A new framework for mode switching in SCT

Gregory FARAUT, Laurent PIÉTRAC, Eric NIEL
Laboratoire Ampère, INSA-Lyon
Bât. St-Exupery, 20 av Albert Einstein
69621 Villeurbanne, France
First.Lastname@insa-lyon.fr

Abstract— **An usual way in industry to design discrete events system (DES) consists of using a multi-modal approach to decompose the complexity of processes and specifications. The supervisory control theory (SCT) allows to prove that the process can be controllable to fulfill the requirements. Nevertheless, even based on a simple specification about commutations, it is very difficult to prove that the connections among modes are correct and reliable. This paper presents a framework allowing to design a system and detect specification incompatibilities by using the Supervisory Control Theory. The presented framework detects the specifications to change or to control them to promote correct mode switching.**

I. INTRODUCTION

Initiated by Ramadge and Wonham, the Supervisory Control Theory (SCT) has allowed to introduce some important properties such safety, liveness, controllability, observability and more recently diagnosability in Discrete Events System (DES) area. Supported by Finite State Machine (FSM), SCT has a problem of scalability and interpretation of models when is applied to industrial applications. Several approaches have been proposed to solve scalability: statecharts [1], modecharts [2], hierarchical finite state machines and mode automata [3], [4]. However, these approaches handle always the whole process and the whole specification, involving a difficult interpretation of models, in particular about commutation between system's modes that are not clearly identified and defined.

In DES, numerous works are focused on multi-modal control law. However, most of them applied compositional formalisms on modeling configurations : for instance state charts [1], mode charts [2], hierarchical finite state machines and mode automata [3]. In these approaches, the model of the process is not specifically representative of the state of the process, and thus, is unable to detect automatically (using the synthesis or validation) the issues about mode switching. Formal approaches like SCT [5] seems to be more convenient for mode management by distinguishing process and specifications.

Nevertheless, Few approaches using SCT with modal point of view exist [6], [7]. These works allow to study the intramodal behavior of each mode independently and identify the incompatible states when a commutation could happen. However these works depend on particular specifications required by the system. This leads to a lack of possible commutations.

Based on these previous works, this paper presents a new framework taking all possible commutations into account, whatever the specifications. This framework thus ensures a complete reachability, the major property concerning commutation phenomena, between system's modes.

This article is decomposed as following. Section 2 introduces Supervisory Control Theory (SCT), on which this work is formally based. We explicitly define the SCT approach. Section 3 explains, throughout the article, the new framework to obtain the final command law of a system. We conclude on actual works and some perspectives.

II. SUPERVISORY CONTROL THEORY

Ramadge and Wonham's theory [5] underpins the study of Discrete Event System control. This theory is based on separation between the model representing what the system **can do** (the process), the model of what the system **must or must not do** (the liveness and safety properties of the process), the model of what the system **does** (the controlled process) and the model of what the system **should do** (the desired language).

The process [8] G is a generator $G = (Q, \Sigma, \delta, q_0, Q_m)$ where Q is the finite state set, Σ the finite alphabet of symbols (event labels), $\delta : Q\times\Sigma \to Q$ is the partial transition function, q_0 is the initial state and $Q_m \subseteq Q$ is the set of marked states. States exist for periods of time (duration), whereas events occur instantaneously, asynchronously and at virtually random (unpredictable) times. For a machine, examples of states are "idle", "operating", "broken down", "under repair". Examples of events are "machine starts to work", "breaks down", "completes work" or "start to repair".

For the alphabet Σ, we have the partition $\Sigma = \Sigma_c \cup \Sigma_{uc}$ where the disjoint subsets Σ_c and Σ_{uc} comprise controllable and uncontrollable events respectively. We can also have the partition $\Sigma = \Sigma_o \cup \Sigma_{uo}$ where the disjoint subsets Σ_o and Σ_{uo} comprise observable and unobservable events respectively. No particular relation is assumed to exist between Σ_c and Σ_o: in particular, an event can be controllable but unobservable.

The languages associated with G are the closed behaviour $L(G) = \{s \in \Sigma^* | \delta(q_0, s)!\}$ and the marked behaviour $L_m(G) = \{s \in \Sigma^* \mid \delta(q_0, s) \in Q_m\}$. $L(G)$ represents the set of all possible trajectories, i.e. all possible system behaviours, whereas $L_m(G)$ represents the subset of trajectories leading to a marked state. These states allow us to model ends of tasks, states to be reached or states in which the system can be stopped.

