This regular paper was presented as part of the main technical program at IEEE ETFA'2011

Process Tracking by Equivalent States in Modal Supervisory Control

Gregory Faraut, Laurent Piétrac and Eric Niel
INSA-Lyon, Lab. AMPERE, University of Lyon
20 av. albert Einstein, 69100 Villeurbanne, France
name.lastname@insa-lyon.fr

Abstract

This paper proposes an extension of the process tracking in modal supervisory control that takes into account the models of the controlled processes which are not trimmed. Indeed, in Supervisory Control Theory (SCT), when a process is not controllable to respect the specifications, the controlled process is computed by the algorithm of the supremal controllable sublanguage. However, the final step of the algorithm of the supremal controllable sublanguage computes the trim of the model. This is coherent to remove states that are neither accessible nor co-accessible because the process is a unique model. Nevertheless, in the case where the design is done with many models, like in modal approach, some inaccessible states in a mode can be accessible from another one. The proposed framework identifies the accessible states, called equivalent states, by other models. This identification uses the name of states to determine their equivalents among models of modes. The aim is to improve switching modes, not only by using languages, but also the name of states in the automata.

1. Introduction

Initiated by Ramadge and Wonham [7, 8], the Supervisory Control Theory (SCT) encompasses significatively the design of Discrete Event systems (DES). Properties such as controllability, observability, etc., previously reserved to continuous systems, are now accessible for DES. Furthermore, the SCT is based on the Language Theory, this implies that it belongs to formal methods. Then, it is possible to formally prove that the process under control, designed by SCT, respects the requirements. Even in the case where the system is difficult to control, in relation to many uncontrollable events, the SCT, by the computation of the supremal controllable sublanguage [10], ensures that the controller is the most permissive language that respects the specification. When the most permissive language is empty, the worst case, the SCT thus formally proves no solution exists. So, specifications have to be modified.

However, the design of real control application implies very large models, with two main problems. The first concerns the state-space explosion: real system models may be too large to be computed. The second concerns the interpretation of the models: large models are difficult to understand even if computation is successful. To solve the scalability problems, several approaches have been proposed, mainly in decomposing the model of SCT. If these approaches are suitable to reduce complexity of the models, they have to verify more other properties to keep ensuring the SCT, with these approaches, computes the most permissive solution which respects the requirement. These properties are, for example, the property of nonblocking between supervisors for the modular approach [11, 4] or the property of consistence for the hierarchical approach [12, 6].

These approaches reduce the complexity, but do not improve the understanding. Indeed, they usually handle the whole process and the whole specification. In our previous work [2, 3], we propose a framework, based on modal approach in DES, that is able to take several modes into account, with one model per mode and smaller models than in centralized approach, and formally verify the switching between models are safety. This last property on switching is verify by the properties of incompatibility and is computed in the process tracking step. Nevertheless, even if the modal approach improves the understanding, the verification to ensure the controller is the most permissive solution is still difficult. Indeed, all these approaches decompose the computation whereas the centralized approach, the classic one in SCT, computes the solution through one unique model of process of the system. In centralized approach, the whole behavior of the system is represented by the process and all trajectories are generated. In others approaches, some trajectories, existing in centralized approach, are removed by the computation of the supremal controllable sublanguage because they are inaccessible in the model of the considered mode. But in other modes, they may exist another compatible state which is accessible. This has never been taken into account in approaches using decomposition. In this paper, we propose an extension of the process tracking step that is able to detect the admissible switching, regards to requirement, that should been removed by the computation of the supremal controllable sublanguage. To realize this,

978-1-4577-0018-7/11/$26.00

we propose to work on automata which are not trimmed. Then, some inaccessible trajectories are still represented, and the process tracking step identifies the states that are equivalent among models. The final aim is to increase the number of switchings representing in modal approach to have as much as the centralized approach, in still using smaller numerous models. In the section II, the basic notions of SCT are recalled. The process tracking is a step of a global framework; this one is presented in section III. In section IV, we present the properties newly included in the process tracking to identify equivalent states. In the last section, an short example is illustrated.

2. Supervisory Control Theory

The Supervisory Control Theory, proposed by Ramdage and Wonham and based on Language Theory [7, 8], separates models in relation to the meaning. The model of the process, called G, represents the system without restriction. The behavior represented by this model may evolve in all possible trajectories, even if these are dangerous or undesired.

Formally, the automaton $G = (Q, \Sigma, \delta, q_0, Q_m)$ models the uncontrolled process, with Q the finite state set, Σ the finite alphabet of symbols (event labels), $\delta : Q \times \Sigma \to Q$ the partial transition function, q_0 the initial state and $Q_m \subseteq Q$ the set of marked states. States exist for periods of time (duration), whereas events occur instantaneously, asynchronously and at virtually random (unpredictable) times.

Let Σ^* be the set of all finite sequences, or strings, of events in Σ, including the empty string ε. The function δ is extended to $\delta : Q \times \Sigma^* \to Q$. Any subset of Σ^* is called a language over Σ. The languages associated to G are the closed behavior $L(G) = \{s \in \Sigma^* | \delta(q_0, s) \text{ is defined}\}$ and the marked behavior $L_m(G) = \{s \in \Sigma^* \mid \delta(q_0, s) \in Q_m\}$. $L(G)$ represents the set of all possible trajectories, i.e. all possible system behaviors, whereas $L_m(G)$ represents the subset of trajectories leading to a marked state.

