Identification of Incompatible states in Mode Switching

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

Mode management is one of the problems in Discrete Events Systems control design. Even based on a simple specification, it is very difficult to prove that models of each mode and mutual interaction are correct. This paper demonstrates that Supervisory Control Theory is an effective tool for detecting specification incompatibilities because it clearly separates process, models and specifications. We use simple cases to present a method that introduces flexibility into mode specification. This method can be used to adjust or to modify incompatibilities between specifications and thereby promotes correct mode switching.

1 Introduction

In relation to their component structures, multimode systems exhibit different expected, and sometimes conflicting, behaviours. Their proper behaviour and more significantly, their proper coordination in the event of failures have become essential in control applications. Improper behaviours lead typically to very serious consequences, including human operator injury and material losses, and affect a wide range of industrial applications. The scope and complexity of reactive control for multimode management has grown tremendously in these years and fault effect consideration has led to stricter requirements, the validation of which has become increasingly important. However, when we compare reactive control development methods and switching mode validation, we realize that the latter process cannot be managed on the basis of conventional validation techniques (time-consuming scenario reviews, simulation and testing) and this calls for formal approaches. Conventionally, a mode is defined by a set of components running and mutually interacting to fulfil a required task. In this context, mode switching means changing component configuration. Reconfiguration applications are typically component-based, event-driven, and characterized by clear separation between the system to be reconfigured and its reconfiguration prescription. Their adaptation in the Discrete Event Systems (DES) framework surpasses convenience and is closely related to switching mode studies.

Most DES research studies to date have focused on modelling the different existing configurations using compositional formalisms: state charts [2], mode charts [3], hierarchical finite state machines and mode automata [8, 11]. Few existing studies (excluding failure detection for reconfiguration [9]) take into account commutation phenomena, which indeed constitute the major problem concerning reconfiguration issues. Reconfiguration means that, due to some external or ultimate internal reason - sensor/actuator failure, low energy - the currently active components can no longer ensure the attempted mode, without reference to its redundancy. Moreover, depending on the failure event seriousness, the assignment has to be interrupted and restarted otherwise, if the current operating mode cannot be maintained. Commutation law-based properties, such as reachability, observability and controllability (latter two applicable to conventional control theory [10, 1]) may be considered from a qualitative standpoint. Reachability is perhaps the major property concerning commutation phenomena; it implies that a single trace could exist and be managed, to switch from one mode to another one. Reachability will be discussed in this paper in terms of component state coherence compatibility. This coherence must be determined, when shared components are used in the both modes affected by reconfiguration.

Our approach is based on the work of [5, 6], which allows us to study the internal behavior of each mode independently. The main assumption is that the system is effectively operated in only one mode at any one time. Based on previous switching management studies [7, 4] we propose both process and controlled process model extensions to formalize compatible states. Compatible states are those for which common component states are functionally coherent in considered modes (incoming and outgoing operating modes). Our objectives lead us to adopt a multi-model approach. We also propose a framework for studying mode switching and, when required, a framework for studying and remedying deadlocks.

This paper focuses on the switching problem within the context of automata-modelled DES. Section 2 introduces Supervisory Control Theory (SCT), on which this work is formally based. We define explicitly the SCT approach adopted to describe the switching capability used in its direct (controller synthesis with respect to process and requirement) and inverse (controller synthesis with respect to process and attempted controlled behaviour) approaches. By distinguishing common, faulty or good components, Section 3 formalizes both intramodal and intermodal operation. The notion of compatibility is then considered by managing possible switched states (still valid with common components), when typical faulty component-related events occur (i.e. failure, repair). Section 4 provides a simple example which illustrates the new notions and extended models introduced in this paper.

2 Supervisory control theory

Ramadge and Wonham's theory [10] 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 [12] 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), $\delta : Q \times \Sigma \rightarrow Q$ 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.

1. Problem 1: given process G and specification E, is there a system behaviour that complies with this specification?
2. Problem 2: given process G and desired language K, can we restrict the process dynamic evolution of K?

Problem 1 illustrates a "direct approach". Here, the specification 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 [1] 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', such as $S : L(G) \longrightarrow \Gamma$ is a function from the language generated by G to the power set of enabled events $\Gamma = \{\gamma \in Pwr(\Sigma) | \Sigma_u \subseteq \gamma\}$ where $Pwr(\Sigma)$ is the set of all subsets of Σ. If the language generated by $G||E$ 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". The process or specification must be modified, if the supremal controllable sublanguage is empty.