For a process G, there are two types of conventional problems. In the first one, we have a process G and a specification E that is modelled by an automaton $E = (X, \Sigma, \xi, x_0, X_m)$ where X is the finite state set, Σ the same alphabet of symbols as in G, $\xi : X \times \Sigma \rightarrow X$ the partial transition function, x_0 the initial state and $X_m \subseteq X$ the subset of marked states. The designer checks the existence of the product composition [9] of 'G' and 'E', meaning a subset of system behaviour complying with the specification. This behaviour also represents the restriction on the process behaviour through action of a supervisor 'S'. If the language generated by $G \times E$ can be controlled with respect to 'G', it then represents the obtained behaviour through the action of 'S' on 'G' (S/G). If the generated language cannot be controlled, we determine the largest controllable sublanguage, called "supremal controllable sublanguage". In the second problem, the designer has the process G and already knows the expected behaviour for the controlled process (specified by a language K). He then checks whether there is supervisor that can limit the process G behaviour, such as $L(S/G) = K$. If this supervisor exists, the language K is the language of the controlled process S/G. If the supervisor does not exists, we proceed to obtain the largest sublanguage of K using the supremal controllable in the same way as in the direct approach.

III. Mode switching

A. *General framework applied to simple case*

A system can be operated in different and numerous modes. In mode switching terms, our contribution proposes a framework, in which:

- Each mode is studied independently and separately. In most cases, a mode is characterized by the components used and by the components necessary to enter or exit of it;
- The (intramodal) specifications to be satisfied by the system in this mode are studied on models built independently of others specifications. SCT theory is applied "normally" in each mode (we limit ourselves to a centralized case);
- The (intermodal) framework, including the (intermodal) specifications used to extend the behaviour of modes and have all possible trajectory, and the (switch) specifications used to limit the commutation phenomena, i.e. forbid the trajectories to switch that are not desired. Of course, this will depend if switch events can be observed and controlled or not.

The main problem is to determine the state of the model (process, controlled process and specification), when we leave a mode (the "initial mode") to enter another mode (the "final mode").

In this paper, as example and for a better understanding, we apply directly our framework on a simple system, represented "Fig. 2(a)", used for different tasks. The system has two behaviours depending on a machine failure, and in this case, the system performs another task.

The following sections describe successively the intramodal framework for each mode and the intermodal design process applied on the given example.

B. *Definitions*

Our system, shown "Fig. 2(a)", is composed by two different machines and by one buffer. The machine 1 can break down due to malfunctioning and this fact is modelled using the event f_1. Repair is modelling using the event r_1. These switch events involve a switch for the system from nominal to degraded mode.The machine 2 works both nominal and degraded modes, but its dynamic is different. In the nominal mode, it takes pieces in buffer, while in the degraded mode it takes pieces somewhere else. In this system, the buffer has a capacity of one and is considered as specification.

***Definition** 1:* A component is modelled by an automaton $G^{\mathcal{C}_i}$ where $G^{\mathcal{C}_i} = (Q^{\mathcal{C}_i}, \Sigma^{\mathcal{C}_i}, \delta^{\mathcal{C}_i}, q_0^{\mathcal{C}_i}, Q_m^{\mathcal{C}_i})$, with:

- $Q^{\mathcal{C}_i}$ is the state set of component i;
- $\Sigma^{\mathcal{C}_i}$ is the event set of component i, including three partitions:
  - $\Sigma^{\mathcal{C}_i} = \Sigma_c^{\mathcal{C}_i} \cup \Sigma_{uc}^{\mathcal{C}_i}$ with $\Sigma_c^{\mathcal{C}_i} \cap \Sigma_{uc}^{\mathcal{C}_i} = \phi$;
  - $\Sigma^{\mathcal{C}_i} = \Sigma_o^{\mathcal{C}_i} \cup \Sigma_{uo}^{\mathcal{C}_i}$ with $\Sigma_o^{\mathcal{C}_i} \cap \Sigma_{uo}^{\mathcal{C}_i} = \phi$;
  - $\Sigma_{\leftrightarrows}^{\mathcal{C}_i} \subset \Sigma^{\mathcal{C}_i}$ is the set of switch events;
- $\delta^{\mathcal{C}_i}$ is the transition function and includes $\delta_{\leftrightarrows}^{\mathcal{C}_i}$ which represents the set of switch transitions;
- $q_0^{\mathcal{C}_i}$ is the initial state of component i;
- $Q_m^{\mathcal{C}_i}$ is the marked states set of component i;

