top-Down" view and specifies the internal structure of a state.

The Statecharts tool furthers the use of these two description ways for complex reactive systems, and so, offers a sufficiently clear and rigorous means to valid the desired behaviour.

3.2.2 - The two types of decomposition

One of the major advantages of Statecharts in rapport of the classical state-transition diagrams is the ability to structure states to express conceptual grouping. We can represent those clusterings by using two types of state decomposition :

- an XOR decomposition : a macro state encapsulates a set of sub-states to indicate that the system is in only one of the sub-states while the system is in the macro-state.
- an AND decomposition : to ensure the specification of process that evolve in parallel, we use an AND macro-state. When it is active, all its components are active too.

The macro-state constructs allow a hierarchical representation and give the designer a possibility to simplify the structure of the designed system. In the sequel of this paragraph, we will see how this syntactical entities are handled and we will focus more specially on the links between the different levels of a specification. Then we will classify the transition to deal with to specify :

- the entry in a macro-state,
- the exit of a states grouping.

At first instance, no problem arises when the representation of one of those two points is tackled. Nevertheless, we distinguish in the following the constructs that respect an "inter-level partitioning" from the others. That distinction splits all possible transitions in two distinct sets "internal" transitions and "external" ones. This former transition qualification conveys the membership of an arrow to one hierarchical level of a specification. We are giving next the interest in using a graphical model based on "internal" transitions.

3.2.3 - The different sorts of entries

The Statecharts formalism has a flexible approach concerning the entrance into a state ; the possibilities offered are powerful constructs. They are :

- default entry,
- selective or conditional entry,
- historic and recursive historic entries.

In Fig. 13, all these entries are represented and serve as support example for each following paragraph.

-i- Entries keeping "inter-level partitioning"

Among all variant that Statecharts provide to perform the entrance into a state, only the default entry has an "internal" transition characteristic. Default entry shows the sub-state in which is going to when the system enters a macro-state. Such a mechanism illustrated in Fig. 13 when transition "a" is fired ; then the control evolves towards state "S2" which is the default entry state. In addition to that, this entry constitutes the only way to design the configuration of active states at the beginning of execution for the described system.

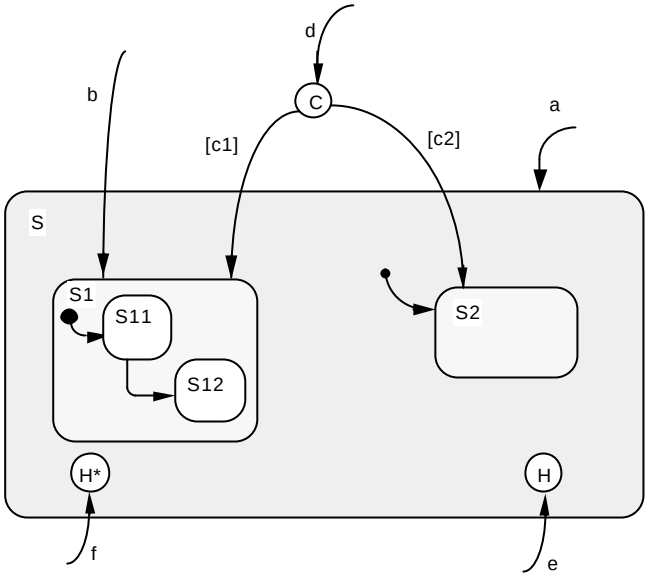

Fig. 13.

- *ii-Entries that do not respect "inter-level partitioning"*

In general, the Statecharts formalism does not make any restriction relevant to states that are goals and sources of a transition arrow. Then nothing keeps us using transitions that we qualify as "external" ones (source and goal states of an arrow belong to different hierarchical levels). In fact, those transitions represent the simple method to specify a behaviour as accurately as possible. So all diverse alternatives to enter a sub-state are proposed now :

- the entrance to a specific state : it is a particular case of the global definition to enable a transition to be fined whatever its source state is and wherever its goal is located. See "b" labelled transition in Fig. 13.

- selective or conditional entries : the purpose of those two entries attempt to reduce the number of arcs. Those entries are specified by two connectors denoted respectively by S and C. The selective entry is devoted to different events that activate different sub-states inside one macro-state. The conditional entry addresses cases where one event "e" generates the entrance in one sub-state between more other ones of a macro-state, according to different conditions. It is then possible to replace all arcs labelled by "e [ci]" with one event "e" that points to a conditional entry. See "d" labelled transition in Fig. 13.

- historic entries : Statecharts is the only known formalism dealing with historic construct which is probably the most powerful mechanism of the language. It ensures to return in the last deactivated sub-states that the system leaves when the macro-state is exited. There are two types of historic entries, H and H*, that differ depending on the ability to re-enter all the left states. See transitions labelled by "e" and "f" in Fig. 13. For the former historic entry the state which the system goes back belongs to the level where the connector H is drawn. For the latter historic entry, also called recursive entry, the system reverts all the last exited states whatever the level is.

- *iii- Remark*

This classification based on respect of the "inter-level partitioning" does not affect the mixed use of the different types of entries we have presented. The Fig. 17 actually shows that fact.

3.2.4 - The different types of exit

Statecharts do not have as many mechanisms to exit a macro-state as there are different constructs to enter a refined state. The sole solution to leave a macro-state consists in processing a transition whose source is a sub-state or the state itself as shown in Fig. 14. The classification we are trying to make is in that sense depending on the representation.

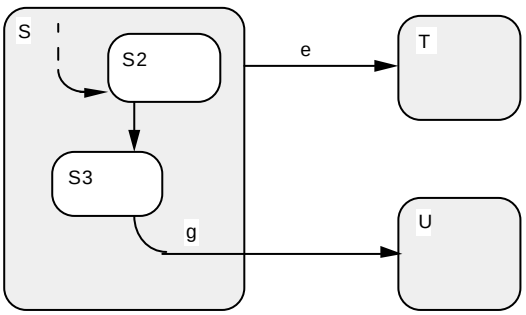

Fig. 14.

- *Exits keeping "inter-level partitioning"*

To let a transition that exits a macro-state be "internal", one must avoid specifying it by an inter-level arrow. The issue is in drawing an arc whose origin is taken on the edge of the macro-state we consider. That transition realises then an abstraction of transition [DeRo-91] since its firing causes the system to evolve forwards the state pointed, whatever the active sub-state is. A representation of this situation is given in Fig. 14 by the "e" labelled transition.