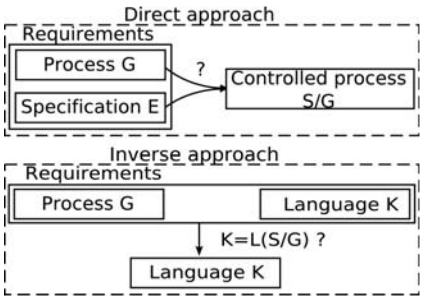


Figure 1. Two types of problems

Problem 2 illustrates an "inverse approach", in which the designer 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.

3 Mode switching

3.1 Overview

In this paper, we consider systems with several components (a plant with machines, a machine with actuators, etc.), which are used for different tasks. To increase availability, the system is required to remain in operation even if its components fail and this is only possible if a number of alternative components are available. These available components can stand in for failed components by performing the same task or by performing other tasks.

The system can be operated in one particular mode (production, initialization, etc.) at any time. In mode switching terms, our contribution comprises proposing 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) specifications about switching modes can occur directly in more that intramodal specifications in the considered modes. 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").

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

3.2 Definitions

A system is composed by different components. The dynamic of each component is the same irrespective of the system mode. These dynamics include possible failures and recoveries. Such events will be used to model switching modes.

Definition 1 *A component is modelled by an automaton G_i where $G_i = (Q_i, \Sigma_i, \delta_i, q_{0,i}, Q_{m,i})$, with:*

- *Q_i is the state set of component i;*
- *Σ_i is the event set of component i, including three partitions:*
  - *$\Sigma_i = \Sigma_c \cup \Sigma_{uc}$;*
  - *$\Sigma_i = \Sigma_o \cup \Sigma_{uo}$;*
  - *$\Sigma_i^{\leftrightarrows} \subset \Sigma_i$ the set of switch events such as it is composed by fault events "f_i" and repair events "r_i";*
- *δ_i is the transition function and includes $\delta_i^{\leftrightarrows}$ which represents the set of failures or repairs transitions;*
- *$q_{0,i}$ is the initial state of component i;*
- *$Q_{m,i}$ is the marked states set of component i;*

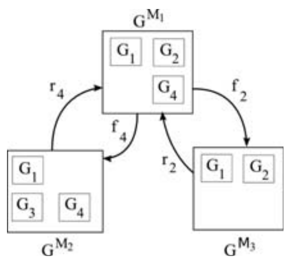
Definition 2 *The list of operating modes is denoted by $\mathcal{M} = \{M_1, M_2, \ldots, M_n\}$, where $n \in \mathbb{N}$ and $n \geq 1$ (by convention, we assume the initial active mode is M_1). $\mathcal{C}$ is the list of components and $\mathcal{C}^{M_j \leftrightarrows}$ the list of components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

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



$\mathcal{M} = \{M_1, M_2, M_3\}$
$\mathcal{C} = \{G_1, G_2, G_3, G_4\}$
$\mathcal{C}^{M_1} = \{G_1, G_2, G_4\}$
$\mathcal{C}^{M_2} = \{G_1, G_3, G_4\}$
$\mathcal{C}^{M_3} = \{G_1, G_2\}$
$\Sigma^{\leftrightarrows} = \{f_2, r_2, f_4, r_4\}$
$\Sigma^{M_1\leftrightarrows} = \{f_2, r_2, f_4, r_4\}$
$\Sigma^{M_2\leftrightarrows} = \{f_4, r_4\}$
$\Sigma^{M_3\leftrightarrows} = \{f_2, r_2\}$

Figure 2. Example of mode decomposition

Figure.2 is an example of mode decomposition. We have three modes, in which the mode G^{M_1} is composed of components G_1, G_2 and G_4. From this (nominal) modes, a switch is possible to the (degraded) mode G^{M_2}, if the component G_4 breaks down, or G^{M_3} if it is the component G_2 that breaks down. Thus, the mode G^{M_3} is only composed of component G_1, (i.e. $\mathcal{C}^{M_3\circlearrowleft} = \{G_1\}$), but is extended with the component G_2 ($\mathcal{C}^{M_3\leftarrow} = \mathcal{C}^{M_3\rightarrow} = \{G_2\}$) because it is this component that causes the switch from the mode G^{M_1} to G^{M_3}. In the same way, the mode G^{M_2} is composed of components G_1 and G_3 ($\mathcal{C}^{M_3\circlearrowleft} = \{G_1, G_3\}$), but is also extended next with the components G_4 ($\mathcal{C}^{M_2\leftarrow} = \mathcal{C}^{M_2\rightarrow} = \{G_4\}$)that is responsible for the switch.