***Definition** 2:* The list of operating modes is denoted by $\mathcal{M} = \{M_1, M_2, \ldots, M_n\}$, where $n \in \mathbb{N}$ and $n \geq 1$ (by convention, we assume the initial active mode is M_1). $\mathcal{C}$ is the list of components and $\mathcal{C}^{M_j}$ the list of components used in the mode M_j, where $\mathcal{C}^{M_j} = \mathcal{C}^{M_j \circlearrowleft} \cup \mathcal{C}^{M_j \leftarrow} \cup \mathcal{C}^{M_j \rightarrow}$ such that:

- $\mathcal{C}^{M_j \circlearrowleft}$ is the list of components representing the intramodal behavior of mode M_j;
- $\mathcal{C}^{M_j \leftarrow}$ is the list of components that can enter into the mode M_j;
- $\mathcal{C}^{M_j \rightarrow}$ is the list of components that can exit from mode M_j;
- $\mathcal{C}^{M_j \leftrightarrows} = \mathcal{C}^{M_j \leftarrow} \cup \mathcal{C}^{M_j \rightarrow}$ is the list of switch components;

No particular relation is assumed to exist between $\mathcal{C}^{M_j \circlearrowleft}, \mathcal{C}^{M_j \leftarrow}$ and $\mathcal{C}^{M_j \rightarrow}$ except they are all included in $\mathcal{C}$: in particular, a component can be included in :

- $\mathcal{C}^{M_j \circlearrowleft}$, but not be a switch component of $\mathcal{C}^{M_j \leftrightarrows}$.
- $\mathcal{C}^{M_j \circlearrowleft}$ and in $\mathcal{C}^{M_j \leftarrow}$. It means this component is used in the mode and is necessary to enter into this mode,
- $\mathcal{C}^{M_j \leftarrow}$ or $\mathcal{C}^{M_j \rightarrow}$. It means this component is only necessary to switch (enter or exit).

$\Sigma_{\leftrightarrows}^{M_j} \subset \Sigma^{M_j}$ is the set of commutation events of mode M_j: $\Sigma_{\leftrightarrows}^{M_j} = \bigcup_{i \in \mathcal{C}^{M_j}} \Sigma_{\leftrightarrows}^{\mathcal{C}_i}$.

In the given example, we have two modes ($\mathcal{M} = \{N, D\}$). The nominal modes represent the usual work where machine 1, illustrated "Fig. 2(b)", produces a piece that is loaded in the buffer, then is taken by machine 2, shown "Fig. 2(c)". In the case where the machine 1 breaks down, the machine 2 works alone. For the mode N, $\mathcal{C}^N = \{G^{C_1}, G^{C_2}\}$, $\mathcal{C}^{N\circlearrowright} = \{G^{C_1}, G^{C_2}\}$, and $\mathcal{C}^{N\leftrightarrows} = \{G^{C_1}\}$. For the mode D, $\mathcal{C}^D = \{G^{C_1}, G^{C_2}\}$, $\mathcal{C}^{D\circlearrowright} = \{G^{C_2}\}$, and $\mathcal{C}^{D\leftrightarrows} = \{G^{C_1}\}$.

***Definition** 3:* Let a mode automaton representing the switch behaviour of the system, that represents also the desired switch behaviour of the designer. Formally, this automaton is denoted by $G^{Ma} = (Q^{Ma}, \Sigma^{Ma}, \delta^{Ma}, q_0^{Ma}, Q_m^{Ma})$ such that:

- $Q^{Ma} = \mathcal{M}$;
- $\Sigma^{Ma} = \bigcup_{a\in\mathcal{M}} \Sigma^{M_a}_{\rightleftarrows}$;
- $q_0^{Ma} = \mathcal{M}_1$;

This automaton serve us to easily add information in which mode the system is. It also allows us to add strategies to switch in modifying it. In the example, the mode automaton G^{Ma} is shown "Fig. 2(d)".