- *Exits that do not respect "inter-level partitioning"*

It is the solution we use the more frequently because it makes visual the specific sub-state in which the system has to reside to go out the macro-state. See Fig. 14 the "g" transition. In the same way, when several transitions enable the system to exit one refined state from different sub-states, the representation of inter-level arrows expresses the means the more natural and the easier way to describe that kind of behaviour.

3.2.5 Interest of the offered possibilities

Abstraction and refinement constructs can be seen as solution to make a representation simpler. In domain of AMS control, abstraction can turn out to be an practical issue when one wants to address interruption problems.

This case is encountered in AMS field by someone who needs a way to specify that the control must react in a safe manner as soon as a behaviour able to alter a component performance is detected. Then, within the scope of exception treatment, we can equip each machine with a watch-over device to inform the control system about a failure occurrence. The receipt of such an event starts afterwards a procedure of errors recovery in any state the controlled activity is. Moreover, it is that behaviour we have taken up to illustrate this kind of situations.

This part of specification describes the behaviour of the Station "A" when happens a problem while a part mounted on pallet "P1" is machined. It appears clearly that only one transition has its source on macro-state "A allocated to P1" and that points to the "Diagnosis" state. So taking that transition produces :

- the control to enter the "Diagnosis" state and begin error handling,
- the exit of macro-state "A affected to P1" and also of the last refining state in which the system has been resident.

One of the goals of a specification in the production field consists in managing the different running modes. To complete the error detection aspect, the specification of the different steps that lead to resumption in normal running mode sets out an interesting problem.

We do not detail here the method we should use to treat the "Diagnosis - Recovery" cycle. In fact, for such situations, we will consider that the operator is called to solve the failure and to put the controlled process back into the current state it resided before the exception event occurred.

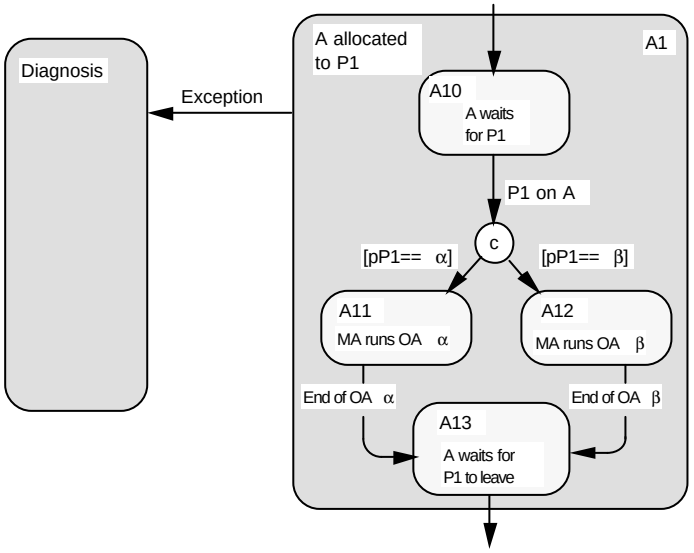

Fig. 15.

Here, the historic connector allows us to perform easily a specification of the system behaviour that we intend to be, once the breakdown is resolved. The Statechart of Fig. 16 shows us an example which adds this resumption handling to the situation presented in Fig. 15.

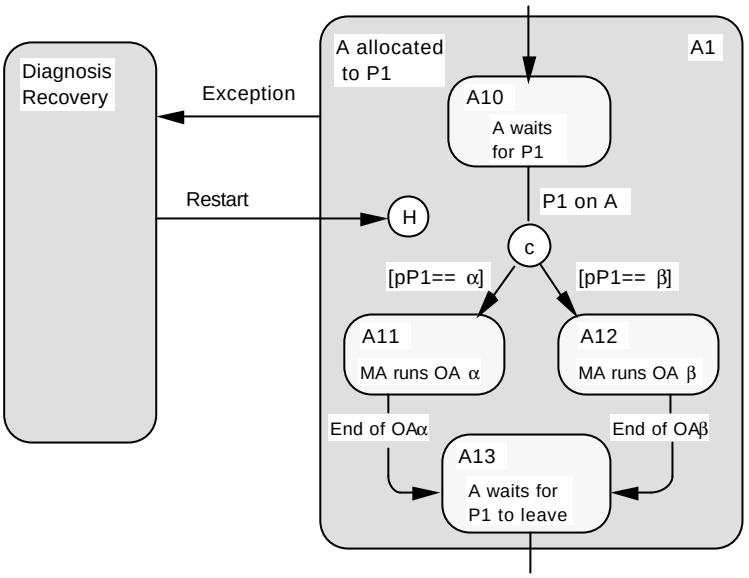

Fig. 16.

Thus, when the operator has solved the problem on the post "A", he orders the control to restart. This event causes the model to evolve in the following way :

- state "Diagnosis - Recovery" is left,
- the historic entry makes activated macro-state "A affected to P1" and the last sub-state too that the system has exited.

Suppose, now, we would like to extend this work of supervision on the entire station "A" to detect an error whatever the post occupation is. Then the scope of an exception event must increase such that on its occurrence the state "A" is left. Hence one needs a construct based on the recursive historic entry to express that the control must return in the last exited state wherever level this state is located.

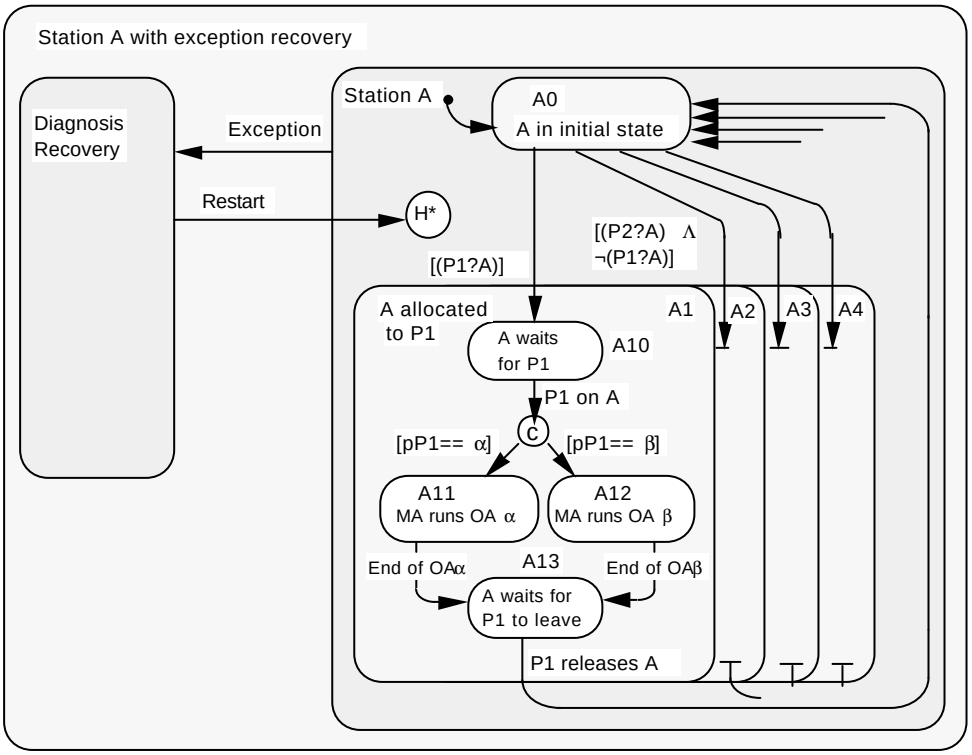

Fig. 17.