Let us assume that automaton G models the uncontrolled behavior of the process. This behavior is not satisfactory and must be restricted to a subset of $L(G)$ [1]. Let the requirements designed by the model of specifications and represented by an automaton $E = (X, \Sigma, \xi, x_0, X_m)$ with X the state set, Σ the same alphabet as in G, x_0 the initial state and X_m the set of marked states. This specification models the liveness and safety requirements of the process. The objective is to adjoin a supervisor, denoted by S, to interact with G. To do this, the alphabet Σ is partitioned into two disjoint subsets Σ_c and Σ_{uc} which comprise controllable and uncontrollable events respectively. The controllable events are the events that can be prevented from happening by supervisor S, the uncontrollable events cannot. Formally, the supervisor S is a function from the language generated by G to the power set (the set of all subsets) of Σ: $S : L(G) \to 2^{\Sigma}$. Finally, our goal is to compute the last model which is represented by the controlled process, modeled by an automaton $H = (Y, \Sigma, \tau, y_0, Y_m)$, such that:

- The marked language of H is included in that of G: $L_m(H) \subseteq L_m(G)$.
- This controlled process satisfies the specification: $L_m(H) \subseteq L_m(G \times E)$.
- This controlled process is controllable, i.e. a supervisor S such that $L(S/G) = L(H)$ exists.

If $L(G \times E)$ is controllable with respect to $L(G)$, then $H = G \times E$. Else, compute the automaton that generate the largest controllable sublanguage of $L(G \times E)$, called "supremal controllable sublanguage". If this automaton exists, the controlled process is non blocking and minimally restrictive.

2.1. Modal Supervisory Control

In this paper, we consider our previous work where the system is designed by modal approach [2, 3]. Indeed, a system is composed by numerous components (plants, actuators, sensors, etc.) and has to respect several requirements. Then, the system has to use a part of the set of components to achieve tasks and has to respect a part of the set of requirements. This is considered as a "mode" of the system, and a system has several modes among which it may switch. The behavior of a mode represents the temporary behavior of the system until another mode is activated. The goal of the modal approach is to compute the controlled process of each mode such that the sequence of the activation of modes represents the constant behavior of the system and is equivalent to the behavior computed by centralized approach.

2.2. Framework of the modal approach

The framework of this approach is illustrated by Fig.1. The first step formalizes models (models of components, models of requirements, models of modes, etc.). The second step studies, with the SCT, the internal behavior of each mode, independently of others. The aim is to certify the requirements in each mode are respected independently of the switch requirements of this mode or the requirements of others modes. This step is not further discussed on this paper. The third step, called intermodal, studies the behavior that includes the previous internal behavior and the switching behavior. The goal of this step is to ensure the system can switch between modes in keeping respecting requirements between modes. The computed models formally respect the requirements. However, the SCT does not ensure the deactivation of a mode leads to the activation of another one. This is the aim of the fourth step, the process tracking. Finally, a merging step is computed at the end of the framework to reduce the size of models.

In this paper, we mainly focus on the third and fourth steps. Nevertheless, we need to clearly formalize the models that are manipulated. The first definition concerns the components and their models:

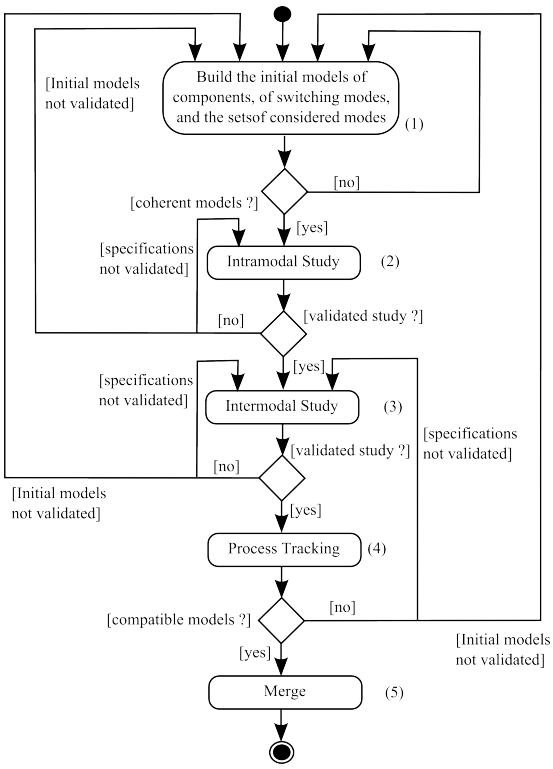

Figure 1: Framework of modal approach

Definition 1
The set of components is denoted by $\mathcal{C} = \{C_1, C_2, \ldots, C_i\}$, where $i \in \mathbb{N}$ and $i \geq 1$. A component C_i is modeled by an automaton G^{C_i} where $G^{C_i} = (Q^{C_i}, \Sigma^{C_i}, \delta^{C_i}, q_0^{C_i}, Q_m^{C_i})$, with:

- *Q^{C_i}, $Q_m^{C_i}$ and $q_0^{C_i}$ respectively are the set of states, the set of marked states and the initial state of the component C_i;*
- *Σ^{C_i} is the event set of the component C_i, including four subsets:*
  - *$\Sigma^{C_i} = \Sigma_c^{C_i} \cup \Sigma_{uc}^{C_i}$ with $\Sigma_c^{C_i} \cap \Sigma_{uc}^{C_i} = \emptyset$. Σ_c and Σ_{uc} are respectively the disjoint sets of controllable and uncontrollable events generated by the component C_i;*
  - *$\Sigma^{C_i} = \Sigma_{\rightleftarrows}^{C_i} \cup \Sigma_{\circlearrowleft}^{C_i}$ with $\Sigma_{\rightleftarrows}^{C_i} \cap \Sigma_{\circlearrowleft}^{C_i} = \emptyset$. $\Sigma_{\rightleftarrows}^{C_i}$ is the set of switch events generated by C_i. $\Sigma_{\circlearrowleft}^{C_i}$ are all other events generated by the components.*
- *δ^{C_i} is the transition function and includes $\delta_{\rightleftarrows}^{C_i}$ which represents the set of switch transitions. A switch transition is a transition such as a switch event is included in $\Sigma_{\rightleftarrows}^{C_i}$.* ♦

The second definition concerns the requirements.

Definition 2
The set of requirements is denoted by $\mathcal{R} = \{R_1, R_2, \ldots, R_l\}$, where $l \in \mathbb{N}$ and $l \geq 1$. A requirement R_l, is modeled by a specification E^{R_l} represented by an automaton such as $E^{R_l} = (X^{R_l}, \Sigma^{R_l}, \xi^{R_l}, x_0^{R_l}, X_m^{R_l})$, with:

- *X^{R_l}, $X_m^{R_l}$ and $x_0^{R_l}$ are respectively the set of states, the set of marked states and the initial state of the specification E^{R_l};*
- *Σ^{R_l} is the events set of the specification E^{R_l};*
- *ξ^{R_l} is the transition function.* ♦

The next definitions concern the set of modes and the modes of the system.

Definition 3
The set of 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). ♦

Definition 4 (A mode)
Let $\mathcal{C}^{M_j} \subset \mathcal{C}$ be the set of components used in the mode M_j and let $\mathcal{R}^{M_j} \subset \mathcal{R}$ be the set of requirements which have to be respected in the mode M_j, then a mode represents the temporary functioning of the system such that it only uses for a time a subset of the components and has to respect a subset of the requirements. Then, a mode M_j is defined by a tuple such that: $M_j = (\mathcal{C}^{M_j}, \mathcal{R}^{M_j})$. ♦

2.3. The controlled processes

With the previous definitions, the step 3 of the framework, illustrated by Fig.1, computes the models H^{M_j} representing the controlled process in the mode M_j. Formally, this computation, based on SCT, is defined as follows:

Definition 5 (Controlled process H^{M_j})
$H^{M_j} = (Y^{M_j}, \Sigma^{M_j}, \tau^{M_j}, y_0^{M_j}, Y_m^{M_j})$ *with:*
$L_m(H^{M_j}) = [L_m(G^{M_j} \times E^{M_j})]^{\uparrow c}$ ♦

Two operations are recurrent in this computation. The first one is the parallel composition of automata. This parallel composition is defined as follows:

Definition 6 (Parallel composition)
Let G_1 and G_2 be two automata such that $G_1 = (Q_1, \Sigma_1, \delta_1, q_{0,1}, Q_{m,1})$ and $G_2 = (Q_2, \Sigma_2, \delta_2, q_{0,2}, Q_{m,2})$, the automaton $G = (Q, \Sigma, \delta, q_0, Q_m)$ is computed by parallel composition of G_1 and G_2 such that:
$G = G_1 || G_2 = Ac(Q_1 \times Q_2, \Sigma_1 \cup \Sigma_2, \delta, (q_{0,1}, q_{0,2}), Q_{m,1} \times Q_{m,2})$ *with:*

- $Q = Q_1 \times Q_2$;
- $\Sigma = \Sigma_1 \cup \Sigma_2$;

- $\delta((x_1,x_2),\sigma) =$

$$\begin{cases} (\delta_1(x_1,\sigma),\delta_2(x_2,\sigma)) & if\ \delta_1(x_1,\sigma)!\ and\ \delta_2(x_2,\sigma)! \\ (\delta_1(x_1,\sigma),x_2) & if\ \delta_1(x_1,\sigma)!\ and\ \sigma \notin \Sigma_2 \\ (x_1,\delta_2(x_2,\sigma)) & if\ \delta_2(x_2,\sigma)!\ and\ \sigma \notin \Sigma_1 \\ undefined & otherwise \end{cases}$$

- $q_0 = (q_{0,1}, q_{0,2})$;
- $Q_m = Q_{m,1} \times Q_{m,2}$ ♦

The second operator is the algorithm of the supremal controllable sublanguage, noticed by $\uparrow c$. Two algorithms exist to compute the supremal controllable sublanguage [10, 5]. In these two algorithms, the final step computes the trim of the automaton [1]. For recall, the function trim removes all states that are neither accessible nor co-accessible (i.e. states such that marked states are not accessible from them). The process tracking is operated on models H^{M_j} computed by this second operator.