3.3 intramodal design

For each mode M_j, the process G^{M_j} resulting from parallel composition [1] of automata G_i of components used in this mode is defined on $\Sigma^{M_j} = \bigcup_{i \in \mathcal{C}^{M_j \circlearrowleft}} \Sigma_i$.

For each mode M_j, the specification E^{M_j} (problem 1) or the desired language K^{M_j} (problem 2) are defined on the alphabet Σ^{M_j}. The overall specification E^{M_j} 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}, 'n' specifications E^{M_j} and 'n' controlled processes S^{M_j}/G^{M_j} (called from now $G^{M_j}_{sup}$). It remains to consider the switching modes.

The intramodal design, as shown in fig.3, is very similar to process of supervisory control theory used to synthesize the command law. In the intramodal case, we limit us to internal behavior of each mode to build $G^{M_j}_{sup}$.

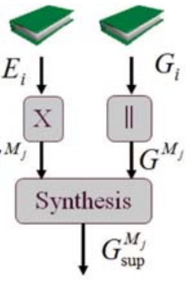


Figure 3. Intramodal design framework

3.4 Intermodal design

At this step, we have completely built the internal behavior of modes. In this section, we will include the external behavior, and detect the trajectories connecting the modes between them. The framework that we will use is shown in fig.8 in Appendix A .

3.4.1 Extension of controlled process

The system is represented by several automata $G^{M_j}_{sup}$, so we have to add an information item to the models allowing us to determine the behaviour to enter into, or to exit from, the modes. To do this, we extend each model $G^{M_j}_{sup}$ by parallel composition with the components of $\mathcal{C}^{M_j \leftarrow}$ and $\mathcal{C}^{M_j \rightarrow}$ that are not already included in $\mathcal{C}^{M_j \circlearrowleft}$.

Definition 3 *Let $G^{M_j}_{sup}$ be such that : $G^{M_j}_{sup} = (Q^{M_j}_{sup}, \Sigma^{M_j}_{sup}, \delta^{M_j}_{sup}, q^{M_j}_{sup,0}, Q^{M_j}_{sup,m})$.*
$G^{M_j \leftrightarrows}$ is the extended model of $G^{M_j}_{sup}$ where : $G^{M_j \leftrightarrows} = G^{M_j}_{sup} || (||_{A \in (\mathcal{C}^{M_j \leftrightarrows} \setminus \mathcal{C}^{M_j \circlearrowleft})} A)$

3.4.2 Process tracking

The correlation between modes is represented by switch events generated by components (shared or not). We have indeed to identify which switch event will exit of the initial mode to go into the final mode. The dynamic behaviour of the initial mode stops where 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 $K_{M_j}(G^{M_j \leftrightarrows} \rightarrow G^{M_k \leftrightarrows})$, the desired language which generates switch events in mode M_j leading to the mode M_k and including all traces leading to a first occurrence switch event. The words in $K_{M_j}(G^{M_j \leftrightarrows} \rightarrow G^{M_k \leftrightarrows})$ 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 [5].

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

Definition 4 *Let $P_{j,k} : \Sigma_j^* \rightarrow \Sigma_k^*$ such as $\forall \sigma \in \Sigma_j$ and $\forall s \in \Sigma_j^*$:*

$$P_{j,k}(\varepsilon) = \varepsilon$$
$$P_{j,k}(s\sigma) = \begin{cases} P_{j,k}(s)\sigma & \textit{if } \sigma \in \Sigma_j \cap \Sigma_k \\ P_{j,k}(s) & \textit{if } \sigma \in \Sigma_j \setminus \Sigma_k \end{cases}$$

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 [1]. 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.