3.3 Rules to attenuate complexity

3.3.1 - Introduction

To lighten a state-transition diagram and hence to improve its readability has constituted the object of the study of one sets of Statecharts constructs dealing with hierarchical decomposition. But yet conciseness is not the only aspect of a "successful" design. Flexible manufacturing workshops reach some important degrees of complexity that one would have great difficulty representing the whole control system on a unique sheet of paper or even on a same physical support. It is then necessary to have a method that permits to split the graphical description without breaking its global understanding.

To solve that problem, an extension., issued from the Statecharts one, has been developed in order to suggest a conception way by descending analysis. The described system is so decomposed within a behavioural point of view. It consists in structuring each studied set from a functioning analysis to isolate elementary behaviours which are translated by a state-transition diagram. But the expression of the controlled system working leads, in using Statecharts, to introduce inter-level transitions that refute the graphical decomposition aspect of the control into communicating sub-systems.

Therefore the notion of inter-level transition is essential to specify all the systems we are interested. To elude that constraint while keeping the Statecharts tool, we have to define representation rules to give a more directive method to design automated systems of fabrication. Those rules concern :

- the entrance into a macro-state to offer more possibilities than the default entry which is, as we have seen, the only means to enter a macro-state with respect to the "inter-level partitioning".
- the exit of a refined state (achieved evolution for the sub-system) to remove the ambiguity bound to the representation of transition, abstraction of transition.

In fact, the solution to reach this goal realises the limitation of entries and exits in a macro-state to the ones that provide respect to the "inter-level partitioning". But in that case, the expressive power of Statecharts becomes then greatly decreased. And that's why we present now constructs that allow a designer to specify any situation without using inter-level transitions. In the sequel, all the graphical constructs will not be extracted from our example.

3.3.2 - Structures of substitution for entries with non "inter-level partitioning"

When we should like to ensure the graphical representation to be the result of an approach by descending analysis, we must adopt a specific procedure. We propose a compositional mode to divide a global transition into "internal" ones. This design way has then to offer the possibility to part the information each level needs. We use the broadcasting property of Statecharts to solve that problem and have, in final, the guarantee we have well modelled the expected behaviour. We now give case by case an illustration of the graphical constructs we recommend :

- the entrance into a specific state : to split an inter-level arrow, when it is alone to enter a refined state, the default entry is convenient as shown on Fig. 18a. This solution stays sufficient if several "external" transitions point on the same sub-state, Fig. 18b. However, when two transitions at least end in different states of the sub-system, a selective or conditional entry is necessary.

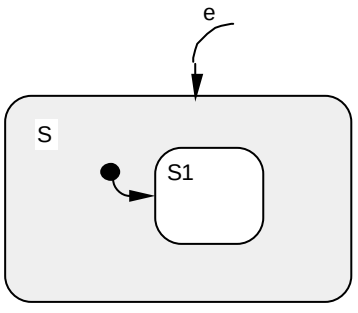

Fig. 18a.

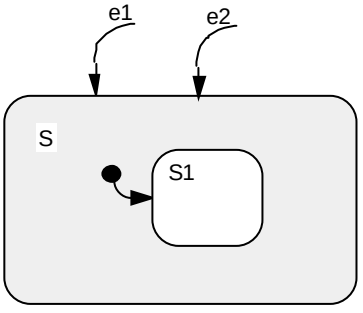

Fig. 18 b.

- the selective or conditional entry : these two graphical constructs are helpful when inter-level transitions reach different states inside one refined state. In that case, the representation lets appear on the macro-state level two arcs pointing on the edge of this state. We give then the ability to that state to be activated by two distinct transitions. But inside this macro-state, the entrance depends actually on the fired transition. So it is important to generate that information to see the system react as expected. The Statecharts communication comes here to ensure the passage of information which is either realised by a generated event in case of a selective entry (Fig. 19a), or either by modifying a variable in case of a conditional entry (Fig. 19b).

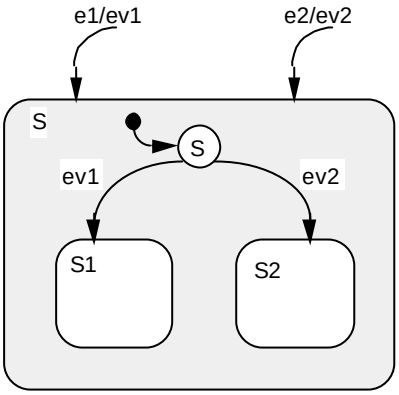

Fig.19a.

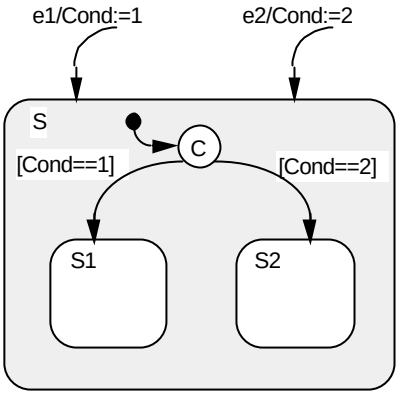

Fig. 19b.

- the historic entries : their definition involve the use of inter-level arrow. Nevertheless, a way is presented in (Harel, D. 1987a) based on the default entry to support the same meaning without "cutting the border" of a macro-state. See Fig. 20.

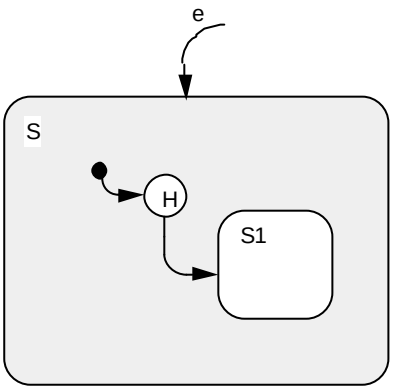

Fig. 20.