2.4. Process Tracking

The models H^{M_j} respect the internal and switching specifications. This is ensured by SCT. However, due to the decomposition on several models - one by mode - the switch from one mode to another one is not sure. A mode can be deactivated by the occurrence of switch event, but not other mode may be activated by the same occurrence of this event. The aim of the process tracking is to ensure that each switch event which deactivates a mode corresponds to a switch event in another mode which activates this one. If a switch event deactivates a mode without activating another one, then a dangerous behavior about switching exists and has to be forbidden by a new requirement. This kind of trajectories, leading to a switch event which does not activate a mode, are defined as "incompatible". The process tracking is able to characterize trajectories between modes. Another kind of trajectories exists; they concern switch events which may activate a mode in different states. This is due to the property of the projection function when the behavior is tracked between modes. This case means information is missed about switching and need to be added in the models to ensure a suitable switching between modes. These trajectories are characterized as "inconsistence". This characterization, allowed by the procedure of process tracking, is a helpful tool for the designer. Then, it only concenters on the characterized problematic trajectories. To be able to characterize the trajectories, about compatibility and consistency, the process tracking is based on several definitions. In the next, we consider the system in the mode M_j and the switch event α that may occur to lead the system in the mode M_k.

The first definition concerns the sets of switch events in modes

Definition 7 (Switch events sets)
$\Sigma^{M_j}_{\leftrightarrows} \subset \Sigma^{M_j}$ *is the switch events set of the mode* M_j *such as:* $\Sigma^{M_j}_{\leftrightarrows} = \bigcup_{n \in \mathcal{C}^{M_j}_{\leftrightarrows}} \Sigma^{\mathcal{C}_n}_{\leftrightarrows}$.
$\Sigma^{M_j}_{\leftrightarrows} = \Sigma^{M_j}_{\leftarrow} \cup \Sigma^{M_j}_{\rightarrow}$ *where* $\Sigma^{M_j}_{\leftarrow}$ *(resp.* $\Sigma^{M_j}_{\rightarrow}$*) is the set of events which activate (resp. deactivate) the mode* M_j*. They are defined as follows:*

- $\Sigma^{M_j}_{\rightarrow} = \{\alpha \in \Sigma^{M_j}_{\leftrightarrows} | M_k \in Q^{\mathcal{M}}, \delta^{\mathcal{M}}(M_j,\alpha) = M_k \text{ is defined}\}$;
- $\Sigma^{M_j}_{\leftarrow} = \{\alpha \in \Sigma^{M_j}_{\leftrightarrows} | M_k \in Q^{\mathcal{M}}, \delta^{\mathcal{M}}(M_k,\alpha) = M_j \text{ is defined}\}$; ♦

From this definition, we are able to define the set of states where a switch event is generated in modes.

Definition 8 (switch states)
$Y^{M_j}_{M_j \xrightarrow{\alpha} M_k} = \{y \in Y^{M_j} \mid M_j, M_k \in Q^{\mathcal{M}},\ \alpha \in \Sigma^{M_i}_{\rightarrow} \cap \Sigma^{M_k}_{\leftarrow},\ \tau^{M_j}(y,\alpha) \text{ is defined}\}$ ♦

The next definition defines the language which leads from the initial state to the states where a switching is possible:

Definition 9 (Switch language)
$L^{y}_{M_j \xrightarrow{\alpha} M_k}(H^{M_j}) = \{s \in \Sigma^{M_j *} \mid y \in Y^{M_j}_{M_j \xrightarrow{\alpha} M_k},\ \tau(y_0,s) = y \text{ is defined}\}$ ♦

This language represents all trajectories in the mode M_j which lead to a state "y" where a switching event may happen. This language is next projected to determine the equivalent language in the mode M_k.

Definition 10 (Projection function)
Let $P_{M_j,M_k} : \Sigma^{M_j *} \rightarrow \Sigma^{M_k *}$ *such as* $\forall \sigma \in \Sigma^{M_j}$ *and* $\forall s \in \Sigma^{M_j *}$*:*

$$P_{M_j,M_k}(\varepsilon) = \varepsilon$$
$$P_{M_j,M_k}(s\sigma) = \begin{cases} P_{M_j,M_k}(s)\sigma \ if\ \sigma \in \Sigma^{M_j} \cap \Sigma^{M_k} \\ P_{M_j,M_k}(s) \quad if\ \sigma \in \Sigma^{M_j} \backslash \Sigma^{M_k} \end{cases}$$

In words, this function takes a language defined on alphabet Σ^{M_j}, and *erases* the events that are not included on alphabet Σ^{M_k}.

Definition 11 (Equivalent language)
Let a system be in the mode M_j *and it switches into the mode* M_k*,* $L^{y}_{M_j \xrightarrow{\alpha} M_k}(H^{M_j})$ *is the sublanguage of* H^{M_j} *leading to the state y where a switch event* α *could be generated.* $L^{y}_{M_j \xrightarrow{\alpha} M_k}(H^{M_k})$ *is the language projected on the alphabet* Σ^{M_k}*, such as:* $L^{y}_{M_j \xrightarrow{\alpha} M_k}(H^{M_k}) = P_{M_j,M_k}[L^{y}_{M_j \xrightarrow{\alpha} M_k}(H^{M_j})]$ ♦

With this equivalent language, we are able to determine if the trajectory is compatible and/or consistent. A trajectory is compatible if the equivalent language exists in the mode M_k.