C. *Intramodal design*

For each mode M_j, the process $G^{M_j\circlearrowright}$ resulting from parallel composition [9] of automata $G^{\mathcal{C}_i}$ of components used in this mode and is defined on $\Sigma^{M_j\circlearrowright} = \bigcup_{i\in\mathcal{C}^{M_j\circlearrowright}} \Sigma^{\mathcal{C}_i}$. For each mode M_j, the specification $E_k^{M_j\circlearrowright}$ (problem 1) or the desired language $K^{M_j\circlearrowright}$ (problem 2) are defined on the alphabet $\Sigma^{M_j\circlearrowright}$. The overall specification $E^{M_j\circlearrowright}$ is the product composition of automata of specifications to be complied with in this mode. After designing the 'n' modes required, the designer obtains 'n' uncontrolled processes $G^{M_j\circlearrowright}$, 'n' specifications $E^{M_j\circlearrowright}$ and 'n' controlled processes $S^{M_j\circlearrowright}/G^{M_j\circlearrowright}$ (called from now $G_{sup}^{M_j\circlearrowright}$). It remains to consider the switching modes.

The intramodal design is very similar to the process of supervisory control theory used to synthesize the command law. In the intramodal case, we limit us to intramodal behaviour of each mode to build $G_{sup}^{M_j\circlearrowright}$.

In the nominal mode of the example, the models must be controllable to respect the specification defined by $E^{N\circlearrowright}$, shown "Fig. 2(f)", and that represents the capacity of the buffer and the work that machine 2 must do, i.e. the task represented by s_{22} is forbidden and only the task launched by s_{21} is allowed. As the system is uncontrollable, we have to build the supremal controllable sublanguage, illustrated "Fig. 2(g)". By the same way, we build the degraded mode $G_{sup}^{D\circlearrowright}$, shown "Fig. 2(i)", which respects the specification defined by $E^{D\circlearrowright}$, shown "Fig. 2(h)" At this step, we have completely built the intramodal behavior of modes and we are sure the behaviour of each mode respects expected properties.

D. *Intermodal design*

In this section, we focus on the intermodal behavior of modes, and take into account the behaviour that could lead to switch between modes. The framework we use is shown in fig.1.

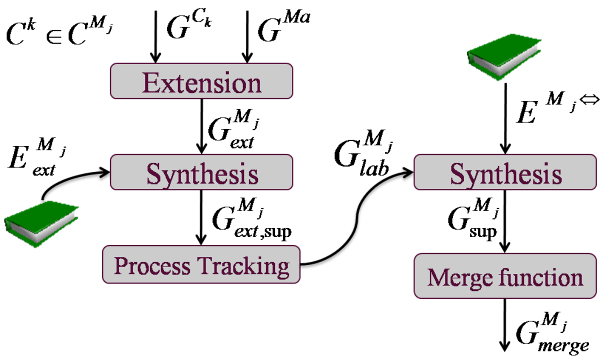

Fig. 1. Intermodal design framework

This framework includes different steps allow us to identify all trajectories connecting modes, and to be able to forbid some of them. The first step is the extension of the mode's dynamics. The second one is the synthesis of these extended mode's behaviours by the extended intramodal specifications regarding these new dynamics and by the intermodal specifications. The aim of the next steps is to avoid the non-determinism (step three "process tracking") that involve the reduction complexity of models of mode (step five "merge function"). Between the last two steps, we can use the synthesis (step four) again, to forbid the undesired trajectories. The final models resulting of the fifth step are the command law for mode.

*1) **Extension of process**:* While the intramodal framework, we focus on the intramodal behaviour of each mode. Nevertheless, to be able to identify all trajectories relying modes, we have to overload the mode's dynamic with some components' dynamic. Indeed, the intramodal framework does not take the switch dynamics into account. To obtain the extended models $G^{M_j}_{ext}$, we use parallel composition of both included components $G^{\mathcal{C}_i}$ in $\mathcal{C}^{M_j}$ (and not only $\mathcal{C}^{M_j\circlearrowright}$), and the mode automaton G^{Ma} shown "Fig. 2(d)".

***Definition** 4:* Let $G^{M_j}_{ext}$ be such that :

$$G^{M_j}_{ext} = (Q^{M_j}_{ext}, \Sigma^{M_j}_{ext}, \delta^{M_j}_{ext}, q^{M_j}_{ext,0}, Q^{M_j}_{ext,m}),$$

where : $G^{M_j}_{ext} = G^{Ma} ||_{A\in\mathcal{C}^{M_j}} A$

This extension ensures that the whole dynamic is able to detect the trajectories relying modes and it allows us to switch. The extended model G^N_{ext} applied for the nominal mode is shown "Fig. 3(a)".

*2) **Extended intramodal and intermodal specifications**:* The synthesis of the intramodal framework only deals with built specifications regarding taken components in the intramodal behaviour of modes. In this step, we have to take into account specifications more that intramodal framework to respect the intermodal behaviour of each mode. For that, the specifications are of two types.