In a more general purpose, the Fig. 21 takes up the example of Fig. 13 and makes use of all developed aspects to satisfy the concepts of structured analysis.

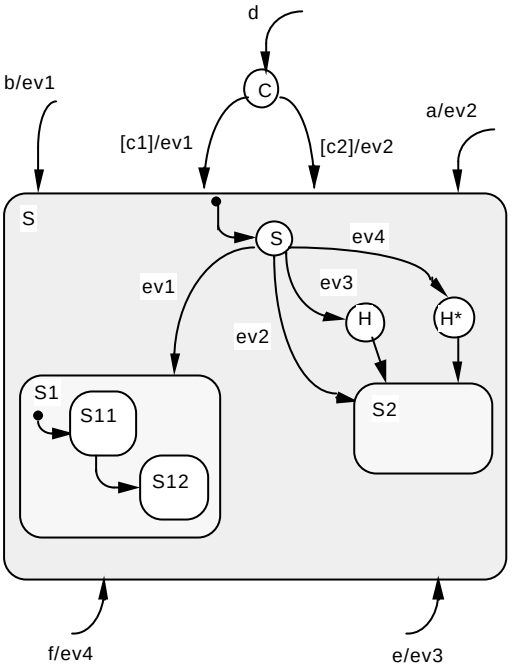

Fig. 21a.

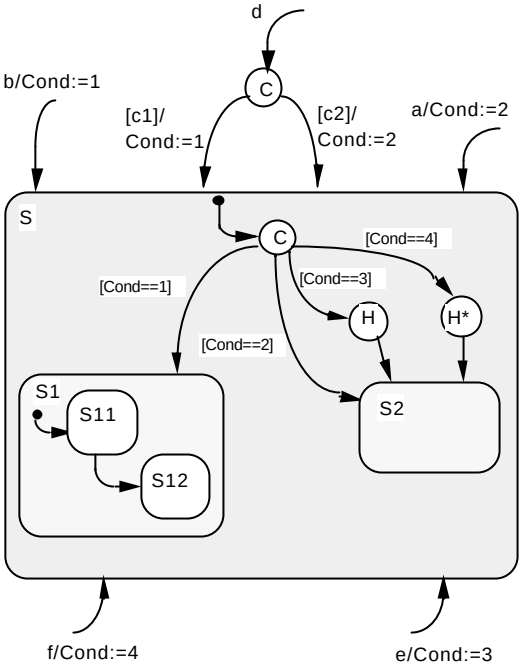

Fig. 21b.

3.3.3 - Structures of substitution for exits with non "inter-level partitioning"

To treat that aspect we can distinguish two variants of solution according to the broadcasting we dispose with Statecharts formalism. So there are :

- constructs using conditions : the idea to part an inter-level transition consists in saying that an assert expressible on the macro-state level can be found and model that way the end of the behaviour described by the set of refining states (Fig. 22a).

- constructs based on generated events : in that case, the designer knows the diverse possibilities of termination for each refined state and put them in place on the superior level. The sub-system takes charge to advert that it ends its treatment by sending an event allowing to take one of the defined transitions of the macro-state (Fig. 22b).

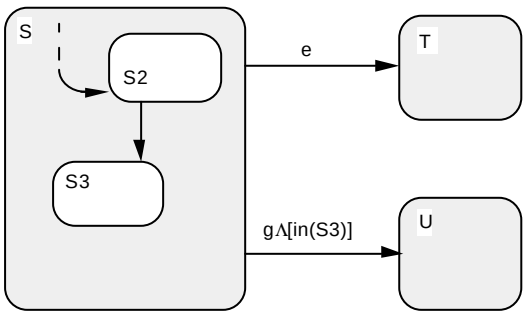

Fig. 22a.

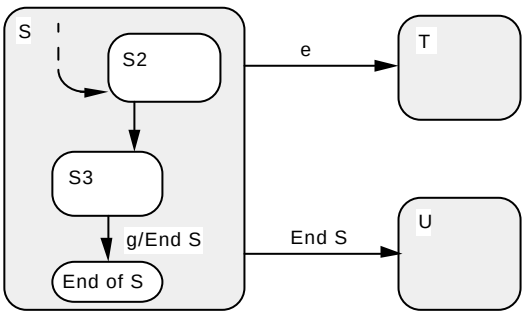

Fig. 22b.

3.3.4 - Application to the cell

For such a system presented in section II, using the rules we have proposed leads to a specification which remains very close to the representation of Fig. 12 and brings to the fore modularity plus structural aspects.

Fig. in annex 3 gives an outline of some elements of that specification to illustrate how one can handle a hierarchical decomposition approach. Suppressing inter-level transitions enables to study each part of the model in a more autonomous and homogeneous way.

4. Generic specification

Several advanced features were proposed to the basic formalism but for most the syntax and semantics have not been formally established. Among all those adds, one mechanism seems to be particularly well adapted to problems we would like to address here ; these are parameterized states.

By the standard and reusable nature of entities we have to control, it is interesting to make appear generic behaviour. The expression of those behaviours with Statecharts is made easier by using parameterized states that enable to support bindings between the description of changes and a parameter which stands for an instance of the class on which such an evolution is applicable.

We can distinguish for that construct means two variants according to the operator of hierarchical decomposition AND or XOR which is employed.

4.1 - Generic parallel processes

On the global system representation (annex 2), we can remark that the specification of pallets P2, P3 and P4 only makes reference to the P1's one. The explanation to this fact is bound to the role pallets have inside an automated manufacturing system. The reserved purpose of those entities concerns the parts handling. Moreover, the four pallets are identical to guarantee that they can be accepted on the different posts, and, also to allow the operator to fix a part according to his choice. Then all the part-pallet combinations are authorised while the production takes place. Consequently the state-transition diagram of pallets must be identical.