Procedure 1 *Let two modes M_1 and M_2, where $G^{M_1}_{sup}$ (respec. $G^{M_2}_{sup}$) is the parallel composition of components G_a and G_c (respec. G_b and G_c). The second mode is extended with component 'G_a' responsible for the switch. These modes share the component 'G_c', we assume the initial (nominal) mode is $G^{M_1 \leftrightarrows}$, and the final (degraded) mode is $G^{M_b \leftrightarrows}$. The switch events f_a and r_a are generated by component "G_a". The event f_a causes the switch from initial to final mode, while the event r_a causes the switch from final to initial mode. Thus, $f_a \in (\Sigma^{M_1 \rightarrow} and\ \Sigma^{M_2 \leftarrow})$, and $r_a \in (\Sigma^{M_1 \leftarrow} and\ \Sigma^{M_2 \rightarrow})$.*

1. *We calculate the language $K_{M_1}(G^{M_1 \leftrightarrows} \rightarrow G^{M_2 \leftrightarrows})$ that represents all traces leading to a state $q_k^{M_1 \leftrightarrows}$ where a first occurrence of switch event f_a could happen in $G^{M_1 \leftrightarrows}$. $k \in \mathbb{N}$ and represents the index of the language leading to a different commutation.*

2. *We projet $K_{M_1}(G^{M_1 \leftrightarrows} \rightarrow G^{M_2 \leftrightarrows})$ onto $L(G^{M_2 \leftrightarrows})$ to obtain $K_{M_2}(G^{M_1 \leftrightarrows} \rightarrow G^{M_2 \leftrightarrows})$. These traces lead us to states $q_k^{M_2 \leftrightarrows}$ where $q_k^{M_2 \leftrightarrows} \in \delta(q^{M_2 \leftrightarrows}, f_a)$.*

3. *At this step, there are two possible cases:*
   - *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,*
   - *The switch event exists in the initial mode, but not in the final mode. This case means one trace, which deactivates the initial mode exists, but no one exists on the final mode to active it. It is this kind of trace, identified by a switch event, which will generate an error if we implement the model without controlling this trace.*
4. *When all words representing a possible switch event are labeled, we can remove, from the models, all no-labeled switch events (f_a and r_a in our procedure) because these represent events that will never happen, so it is unnecessary to keep the knowledge of these commutations.*
5. *We repeat this operation for all switch events identified by $K_{M_1,l}(G^{M_1\leftrightarrows} \rightarrow G^{M_2\leftrightarrows})$. So we keep the knowledge of what causes the commutation in both modes.*

The second case of step '3' is really important for us, because it is this kind of switch that gives us the incompatible states in 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 that we wanted and required. We will need to keep the knowledge of incompatible states, to forbid in "intermodal" specifications (treated in the section 3.4.4) all traces that are leading us to one of these incompatible states.
Labelling the models $G^{M_j\leftrightarrows}$ gives us the new models $G_{lab}^{M_j\leftrightarrows}$.

We have to make a comment at this point.
It is obvious that the initial state in the final mode will depend on the trace in the initial mode previously calculated. It also means that the different traces possibly lead to different initial states. Thus, when we again want to calculate the traces leading to a switch event in the final mode that go into another mode or to go back to the initial mode, we will have to take into consideration all possible initial states where we could enter. It simply means that instead of calculating all traces beginning at only one state, the initial state, in this mode we will, and should, calculate traces for each possible initial state, to avoid missing a trace existing from one particular initial state, but not from others. Nevertheless, the calculation is more complicated but not more complex.

3.4.3 Merge function

At this step, no model of the modes has more than one switch event with the same label, and for each switch event, there is at most one other switch event, in another mode, that has the same label. Now, we can abstract our model of the modes by using a merge function.

Procedure 2 *Let a automate $G_{lab}^{M_j\leftrightarrows}$ be composed by parallel composition of components include in $\mathcal{C}^{M_j}$. Let the states of $G_{lab}^{M_j\leftrightarrows}$ be named (q_a, q_b) with $q_a \in Q_a$ and $q_b \in Q_b$ (where Q_a, Q_b are the state sets of some components). Let also $Q_i = N_i \cup F_i, N_i \cap F_i = \phi$, where N_i means that the component 'i' works well and, in opposite, F_i means that the component is broken due to a failure event.*