- Extended intramodal specifications. We have to extend these specifications regarding the news added components in the extension step. Indeed, extend the intramodal behaviour of modes involve to extend the intramodal specification to keep respecting the global behaviour. The extended intramodal specifications is shown "3(b1)" for nominal specification.

- Intermodal specifications represent specification which are not necessary in the intramodal behaviour, but can modify the trajectory to switch. we then have to take in consideration these specifications. The intermodal specifications is shown "3(b2)" for nominal specification.

The product composition of these both specifications results the extended intermodal specification $E_{ext}^{M_j}$. The synthesis is applied by the same way than the synthesis in intramodal framework. For each mode, we obtain the model $G_{ext,sup}^{M_j}$ which integrally has both intramodal and intermodal dynamics. "3(c)" represents the model of the supremal controllable sublanguage for the nominal mode ($G_{ext,sup}^{N}$)

3) ***Process tracking****:* The correlation between modes is represented by switch events generated by components (shared or not). We have indeed to identify which switch events will exit of the initial mode to go into the final mode. The dynamic behaviour of the initial mode stops when the dynamic behaviour of the final mode begins. To identify these connections between modes, we take all traces leading to a switch event in the language of the initial mode and project them onto the final mode. In other words, we let $K_{M_j}(G^{M_j} \rightarrow G^{M_k})$, the desired language which generates switch events in mode M_j leading to the mode M_k and including all traces leading to an occurrence switch event. The words in $K_{M_j}(G^{M_j} \rightarrow G^{M_k})$ may not have any switch events but would lead to a state where a switch event could happen. To follow these traces from the initial mode and detect them in the final mode to identify the equivalent states where a switch event could happen, we use the extended projection function introduced by authors of [6].

Formally, the extended projection function $P_{j,k}$ is defined as follows:

Definition *5:* Let $P_{j,k} : \Sigma_j^* \rightarrow \Sigma_k^*$ such as $\forall \sigma \in \Sigma_j$ and $\forall s \in \Sigma_j^*$:

$$P_{j,k}(\varepsilon) = \varepsilon$$
$$P_{j,k}(s\sigma) = \begin{cases} P_{j,k}(s)\sigma \text{ if } \sigma \in \Sigma_j \cap \Sigma_k \\ P_{j,k}(s) \quad \text{if } \sigma \in \Sigma_j \backslash \Sigma_k \end{cases}$$

In words, this extended projection function definition limits neither alphabet Σ_j nor Σ_k and in the case in which $\Sigma_k \subseteq \Sigma_j$, this function is equivalent to the projection used in SCT [9]. This function *erases* effectively from a string s those events σ that are not included in the set of common events $\Sigma_j \cap \Sigma_k$. This allows us to obtain only the equivalent trace in the new mode.

In the following procedure, we apply the above definitions to track trajectories representing commutations on the given example: To recall, we have two modes N and D, where G_{ext}^N and G_{ext}^D are the parallel composition of both components $G^{\mathcal{C}_1}$ and $G^{\mathcal{C}_2}$, and the mode automaton G_{Ma}. The model of the nominal mode G_{ext}^N is controlled by the extended specification E_{ext}^N while the degraded mode G_{ext}^D is controlled by the extended specification E_{ext}^D. We now have two modes represented by the model $G_{ext,sup}^N$ for the nominal mode, and by the model $G_{ext,sup}^D$ for the degraded mode. The initial (nominal) mode is $G_{ext,sup}^N$, and the final (degraded) mode is $G_{ext,sup}^D$. The switch events f_1 and r_1 are generated by the component $G^{\mathcal{C}_1}$.

Procedure *1:* The event f_1 causes the switch from initial to final mode, while the event r_1 causes the switch from final to initial mode. Thus, $f_1 \in (\Sigma_{\rightarrow}^N and\ \Sigma_{\leftarrow}^D)$, and $r_1 \in (\Sigma_{\leftarrow}^N and\ \Sigma_{\rightarrow}^D)$.