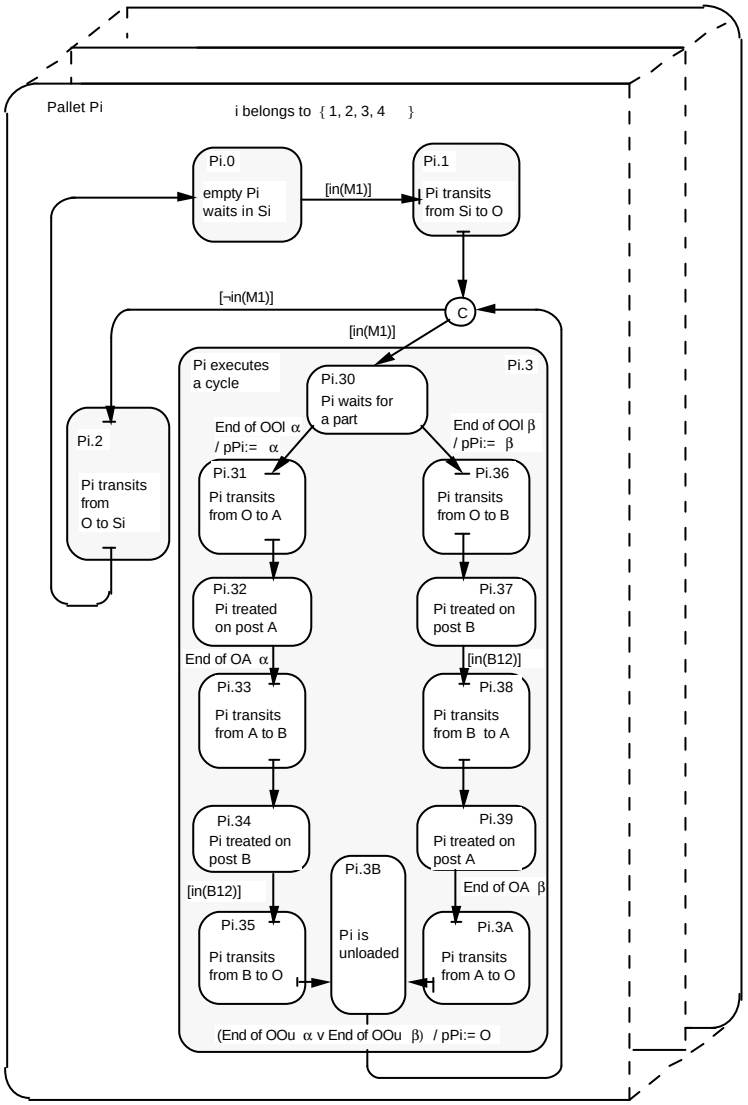

Fig. 23

On the other hand, the instances P1 to P4 of pallets are individual entities and on this account evolve concurrently to describe the process plan of the part they transport. In that sense, the informal manner we used to build Fig. 16 can find out a more suitable solution if we resort to the parameterized-and states. The representation is obviously simplified and shows the desired behaviour of the set of pallet as we can see on Fig. 23.

4.2 - Generic sequential states

Whereas we brought to the fore generic behaviours for entities evolving in parallel, the manufacturing systems make use of choices that lend themselves well to parameterization. For our example, when we are interested in what happens on a station, we have to locate the control between two states, free or allocated, and, in the last case make visual for which pallet the post is working for. Once a pallet-machine relation is effective, the same cycle is always realised :

- the station is loaded,
- a part α or β is machined,
- the station is unloaded.

To represent graphically the station A functioning, the state "A allocated to P1" has been duplicated as many times as there are pallets. Here the use of parameterized-or states with "Pi" as parameter spares us an arrows and states multiplication as we can observe there-after :

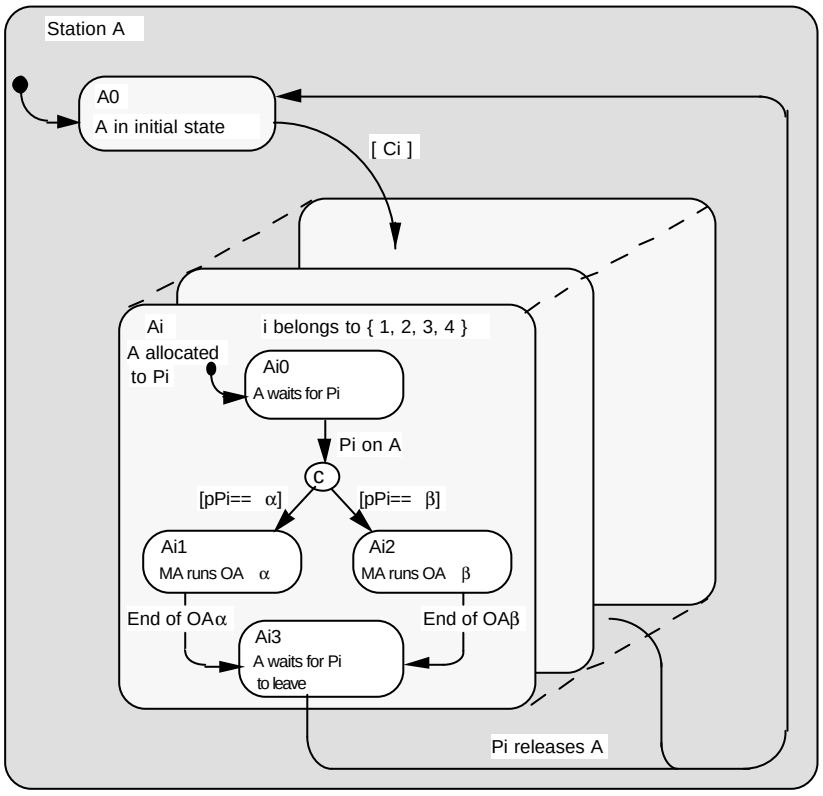

Fig. 24.

Comment : the graphics is lightened and the representation behaves unchanged when the number of pallets is modified. On the other hand this generic aspect needs the condition to be corrected to cover all the situations.

$$C1 = (P1?A) \quad \text{and} \quad \forall\ i\ \varepsilon\ \{ 2, 3, 4 \} \qquad Ci = (Pi?A)\ \{\ \bigwedge_{x=1}^{i-1} \neg(Px?A)\ \}$$

which stands for i = 3 :

$$C3 = (P3?A) \wedge \neg(P1?A) \wedge \neg(P2?A).$$

5 - Extension to information systems modelling

5.1 - Functional view of reactive systems in software engineering framework

Five basic elements may be useful to reactive systems description :

- data (system definition),
- event (system change),
- data transformation (function),
- event transformation,
- time (related to all four aspects).

Each method focuses generally on some specific elements. For instance, Petri Net based methods, which are very adequate for discrete event systems, take into account event and event transformation. SA and SADT methods are more oriented to data transformation through DFD's and actigrams respectively. SA-RT is one of the first approach to deal with all aspects in reactive systems specification and analysis.