1. *We determine in $G_{lab}^{M_j\leftrightarrows}$, a merge set $Q_{mer}^{M_j\leftrightarrows}$ with $Q_{mer}^{M_j\leftrightarrows} \subset Q_{lab}^{M_j\leftrightarrows}$. The states included in $Q_{mer}^{M_j\leftrightarrows}$ will be states that are not significant for the mode.*
   - *Not significant for the nominal mode means, for example, all states named by F_a or F_b, because the nominal mode does not have normally the behavior of G_a of G_b when one of these are broken.*
   - *Not significant for degraded mode means, for example, all states that are not named by (F_a, q_b), $\forall q_b$. This is because, in the degraded mode, we only want the behavior of our system when the component G_a is broken.*
   - *In the same way, we can make a table with all possible combinations with the states of the components. Thus, in our example, we have four modes. The nominal mode will only include states of the form (N_a, N_b) and we merge all the others states, and three degraded modes : (F_a, N_b) which is the mode where the component G_a is broken, (N_a, F_b) when the component G_b is broken and (F_a, F_b) when both components G_a and G_b are broken.*
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 $Q_{mer}^{M_j\leftrightarrows}$, 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.*

It is well-known that merging states in an automaton can cause non-determinism [1]. It is to avoid this that we used the extended projection function as in section 3.4.2. 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 to reduce complexity without having non-deterministic automaton.

At the end of this step, our models of the modes are totally built. We have some news automata $G_{merge}^{M_j\leftrightarrows} = (Q_{merge}^{M_j\leftrightarrows}, \Sigma_{merge}^{M_j\leftrightarrows}, \delta_{merge}^{M_j\leftrightarrows}, q_{merge,0}^{M_j\leftrightarrows}, Q_{merge,m}^{M_j\leftrightarrows})$ such that:

- $Q_{merge}^{M_j\leftrightarrows} = (Q_{lab}^{M_j\leftrightarrows} \setminus Q_{mer}^{M_j\leftrightarrows}) \cup \{q_{id}^{M_j}\}$,
- $\Sigma_{merge}^{M_j\leftrightarrows} =$

$$\Sigma_{lab}^{M_j\leftrightarrows} \setminus \bigcup_{i\in\mathcal{C}^{M_j\circlearrowleft}} \Sigma_i^{\leftrightarrows} \cup \bigcup_{i\in(\mathcal{C}^{M_j\leftrightarrows} \setminus \mathcal{C}^{M_j\circlearrowleft})} \Sigma_i$$

- $\delta_{merge}^{M_j\leftrightarrows} = \delta_{lab}^{M_j\leftrightarrows} \setminus \delta(q_a, s, q_b)$ with $q_a, q_b \in Q_{mer}^{M_j\leftrightarrows}$
- $q_{merge,0}^{M_j\leftrightarrows} =$

$$\begin{cases} q_{lab,0}^{M_j\leftrightarrows} & if\ q_{lab,0}^{M_j\leftrightarrows} \notin Q_{mer}^{M_j\leftrightarrows} \\ q_{id}^{M_j} & if\ q_{lab,0}^{M_j\leftrightarrows} \in Q_{mer}^{M_j\leftrightarrows} \end{cases}$$

- $Q_{merge,m}^{M_j\leftrightarrows} =$

$$\begin{cases} Q_{lab,m}^{M_j\leftrightarrows} & if\ Q_{lab,m}^{M_j\leftrightarrows} \cap Q_{mer}^{M_j\leftrightarrows} = \phi \\ Q_{lab,m}^{M_j\leftrightarrows} \setminus Q_{mer}^{M_j\leftrightarrows} \cup \{q_{id}^{M_j}\} & if\ not \end{cases}$$

3.4.4 Controlled process by intermodal specifications

We have our models of the modes, including their own internal specifications. Their size is reduced due to the merge function and the correlation between modes has been kept with the label function using the extended projection function. Resulting from all these steps, we have new automata $G_{merge}^{M_j\leftrightarrows}$, the model under control, extended, labeled, and merged of mode M_j.

We said at this end of the introduction that the system has to operate in only one mode at any one time. This is to avoid conflicts between modes, i.e. two or more modes could be activated in the same time. We also identified in Section 3.4.2 some incompatible states, where a switch event could happen, and the traces leading to these states. It meant these switch events could happen in the initial mode, but they might not exist in the final mode. These commutations will lock our system and, furthermore, could cause irreversible damage. Thus, we have to forbid the traces that could lead to a switch event.

To satisfy these specifications, included in intermodal specification called $E^{M_j\leftrightarrows}$, we propose to use as base the models shown in Fig.4. It includes the model for the nominal mode, and the model for degraded mode. These models provide the first specification to avoid conflicts between modes in certifying that only one mode is active in any one time. Moreover, we can add on these models the forbidden switch events that were previously identified. Thus, it is straightforward to build intermodal specifications in using these models and adding forbidden switch events. We can build the controlled process with this specification, and we will use the supremal controllable sublanguage in the case where the process is not controllable.