Definition 12 (Compatible language)
A switch trajectory in H^{M_k} is compatible with a switch trajectory in H^{M_j} iff: $\forall y \in Y^{M_j}_{M_j \overset{\alpha}{\to} M_k}$, $(L^{y}_{M_j \overset{\alpha}{\to} M_k}(H^{M_k}) \subseteq L(H^{M_k}))$ ♦

In words, two controlled processes are compatible if all switching between them are compatible.

A trajectory is consistent if for each switch event in the mode M_j correspond only one switch event in the mode M_k

Definition 13 (Consistent language)
Let a trajectory be compatible in both modes. Then, It is consistent iff: $\forall y_1, y_2 \in Y^{M_j}_{M_j \overset{\alpha}{\to} M_k}$,
$(y_1 \neq y_2 \Leftrightarrow L^{y_1}_{M_j \overset{\alpha}{\to} M_k}(H^{M_k}) \cap L^{y_2}_{M_j \overset{\alpha}{\to} M_k}(H^{M_k}) = \emptyset)$ ♦

The trajectories which are consistent are compatible. But compatible trajectories does not imply they are consistent. When compatible but non-consistent trajectories are detected, the designer has to add new requirements to remove them. In this paper, the proposition solves incompatible trajectories without removing them by new requirement.

3. Procedure and Definitions

We propose in this paper to determine equivalent states between modes for incompatible trajectories. Indeed, in our previous work, incompatible trajectories and inconsistent trajectories have to be removed by new requirement. However, concerning incompatible trajectories, we understand they are incompatible due to the specifications of modes. A behavior allowed in the mode M_j leading to a switch event is forbidden in the mode M_k by the property of controllability. Then, the behavior leading to the state where a switch event happens, in the mode M_k is simply removed because this state is not accessible in the model of the controlled process H^{M_j}. But the states which are not accessible in the mode M_k could be accessible by the mode M_j. The incapacity to determine the accessibility between models of modes is the main difficulty when a system is designed by decomposition like modal, decentralized or modular approaches. If the state in M_k is not accessible by language, and thus is incompatible, it could be equivalent to a state in another mode M_j. In order to determine this equivalence, we propose to work on the name of state, instead of language, and use automata that are not trimmed. Then, we do not compute the final step of the algorithms of the supremal controllable sublanguage, and do not remove inaccessible states.

3.1. Notions of sequence

Previously, we had remark the name of states was built as a sequence of elements of some sets. To formalize equivalent states, we need to formalize the operations to manipulate the sequences. In particular, how is it built the name of the state and how to compare two sequences to establish an equivalence. For that, we use basic notions of sequence found in [9]. A sequence of elements is a list of these elements in some order. We usually designate a sequence by writing the list within parentheses. For example, the sequence of elements from the set $\{q_1, q_2, q_3\}$ would be written : (q_3, q_1, q_2, q_1). In a set the order does not matter, but in a sequence it does. Similarly, repetition does matter in sequence, but it does not in a set. Two sequences may be concatenate: $(q_1, q_2) \cdot (q_3, q_4) = (q_1, q_2, q_3, q_4)$. Furthermore, a sequence of sequences results a sequence: $((q_1, q_2), (q_3, q_4), (q_5)) = (q_1, q_2, q_3, q_4, q_5)$. As said in the previous remark, the product of two set Q_1 and Q_2, written $Q_1 \times Q_2$, is the set of all pairs wherein the first element is a member of Q_1 and the second element is a member of Q_2: $Q_1 \times Q_2 = \{(q_{10}, q_{20}), (q_{11}, q_{20}), \ldots, (q_i, q_j)\}$ with $q_i \in Q_1$ and $q_j \in Q_2$. The function "ran" allows to transform a sequence into set : $ran(q_1, q_2, q_3) = \{q_1, q_2, q_3\}$. At last, but not least, the function $\restriction$ allows to "filter" a sequence by a set to remove from the sequence the element that does not belong to the set: $(q_1, q_2, q_3, q_4) \restriction \{q_2, q_4, q_5\} = (q_2, q_4)$ The result also is a sequence, sub-sequence from the first one. With these last notions, we are able to compute formally the name of states of all automata in the modal approach in relation to the several components and the requirements.

3.2. Definition of equivalent states

The studied states are those which are inaccessible in the mode M_k. This non accessibility is identified by the trajectories leading to these states as incompatible trajectories. With name of states standpoint, we consider two states - each one in different mode - are equivalent if the common components in both modes are in the same state and if the common requirements are the same. As we consider the name of states is built from the name of each component and each requirement in mode, in order to establish the equivalence between two states we only consider common part of the name, those of common components and requirements. Formally, we define the equivalence as follows:

Definition 14 (Equivalent states)
Let H^M_j and H^M_k be the controlled processes resp. of the mode M_j and the mode M_k, Let y_1 and y_2 be states where a switch event may occur, let $\mathcal{C}^{M_{jk}}$ be the set of components which are common between both modes M_j and M_k, and let $\mathcal{R}^{M_{jk}}$ be the set of requirements which have to be

respected in both modes M_j and M_k, the states y_1 and y_2 are equivalent if:

$$\forall y_1 \in Y^{M_j}_{M_j \xrightarrow{\alpha} M_k} \text{ and } y_2 \in Y^{M_k}_{M_j \xrightarrow{\alpha} M_k},$$
$$\mathcal{C}^{M_{jk}} = \mathcal{C}^{M_j} \cap \mathcal{C}^{M_k} \text{ and } \mathcal{R}^{M_{jk}} = \mathcal{R}^{M_j} \cap \mathcal{R}^{M_k},$$
$$ran(y_1 \upharpoonright (Q_{eq} \cup X_{eq})) = ran(y_2 \upharpoonright (Q_{eq} \cup X_{eq}))$$

with: $Q_{eq} = \bigcup_{C_i \in \mathcal{C}^{M_{jk}}} Q^{C_i}$ *and* $X_{eq} = \bigcup_{R_l \in \mathcal{R}^{M_{jk}}} X^{R_l}$ ♦

3.3. Procedure of process tracking

With all the definitions of the section *II*, we are able to propose the procedure of process tracking which characterize switching in using trajectories or also name of states.