1) We begin by choosing two modes where we calculate all trajectories, we then obviously choose the only two modes of the example.
   a) We calculate the language $K_N(G^N \rightarrow G^D)$ that represents all traces leading to a state q^N where an occurrence of switch event f_1 could happen in G^N.
   b) We project $K_N(G^N \rightarrow G^D)$ onto $L(G^D)$ to obtain $K_D(G^N \rightarrow G^D)$. These traces lead us to states q^D where $\delta(q^D, f_1)$ exists.
   c) At this step, there are two possible cases for each trace:
      i) The switch event (identified by one trace) exists in the initial mode and final mode. In this case, we just have to rename this switch event by adding specific sub-indices;
      ii) The switch event exists in the initial mode, but not in the final mode. This case means one trace, which deactivates the existing initial mode, but there is none in the final mode to activate it. That kind of trace, identified by a switch event, generates an error if we implement the model without controlling this trace.
2) With the same method, we repeat this operation to calculate all traces leading from degraded mode G^D to nominal mode G^N. These commutations represent the repair event of machine 1.⧫

The second case of step 'c' is really important for us, because this kind of switch gives us the information on bad trajectories while mode switching. It means that, in the case where at least one trace of this type exists, our system can be broken and unable to keep running; this is, in total contradiction with the expectations and requirements. Thus, we have to forbid all traces that are leading us to one of these incompatible states in "intermodal" specifications (treated in the next section III-D.4).
The labeled model G_{lab}^N representing the nominal mode is shown "Fig. 4(a)".

4) ***Switch specifications****:* The switch specifications step is about some trajectories we had detected as undesired during the process tracking, or because we do not want to authorize some dynamic to switch. It is the last step where we forbid some dynamics, and we see that a controllable process exists. Building these models of specifications is really easy, because we use the new switch events label as we did in the last step (III-D.3). In others word, the language of switch specification is defined by the alphabet $\Sigma_{lab}^{M_j}$ without the switch events we have to forbid. Applying these specifications gives us the models $G_{sup}^{M_j}$ that we reduce in the next step in merging some states. At the end of this step, the model are controlled, no model of the modes has more than one switch event with the same label, the undesired switch events have been forbidden, and for each switch event, there is only one other switch event in another mode that has the same label. Now, we can reduce the model of the modes by using a merge function.

*5) **Merge function:*** The merge function reduce the complexity of the model in keeping only the intramodal behaviour of each mode and including the useful intermodal behaviour. To obtain the smallest size of each mode, we use the next procedure applied on the given example.

***Procedure** 2:* Take the last automata $G_{sup}^{M_j}$. Like in the section about extension, we extend each mode with the mode automaton G^{Ma}, each state of mode has a name including the state where G^{Ma} is. In other words, we know, for each state, in which mode the system is. As we are now interested only by the intramodal behaviour of each mode, we just have to "remove" the behaviour which does not represent the intramodal behaviour and add an $idle$ state representing the mode when it will be deactivated to respect the specification that only one mode can be active in the same time. To do this, we execute the next procedure.

1) We determine in $G_{sup}^{M_j}$, a merge set $Q_{mer}^{M_j} \subset Q_{sup}^{M_j}$. The states included in $Q_{mer}^{M_j}$ will be states that are not significant for the mode.
   - Not significant for the nominal mode 'N' are all states which do not have 'N' part of name. These states are certainly added during the extension of the different models.
   - Not significant for degraded mode called, for example, 'D' are all states not having 'D' in a part of their name. It means these states do not represent the intramodal "$degraded$" behaviour of mode D.
   - With this easy identification, we can identify all states that are not included in the mode where they are at the instant, and include them in the merge set $Q_{mer}^{M_j}$.
2) The new state formed by the merge is called $q_{id}^{M_j}$.
3) We remove all self-loops at $q_{id}^{M_j}$.
4) If the initial state is included in $Q_{mer}^{M_j}$, then $q_{id}^{M_j}$ will be the new initial state.
5) If a marked state is included in $Q_{mer}^{M_j \leftrightarrows}$, then $q_{id}^{M_j}$ will be a marked state.⧫

In the example, the merge function applied on G_{sup}^{N}, and G_{sup}^{D}, give us the final models G_{merge}^{N}, illustrated "Fig. 4(b)", and G_{merge}^{D}, shown "Fig. 4(c)". These models represent the command law we can use to implement and control the system.

It is well-known that merging states in an automaton can cause non-determinism [9]. It is to avoid this that we used the extended projection function as in section III-D.3. We had to keep knowledge about the switch events that produced them. In other words, we anticipated the merge function in using the extended projection function. Using both of these functions allows the reduction of complexity without having non-deterministic automaton.

IV. Conclusion

The main contribution of this paper is to present a advanced framework allows us to design, by multimodal approach, a whole system. The first step is the intramodal study where each mode is studied independently. The second step, and the major contribution of this work, focuses on formal way to design complete modes including its different switch dynamics, identify some incompatibilities during the mode switching and forbid these incompatibilities by using SCT. Being an off-line study, this work gives us a formal method to completely design a command law respecting all specifications the system needs. Current research involves defining strategies, when incompatible states have been recognized, using uncontrollable switch events, and when the supremal controllable does not give us satisfaction on the general specification of our system.