These controlled model of $G_{merge}^{M_j\leftrightarrows}$ including the switch specification ($E^{M_j\leftrightarrows}$) represent the final automata called

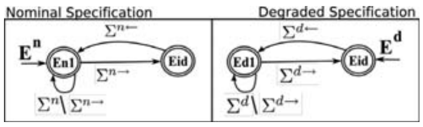


Figure 4. Base models to design intermodal specifications

$G_{sup}^{M_j\leftrightarrows}$ and it will be these automata that we will use to implement our system.

4 Example

Consider the manufacturing system illustrated in Fig.5(a); the system comprises four machines and one buffer. The machines are used to process a part and the stock is used as a ‘buffer’ between the machines. The ‘machine 1’(respec. ‘machine 2’) , denoted G_1 (respec. G_2), are modeled as shown in Fig.5(b).The automaton modelling ‘machine 3’ (respec. ‘machine 4’) are denoted G_3 (respec. G_4) and are shown in Fig.5(c). The events si and ei represent a new process and the end of the process respectively. Whilst all these events are observable, they are not necessary controllable. The ‘machine 1’ (respec. ‘machine 2’) can break down due to malfunctioning and this fact is modelled using the observable, uncontrollable event f_1 (respec. f_2). Repair is modelled using the observable, controllable event r_1 (respec. r_2).

4.1 Intramodal design

The modal decomposition of the system, where modes use components necessary to run, is modelled in Fig.5(d). These models must be controlled according to the specification defined by a specification automaton, like the nominal specification E^{nom} representing buffer specification, shown in Fig.5(e). It is unlikely that L(E^{M_j}) is controllable with respect to L(G^{M_j}), thus we use the supremal controllable sublanguage of L($G_{sup}^{M_j}$) to control our four modes, as illustrated in Fig.5(f).

4.2 Intermodal design

Our system features four modes (one nominal and three degraded). We extend each mode with switch components that we did not include before. This extension is illustrated in Fig.6(a). The nominal mode is not extended with another component because it is already composed of both machines ‘1’ and ‘2’. We extend the other modes with missing components, G_1 to G_{sup}^{d1}, G_2 to G_{sup}^{d2} and both G_1 and G_2 to G_{sup}^{d3}. We show in Fig.6(b) the extended controlled process of nominal mode. To compare at this step, the degraded modes $1, 2$ and 3 have respectively $27, 24$ and 204 states. Fig.6(c) shows again the previous nominal mode, but in which we track the first occurrence of failure events to switch (and the first occurrence of repair in degraded modes) to label these switch

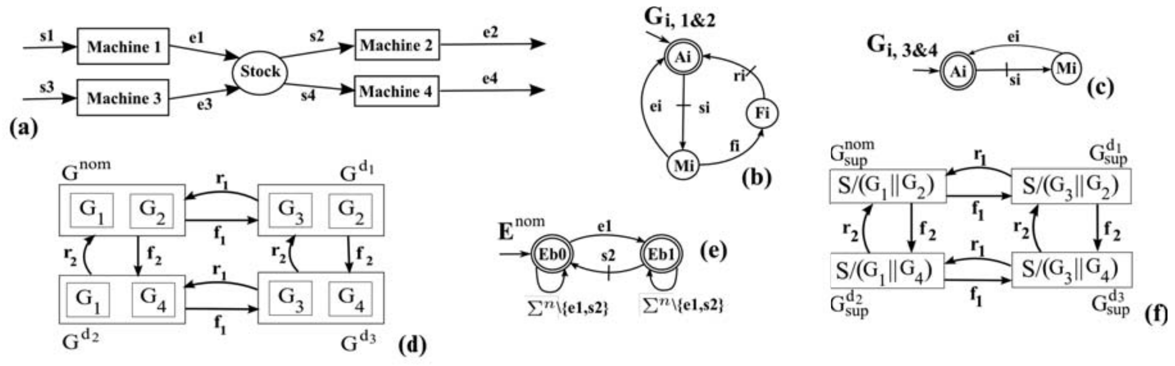


Figure 5. Manufacturing system example : (a) system; (b) model G_i for 'machines 1 and 2'; (c) model G_i for 'machines 3 and 4'; (d) modal decomposition of the system; (e) specification E^{nom} about buffer for nominal mode; (f) modal decomposition of controlled processes.