The extensions made on Statecharts language to be used as a tool in software engineering are supported by the CASE tool named "STATEMATE" *(Harel,D.1990, iLogix. 1991)*. This working environment provides some additional languages and suggests an approach by functional decomposition ; in fact, the main principles are near equivalent to those of SA-RT methods . A critical study of both tools has been carried out *(Sahraoui,A. and Ould-Kaddour,N. 1993)*. In this part, we present a brief introduction to Statemate mechanisms.

Statemate consists of three graphical languages, each for modelling a view of the system to be designed :

- Statecharts for behaviour view : to specify discrete event features of the systems ;
- Activity-charts for functional view ;
- Module-charts for structural view.

In our work, we will use only Statecharts and Activity-charts with a background experience on SA-RT method. In this respect, we give the main analogies :

- Activity-charts in Statemate equivalent to DFD in SA-RT,
- Statecharts equivalent to CFD in SA-RT.

At the top level, the system can be seen as a single activity ; this root-activity RA is described by an activity-chart consisting of a control part, the statechart sc_RA, and some sub-activities SA1, SA2, SA3 as shown in Fig. 25.

Each sub-activity SAi may be detailed by an activity-chart as RA, introducing its own statechart sc_SAi managing its own sub-activities SAij, and so on. Each statechart may be decomposed as any macro-state ; this has been discussed in the previous parts.

Statemate provides specific events, actions and conditions corresponding to activity-charts ; these primitives used in Statecharts specification enable to supervise the execution of activities. See table 1. These primitives behave as the ones used in real-time executives except message passing primitives which are as data flow between activities.

REFERENCES	EVENTS	CONDITIONS	ACTIONS
state : S	entered(S), exited(S)	in(S)	
activity : A	started(A) stopped(A)	active(A) hanging(A)	start(A), stop(A) suspend(A) resume(A)
data : D, F	read(D), written(D) true(c), false(c)	D=F,D<F,D>F...	D:=exp m_true(c), m_false(c)
time	timeout(e,n)		schedule(a,n)

Table 1.

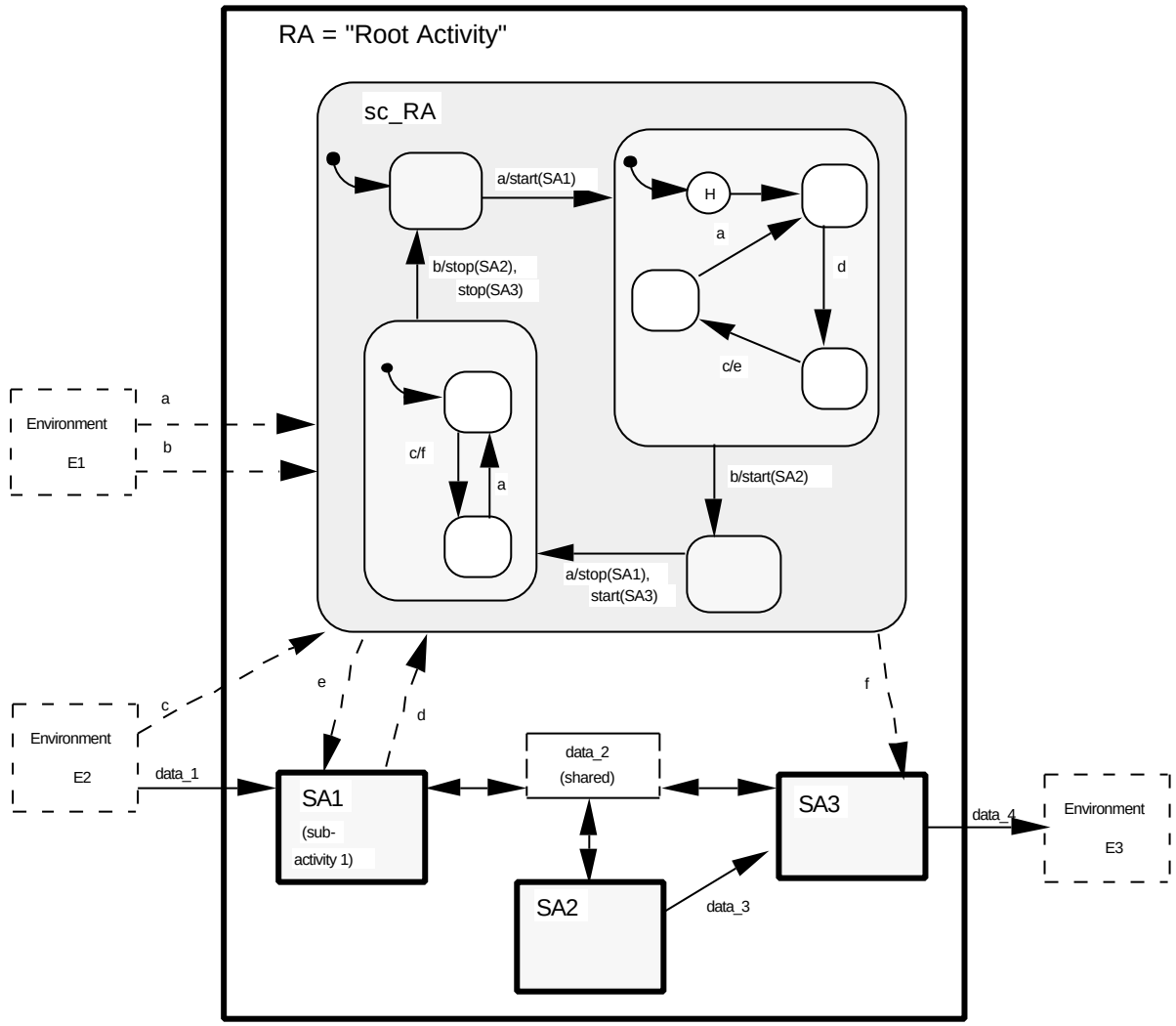

Fig. 25

While an activity must be explicitly initiated by a start or a resume order, it can stop under several conditions :

- the parent activity is stopped ; i.e. if SA1 is stopped, then all active sub-activities SA1j (j=1, ..,n) are stopped ;
- a stop(activity) action is executed in one of its master statechart transitions ;
- if the activity terminates by itself (transformation activity) : these types of activities can be an optimisation program, calculus...

We propose now to tackle an extracted situation of our AMS application using both Statecharts and Activity-charts. The purpose consists in making visual the functional aspects whose are implicitly intended in using Statecharts only. In that sense, Statemate languages are complementary. We take as example how to extend our specification with Activity-charts by looking at the sub-activity SA1 = "Station A treats pallet P1". See Fig. 26.