References

[1] D. Harel, "Statecharts: A visual formalism for complex systems," *Science of Computer Programming*, vol. 8, no. 3, pp. 231–274, June 1987. [Online]. Available: citeseer.ist.psu.edu/harel87statecharts.html

[2] F. Jahanian and A. Mok, "Modechart: A specification language for real-time systems," *IEEE Trans. Softw. Eng.*, vol. 20, no. 12, pp. 933–947, 1994.

[3] F. Maraninchi and Y. Rémond, "Mode-automata: a new domain-specific construct for the development of safe critical systems," *Science of Computer Programming*, vol. 1, no. 46, pp. 219–254, March 2003.

[4] J.-P. Talpin, C. Brunette, T. Gautier, and A. Gamatie, "Polychronous mode automata," in *EMSOFT '06: Proceedings of the 6th ACM & IEEE International conference on Embedded software*. New York, NY, USA: ACM Press, 2006, pp. 83–92.

[5] P. Ramadge and W. Wonham, "The control of discrete event systems," *Proceedings of the IEEE*, vol. 77, no. 1, pp. 81–98, Jan 1989.

[6] O. Kamach, L. Piétrac, and E. Niel, "Multi-model approach to discrete events systems: Application to operating mode management," *Mathematics and Computers in Simulation, Elsevier*, vol. 70, no. 5-6, pp. 394–407, 2005.

[7] G. Faraut, L. Piétrac, and E. Niel, "Identification of incompatible states in mode switching," *Emerging Technologies and Factory Automation, 2008. ETFA 2008. IEEE International Conference on*, pp. 121–128, Sept. 2008.

[8] W. M. Wonham, "Supervisory control of discrete-event systems. ece 1636f/1637s 2006-07," 2006, course notes, departement of Electrical and Computer Engineering, Univeristy of Toronto. [Online]. Available: www.control.toronto.edu/people/profs/wonham/

[9] C. G. Cassandras and S. Lafortune, *Introduction to discrete event systems [Second Edition]*, C. G. Cassandras and S. Lafortune, Eds. Springer, 2007.