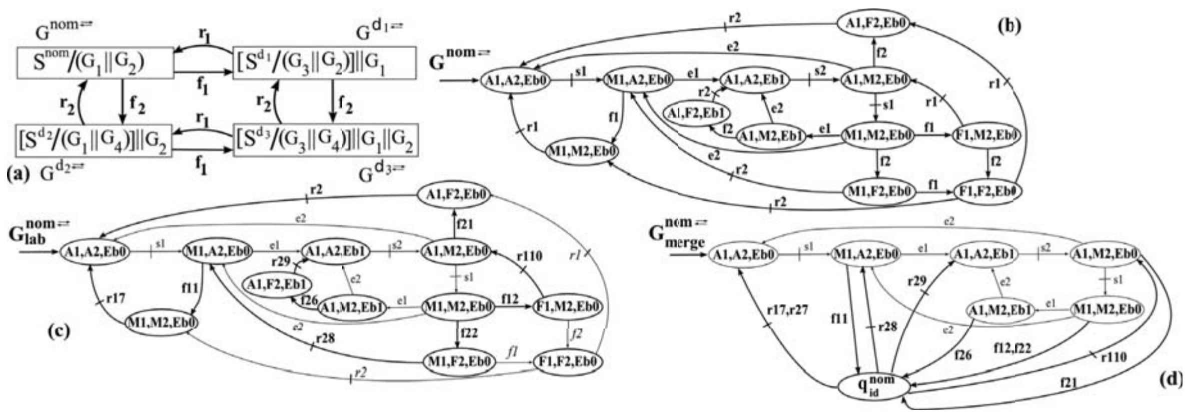


Figure 6. Manufacturing system example : (a) modal decomposition with controlled processes extended; (b) extended controlled process for nominal mode; (c) the extended controlled process of nominal mode including label of switch event; (d) labeled extended controlled process of nominal mode after merging the non-significant states.

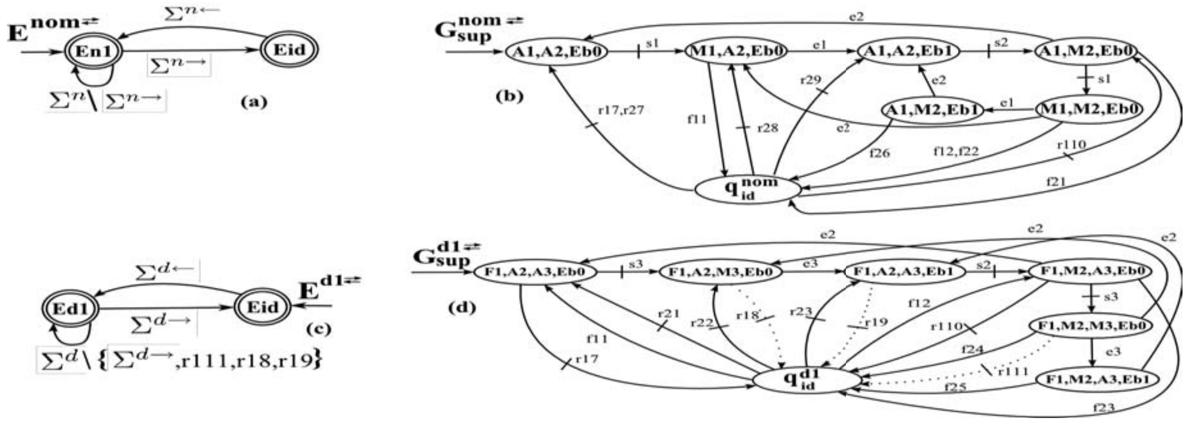


Figure 7. Manufacturing system example : (a) Switch specification of nominal mode; (b) final controlled process of nominal mode; (c) switch specification of degraded mode d_1; (d) final controlled process of degraded mode d_1.

events. We see also in this figure some switch events that were not labeled. Indeed, it means they are not significant events (either they are not the first occurrence, or there is not trace in others modes that could generate these events). At the end, we use the merge function on states representing a failure for one component, and labeled by (F_1, x) or (x, F_2). The result of the merge function is shown in Fig.6(d). We now see that we would have obtained a non-deterministic automaton if we had not labeled before using the merge function. In this case, all "r_{ix}" would be "r_i" and "f_{ix}" would be "f_i".