5.2 - Towards an object specification with Statecharts

It is well proven that an object oriented development is the most interesting way *(Booch,G. 1986)* in terms :

- natural development,
- ease and hence fast development and prototyping,
- interface with actual software environment.

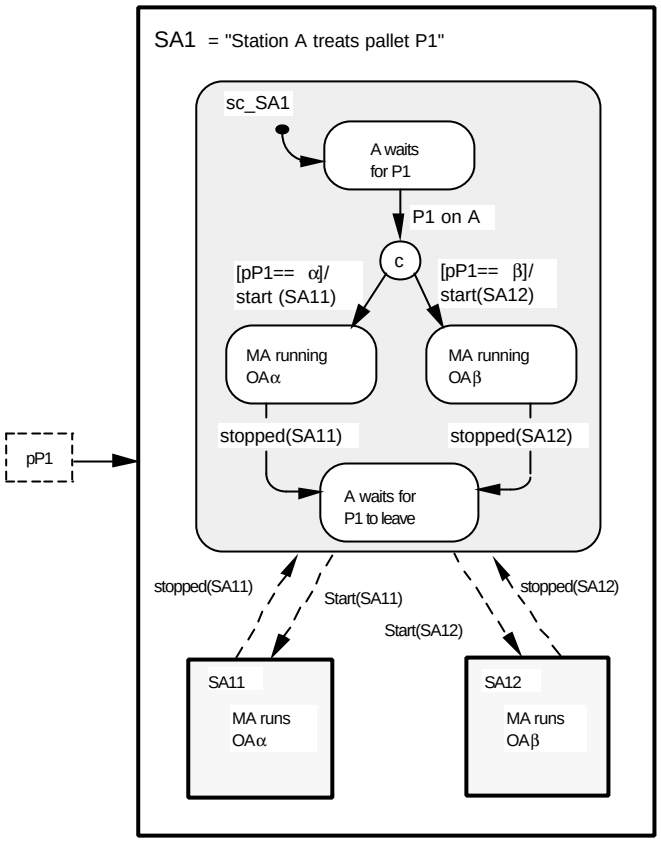

Fig. 26.

Our goal, here, consists in presenting just the approach since the whole method is not yet fixed up. We want this actual approach to be part of a methodology, YASMINA, nowadays being developed in order to offer a complete method and enhance the development of reactive systems by using Statecharts and some other extensions.

5.2.1 - The object paradigm with Statecharts and Activity-charts

We prone, initially, the Booch approach [Boo-86] to apply the object paradigm with Statecharts. Let resume synthetically the Booch approach :

- identify objects,
- identify attributes of objects,
- identify operations of objects,
- identify interfaces between objects,
- implement objects.

This approach has been catered for design issues. However, in our approach, we do mapping of objects into particular activity-charts we could name "object-charts". So we make use of the basic proposal presented by Booch to adapt itself to specification with Statecharts and Activity-charts. We remind that Booch introduced in his method some diagrams in order to fit Ada syntactic entities.

Being just an approach, we give some preliminary rules and guidelines. One root object RO is identified with the top object-chart consisting only of some main sub-objects MOi. Compared with the above functional approach, the top activity-chart has a statechart to control sub-activities. Only messages are exchanged between MOi, and this determines objects interfaces. But an assumption must be made : communication signals are not broadcasted as in Statecharts handling, they are rather emitted towards the interested objects.

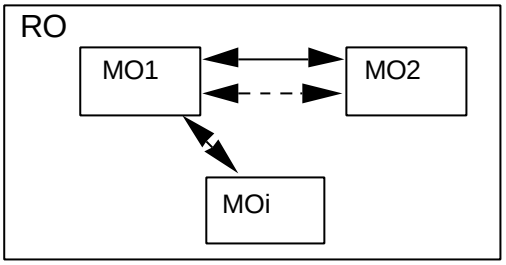

Fig. 27.

Each object can be refined into sub-objects on the one hand, and into sub-activities and a control chart on the other hand (except for the root object which is built solely on its MOi). Consequently, the general form of an object is constituted by a box encapsulating both sub-activities (operations) which are managed by a statechart and sub-objects. In addition to this, it is worth noting that all objects must be initially active since they are self-controlled (no statechart is linked to them).

The main differences with the structured development in a functional specification are :

- no statechart at the top since all objects are identified,
- the root object is decomposed into main objects,
- each object consists in sub-objects, a statechart and operations. Operations are controlled by the statechart. A sub-object is necessarily decomposed until it concerns only operations,
- data and control signals are flowing from objects to objects, they constitute the messages contents,
- an activity-chart is required to describe a basic sub-object (at the bottom level of decomposition). The approach becomes mixed ; this is being encountered in many cases because a functional view is more adequate at this level..

Let see how we can apply the above approach.

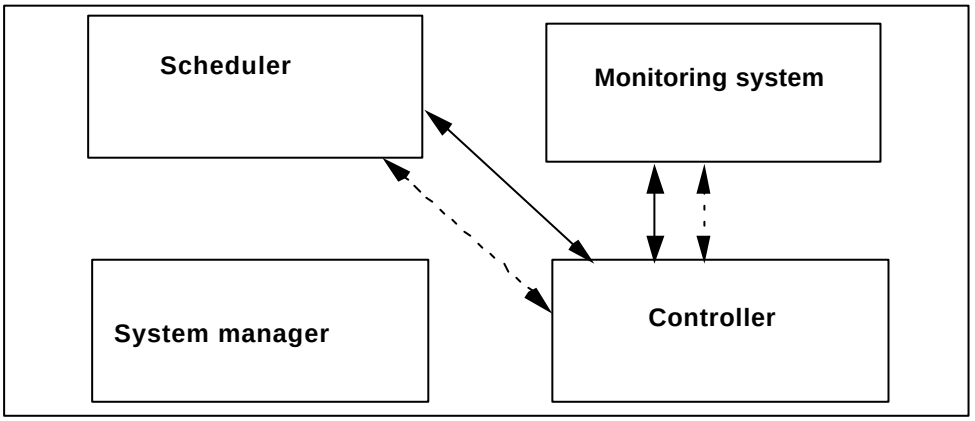

Fig. 28

We refine the object controller into sub-objects, operations and a statechart ; this will be carried out with the proposed case study.

5.2.2 - Application revisited with object specification

We detail the object specification presented in past section and applying it at the lower level (cell level) ; this being the system considered in our case study.