Procedure 1

For each switch event, in H^{M_j}, implying an deactivation of the mode M_j, and leading to the activation of the mode M_k by the occurrence of the switch events:

1. *for each $y \in Y^{M_j}_{M_j \xrightarrow{\alpha} M_k}$ [def. 8]:*
   - (a) *we compute $L^{y}_{M_j \xrightarrow{\alpha} M_k}(H^{M_j})$ [def. 9];*
   - (b) *we compute $L^{y}_{M_j \xrightarrow{\alpha} M_k}(H^{M_k})$ [def. 11].*
   - (c) *the property of compatibility is verified [def. 12]:*
     - i. *For each $(L^{y}_{M_j \xrightarrow{\alpha} M_k}(H^{M_k}) \subseteq L(H^{M_k}))$: the property of consistency is verified [def. 13]:*
       - A. $\exists y_1, y_2 \in Y^{M_k}$ $[L^{y}_{M_j \xrightarrow{\alpha} M_k}(H^{M_j}) \subseteq L^{y_1}(H^{M_k})$ *and* $L^{y}_{M_j \xrightarrow{\alpha} M_k}(H^{M_j}) \subseteq L^{y_2}(H^{M_k})]$ *or* $\exists y' \in Y^{M_j}_{M_j \xrightarrow{\alpha} M_k}$ $(y \neq y' \Leftrightarrow L^{y}_{M_j \xrightarrow{\alpha} M_k}(H^{M_k}) \cap L^{y'}_{M_j \xrightarrow{\alpha} M_k}(H^{M_k}) \neq \emptyset)$: *$H^{M_k}$ is not consistent and the procedure stops here;*
       - B. *if not, the switch event is then considered as consistent. We add a subscript on the switch event in the transition function such as $\tau^{M_j}(y,\alpha)$ and $\tau^{M_k}(y',\alpha)$ are modified by $\tau^{M_j}(y,\alpha_l)$ and $\tau^{M_k}(y',\alpha_l)$ with l the subscript. The new alphabet of this step is defined by: $\Sigma^{M_j}_{lab} = \Sigma^{M_j} \cup \{\alpha_l\}$;*
     - ii. *For each $(L^{y}_{M_j \xrightarrow{\alpha} M_k}(H^{M_k}) \not\subseteq L(H^{M_k}))$: the equivalent states are verified [def. 14]:*
       - A. *For each state where a switch event is generated in H^{M_j} result a set of equivalent states in H^{M_k}. Whatever the number of element, the designer has to choose the best candidate to be the final equivalent state. For all the identified couples, we add a subscript as previously.*
       - B. *In the case where no equivalent state could be determined, the procedure stops here. No solution was found, the designer has to add requirement.*
   - (d) *After the last $y \in Y^{M_j}_{M_j \xrightarrow{\alpha} M_k}$, if the procedure does not stop for inconsistency or incompatibility, then the models H^{M_k} and H^{M_j} are compatible with regard to the switch event α. We can proceed the procedure for the next switch event.*
2. *The automata resulted is non-trimmed. All equivalent states were found, we compute the trim now to removed the useless states, those which are not accessible from other modes.*
   - (a) *$\forall y \in Y^{M_j}_{M_j \xrightarrow{\alpha} M_k}$ are considered as initial states;*
   - (b) *we compute the trim of the automaton;*
   - (c) *we do not consider yet the states y such as $\forall y \in Y^{M_j}_{M_j \xrightarrow{\alpha} M_k}$ as initial states, except for the first initial state $y_0^{M_j}$* ♦

The last step of this procedure manipulates the automaton to compute the trim without removing the accessible states from another mode. It is the reason why we cannot apply simply the trim function.

The final models are automata such as properties are respected, but the models are still possibly inaccessible if the procedure identify equivalent states. This is the goal of the final step of the framework, the merge function which merges all states that are not significant.

4 Example

In this section, an example is presented to illustrate the automata, representing the behavior of each mode, which are not trimmed, the trajectories which are not accessible, and the equivalent states between modes.

4.1 System and requirements

Consider the manufacturing system illustrated in Fig.2(a), the system comprises four components and one buffer. The components are used to process a part and the buffer is used as a storage among the components with a maximal capacity of 1. The component C_i are modeled by the automata denoted G^{C_i} and are shown Fig.2(b) and Fig.2(c). The event s_i and e_i represent a new task and the end of the task, respectively. While all these events are observable, events S_i and r_1 are controllable and e_i and f_1 are not. The system has two modes, such as $\mathcal{M} = \{N, D\}$ and such as N is the nominal mode and D the degraded

mode. In the mode N, the system only uses to produce the components C_1, C_2 and C_4. However, the component C_1 may fail, represented by the event f_1. In this case, the system switches into the mode D where it uses the components C_2 and C_3. Then, the component C_3 replaces the C_1 when this last one fail, and the system does not use the component C_4. The requirement of the activation of the components C_1 and C_3, regards to the considered mode of the system, is illustrated in Fig.2(d). Fig.2(e) represents the requirement about the maximal capacity of the buffer.