From $G_{merge}^{M_j \leftrightarrows}$, we apply at each mode their own specification $E^{M_j \leftrightarrows}$ as illustrated in Fig.7(a) for the nominal mode, or more interestingly, in Fig.7(c) where we have forbidden two switch events (r_{111}, r_{18} and r_{19}) because we do not want to repair while the replacement machine is running. The controlled processes of the nominal and the first degraded mode are shown in Fig.7(b) and Fig.7(d).

5 Conclusion

This paper discusses component state coherency and addresses the issue of incompatible state definition and recognition when mode switching is required, according to the hypothesis that common components run in a set of different operating modes. Incompatibility has been considered as a nonexistent state (component state incoherency) for either the current operating mode or the switched operating mode. We addresses formally this recognition capability in terms of reachability. Switching management by tracking configuration changes forms the starting stage and the major contribution of this work focuses on formalizing incompatibility of this nature. Current research involves defining strategies, when incompatible states have been recognized, using uncontrollable switch events, and when the supremal controllable do not give us satisfaction regarding the general specification of our system.

6 Acknowledgement

The authors would like to thanks Stéphane Lafortune, Richard Hill and Hongwei Liao for the comments that they provided on the preparation of this paper.

7 Appendix A

We give the framework used in section.3.4, hoping it can give some help to follow the article and avoid confuse with notation. The framework is shown in fig.8

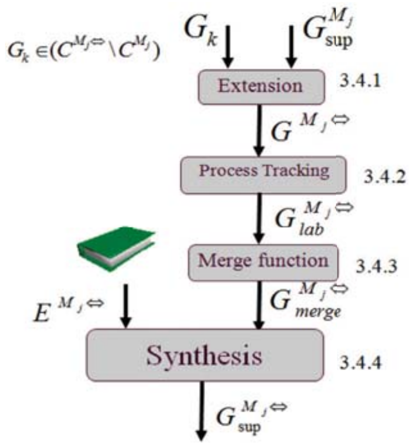

$G_k \in (C^{M_j \Leftrightarrow} \setminus C^{M_j})$
G_k
$G_{sup}^{M_j}$
Extension
3.4.1
$G^{M_j \Leftrightarrow}$
Process Tracking
3.4.2
$G_{lab}^{M_j \Leftrightarrow}$
Merge function
3.4.3
$E^{M_j \Leftrightarrow}$
$G_{merge}^{M_j \Leftrightarrow}$
Synthesis
3.4.4
$G_{sup}^{M_j \Leftrightarrow}$

Figure 8. Intermodal design framework

References

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

[2] D. Harel. Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, 8(3):231–274, June 1987.

[3] F. Jahanian and A. Mok. Modechart: A specification language for real-time systems. *IEEE Trans. Softw. Eng.*, 20(12):933–947, 1994.

[4] L. N. E. Kamach, O.; Pietrac. Forbidden and preforbidden states in the multi-model approach. *Computational Engineering in Systems Applications, IMACS Multiconference on*, 2:1550–1557, 4-6 Oct. 2006.

[5] 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*, 70(5-6):394–407, 2005.

[6] O. Kamach, L. Piétrac, and E. Niel. Supervisory uniqueness for operating mode systems. In *16th IFAC world congress*, Prague, 4–8 July 2005.

[7] A. Khatab and E. Niel. State feedback stabilizing controller for the failure recovery of timed discrete event systems. In *WODES'02 : Proceedings of the Sixth International Workshop on Discrete Event Systems (WODES'02)*, page 113, Washington, DC, USA, 2002. IEEE Computer Society.

[8] F. Maraninchi and Y. Rémond. Mode-automata: a new domain-specific construct for the development of safe critical systems. *Science of Computer Programming*, 1(46):219–254, March 2003.

[9] A. Paoli and S. Lafortune. Safe diagnosability for fault-tolerant supervision of discrete-event systems. *Automatica*, 41(8):1335–1347, August 2005.

[10] P. Ramadge and W. Wonham. The control of discrete event systems. *Proceedings of the IEEE*, 77(1):81–98, Jan 1989.

[11] 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*, pages 83–92, New York, NY, USA, 2006. ACM Press.

[12] W. M. Wonham. Supervisory control of discrete-event systems. ece 1636f/1637s 2006-07. course notes, departement of Electrical and Computer Engineering, Univeristy of Toronto, 2006.