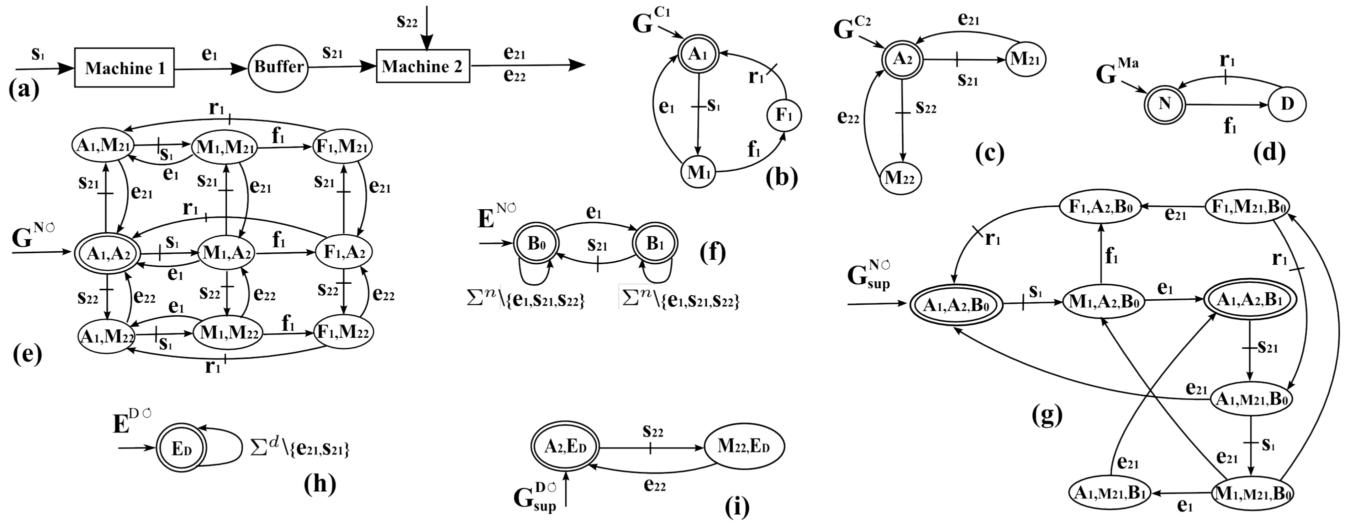

Fig. 2. Manufacturing system example : (a) system; (b) model $G^{\mathcal{C}_1}$ for machine 1; (c) model $G^{\mathcal{C}_2}$ for machine 2; (d) mode G^{Ma} representing the mode automaton; (e) model $G^{N\circlearrowleft}$ representing the parallel composition of components for the *nominal* mode; (f) specification $E^{N\circlearrowleft}$ about buffer and machine 2 for *nominal* mode; (g) model of controlled process for the *nominal* mode; (h) model of specification for *degraded* mode; (i) model under controlled for the degraded mode.

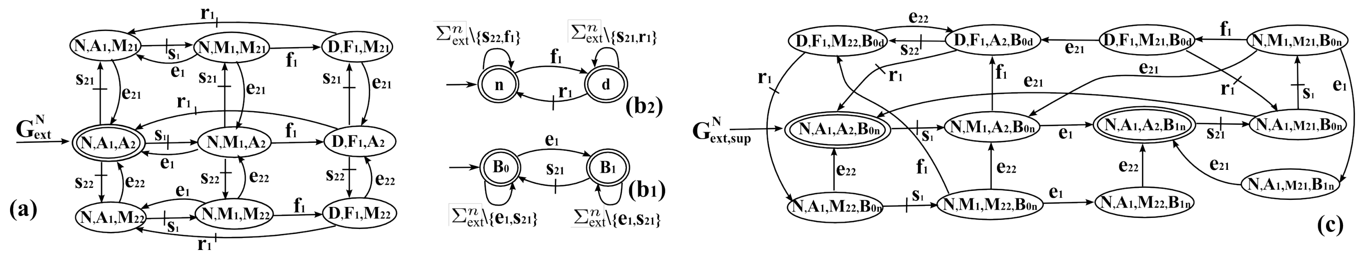

Fig. 3. Manufacturing system example : (a) extended model for the nominal mode; (b) extended specification for the nominal mode; (c) model representing the supremal controllable sublanguage for the nominal mode.

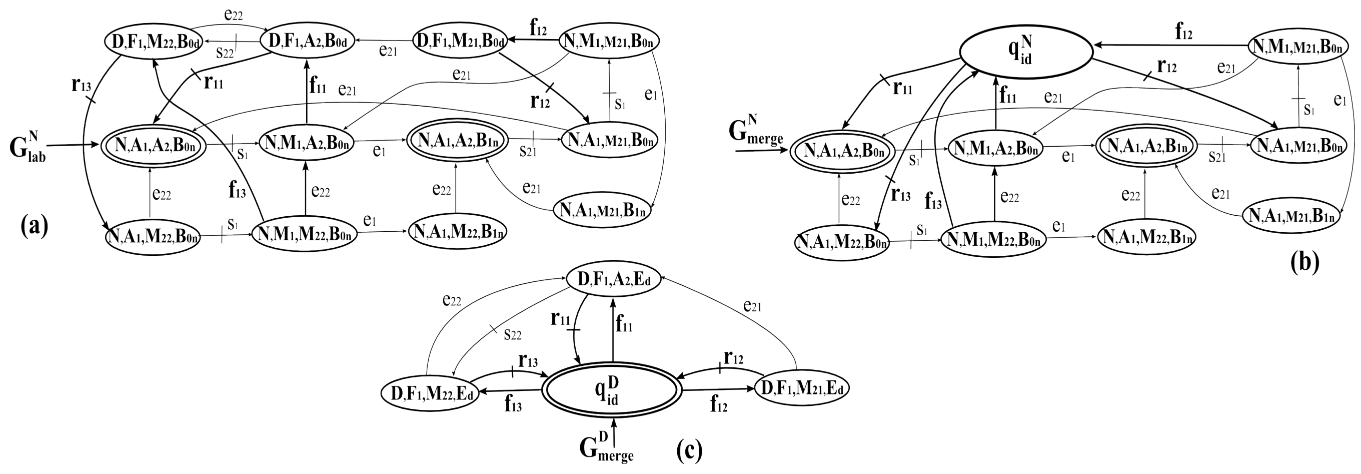

Fig. 4. Manufacturing system example : (a) labelled model for the nominal mode; (b) merged model representing the final mode for the nominal mode; (c) merged model representing the final model for the degraded mode.