In this example, we only focus on the incompatible trajectories. Then, the requirements illustrated on Fig.2(f) and Fig.2(g) represent the deactivation of uncommon components between modes. Indeed, if the components C_3 or C_4 are working when a switch event happen, some trajectories are identified as inconsistent.

4.2 Process Tracking

From these models and SCT, the controlled processes are computed w.r.t the Def.5. However, in order to study the inaccessible states which could be accessible from another mode, we do not compute the final step of the algorithm of the supremal controllable sublanguage. Then, the models of the controlled processes are not trimmed. Fig.3 illustrates the automata of controlled processes with Fig.3(a) for H^N and Fig.3(b) for H^D. The part in dotted-lines represents the inaccessible states. This part will be removed with the classic computation of supremal controllable. However, in Fig.3(b), a trajectory leads into the states $(F_1, A_2, A_3, D, B_1, C_{3id})$ and $(F_1, M_2, A_3, D, B_1, C_{3id})$ where switch event r_1 may happen. Due to the projection function, this trajectory is incompatible in H^N represented by Fig.3(a). Indeed, the states should be removed to respect the requirement of buffer because to access to the state where C_1 is in fault state F_1, it should be in state M_1 before. Nevertheless, the buffer is already in B_1 state, forbidding the component C_1 to work.

The Def.14 is used in order to identify, from the incompatible states in H^D, the possible compatible states in H^N. Then, we have: $(F_1, A_2, A_3, D, B_1, C_{3id}) \upharpoonright (Q^{C_1} \cup Q^{C_2} \cup X^{R_{buf}} \cup X^{R_{mode}}) = (F_1, A_2, D, B_1)$ which is equivalent to the state in H^N : $(F_1, A_2, A_4, D, B_1, C_{4id}) \upharpoonright (Q^{C_1} \cup Q^{C_2} \cup X^{R_{buf}} \cup X^{R_{mode}}) = (F_1, A_2, D, B_1)$.

These two states are equivalents and, even if their trajectories are incompatible, removed by supremal controllable, the switching is allowed from these states by the switch event r_1. Then, thanks to equivalent states, we have identify two more trajectories, illustrated by the dashed lines in Fig.3(a) to switch between modes than only using the equivalent language.

5. Conclusion

We discussed about the difficulty, when models are decomposed in SCT to design the controlled process, to identify the switching between modes. In particular when the supremal controllable sublanguage is computed in order to solve a controllability problem. In this paper, we propose a procedure to identify equivalent states between modes such that the states in the activated mode are inaccessible in the mode but which are accessible for another mode. In order to realize this, we do not only consider the language but also the name of states, built by composition of components used and requirements respected in the modes. Included in a framework, this procedure allows increasing the number of switching between modes and reaching the same behavior in the centralized approach where the model of the process is not decomposed. Further works lead to significant improvement in expression of requirement by name of states instead of language. Indeed, some requirements could be difficult to design by language, and be much easier to design by name of states. In particular, the requirements may be expressed by some combination of name of states, which is difficult to express by language. Furthermore, each extension by the study of the name of states will need a clear formal definition to keep the properties of the SCT.

References

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

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

[3] G. Faraut, L. Pietrac, and E. Niel. Formal approach to multimodal control design: Application to mode switching. *Industrial Informatics, IEEE Transactions on*, 5(4):443–453, Nov. 2009.

[4] J. Komenda, J. van Schuppen, B. Gaudin, and H. Marchand. Supervisory control of modular systems with global specification languages. *Automatica*, 44(4):1127–1134, Apr. 2008.

[5] R. Kumar, V. K. Garg, and S. I. Marcus. On controllability and normality of discrete event dynamical systems. *Systems and Control Letters*, 17(3):157–168, 1991.

[6] R. Leduc, M. Lawford, and W. Wonham. Hierarchical interface-based supervisory control-part ii: parallel case. *Automatic Control, IEEE Transactions on*, 50(9):1336–1348, Sept. 2005.

[7] P. J. Ramadge and W. M. Wonham. Supervisory control of a class of discrete event processes. *SIAM J. Control Optim.*, 25(1):206–230, 1987.

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

[9] M. Sipser. *Introduction to the Theory of Computation, Second Edition*. Course Technology, February 2005.

[10] W. M. Wonham and P. J. Ramadge. On the supremal controllable sublanguage of a given language. *SIAM Journal of Control and Optimization*, 25(3):637–659, 1987.

[11] W. M. Wonham and P. J. Ramadge. Modular supervisory control of discrete event systems. *Mathematics of Control, Signals and Systems*, 1(1):13–30, 1988.

[12] H. Zhong and W. Wonham. On the consistency of hierarchical supervision in discrete-event systems. *Automatic Control, IEEE Transactions on*, 35(10):1125–1134, Oct 1990.