The specification done using the Statecharts approach will be slightly modified but the object decomposition was already done initially. In order to distinguish activities from sub-objects, we draw rectangles with single solid line to point out sub-objects, and those with double solid lines describe operations of the object.
We chose the identification of the objects by simply doing a mapping from the physical layout of the system into an object model. Hence all stations controllers are objects with their own operations activities and also sub-objects

corresponding to the physical entities embedded in the corresponding station. We illustrate this by the following specification (sync_signals are exchanged between sub-objects).

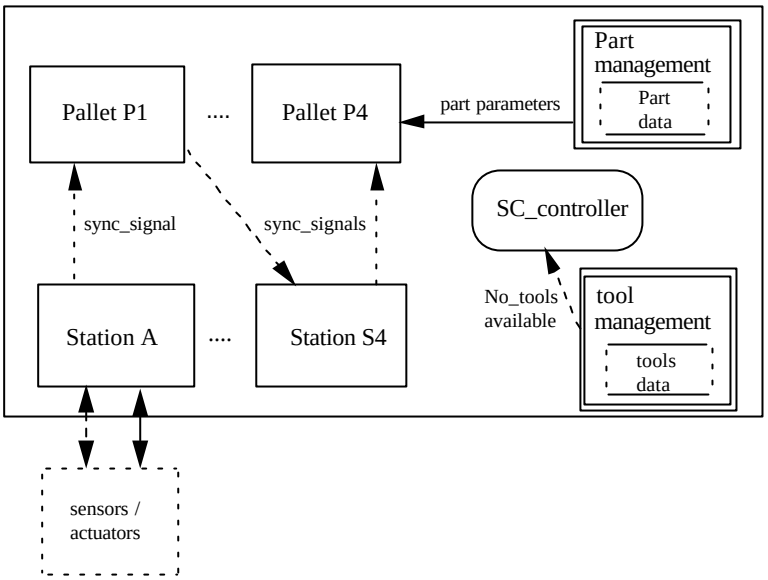

Fig. 29.

Part management and *tool management* are operations of the *controller* object. They provide all data flow handling at this level ; this is just a preliminary approach ; one also could asks why not consider all sub-activities as sub-objects, some arguments have been given above. *Part management* and *tool management* are not considered as sub-objects and cannot be decomposed unless in the structured functional method.
The statecharts activates these operations and also can receive signals from neighbouring statecharts embedded in the scheduler or monitoring manager. A multiformalisms statecharts based approach has been developped for avionics systems *(Sahraoui,A et al, 1996)*

In summary, a preliminary approach has been given to apply object paradigm using the activity and statecharts method. Further work is required to overcome the other aspects as classes and inheritance. The classes aspects can be implemented by using the parameterized states in statecharts (discussed in section 4.1) ; but there is no construct in Activity-charts for generic specification. Extensions need to be developed but a common agreement should be found in order to avoid scattered extensions that may create confusion among users as it has been the case for some other models in the past.

CONCLUSION

This work was mainly focused on the application of Statecharts to specify the control of automated manufacturing systems. An introduction to the language has been presented. Most of the offered constructs has been given and implemented ; the expressiveness was illustrated in many situations. Some possible extensions has been presented in software engineering fields. Indeed, this paper illustrates that Statecharts is a suitable language for specification of automated manufacturing systems and, probably, the most appropriate in terms of balance between formal and semi-formal specification.

Acknowledgements : We are indebted to all colleagues from LGP laboratory at ENITarbes and LAAS-CNRS for discussions and comments on the present work and to the anonymous referees for their suggestions and comments on the earlier versions of the paper.

References

(Benveniste, A. and Le Guernic,P 1990). Hybrid dynamical systems theory and the SIGNAL language. *IEEE Transactions on Automatic Control, vol. 35, No 5, pp. 535-546.*

(Berry, G. P. Couronne,P. and Gonthier,G. 1987) : Synchronous programming of reactive systems : an introduction to ESTEREL. *INRIA report No647, March*

(Booch,G. 1986) Object oriented development. *IEEE Trans.Soft.Eng, Vol 12, No 2, Feb 86,pp211-221.*

(DeMarco,T. 1979) T. De Marco : Structured Analysis and System Specification. *Prentice-Hall.*

(De Roever, C. Huizing 1991) Design choices in Statecharts. *Information processing, Vol 37,No 4, fev 91, pp205-214*

(Harel,D. 1987) Statecharts, a formalism for specifying complex systems, *Journal of science of computer programming, Vol. 8, No 1, 1987, pp 231-274.*

[Har-87b] D. Harel, A. Pnueli, J.P. Schmidt, R. Sherman : On the formal semantics of Statecharts,*IEEE LICS, Logic in computer science, 1987, pp 54-64.*

(Harel,D. 1990) Statemate, a working environment for the development of complex reactive systems. *IEEE Trans. Soft.Eng., Vol 16, No 4, April , pp 403-413.*

[Hatley ,D. and . Pirbhai,J. 1987) Strategies for real-time specification, *Dorset House Publishing*.

[iLogix 1991) Statemate, *Burlington, Mass.*

(Krogh,B and Wilson 1990): Petri Nets for specifying discrete event controllers.IEEE Trans. of Soft. Eng,Vol 16, No 1.

(Pnueli,A. and Harel, D. : Applications of temporal logic to the specification of real time systems. *Workshop on fault tolerant real-time systems, Warwick, UK, Sept 88.*

(Pnueli, A. and. Manna,Z 1991) : The temporal logic of reactive and concurrent systems. *Springer-Verlag, 1991.*

(Sahraoui, A. and . Gilhodes,L. 1991) : Petri nets, temporal logic and Statecharts : a comparative study for specifying discrete event controllers, *European control conference ECC-91, Grenoble, 1991, Edi Hermes.*

(Sahraoui,A. 1992) An approach for monitoring discrete event systems, *AFCET APII (Automatique, Productique et Informatique Industrielle) Journal, Edi Dunod, Vol 26, 1992,No.2, pp 91-106*

(Sahraoui, A. and Ould-Kaddour, 1993) : A critical study on SA-RT and Statecharts methods to specify reactive systems. *IFAC congress , real-time symposium, Sidney, Australia, July*

(Sahraoui,A. et al, 1996) Co-specification for co-design in the development of avionics systems; *Control engineering practice, Pergamon Press, Vol .4,No6, p871-877.*

(Valette,R. 1983) : A Petri Net programmable logic controller. *Computer application in production engineering, CAPE'83.*

(Ward, P andMellor, S.J) : Structured Development for Real-Times systems. *Yourdon Press, 1987.*

(Ward,P 1986) The transformation schema : Control and timing. *IEEE Trans Soft.Eng,Vol-12, No2, Feb 86,pp198-210.*