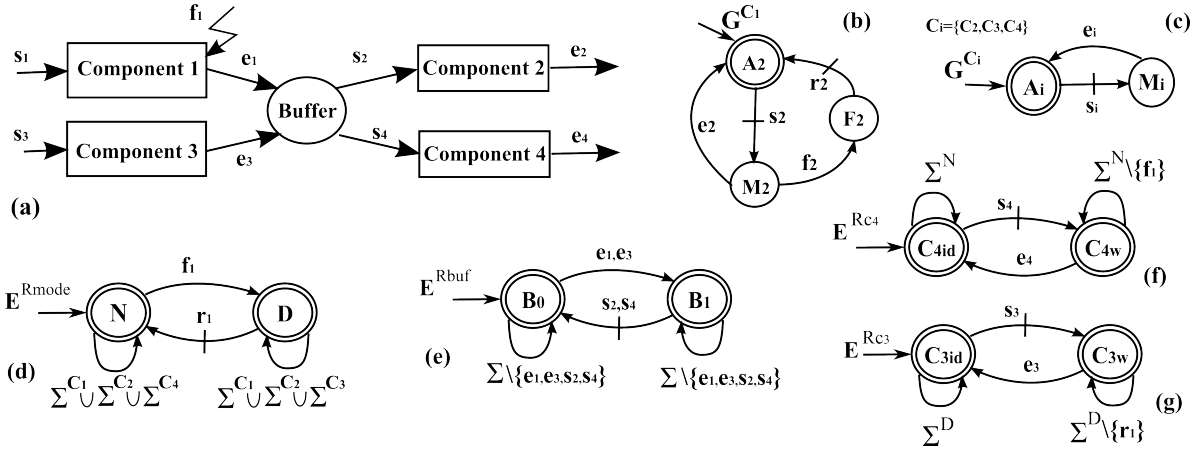

Figure 2: Manufacturing system example : (a) the studied system; (b,c) process of components C_i; (d) model of the specification of the mode; (e) model of the specification of the buffer; (f) model of the specification about C_4; (g) model of the specification about C_3

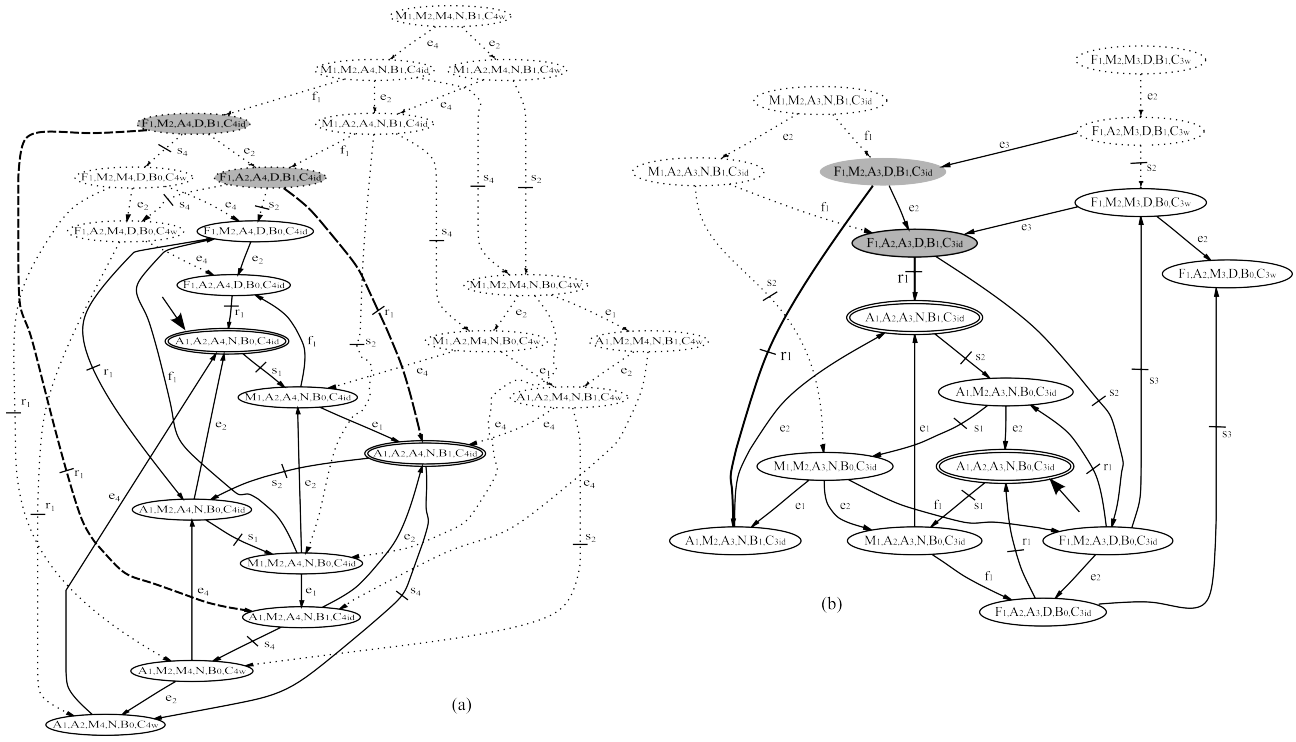

Figure 3: (a) Controlled process H^N;(b) Controlled process H^D. The dotted-lines represent the inaccessible states that should be removed by supremal controllable. The grey states are the compatible states allowing to switch from the degraded mode H^D into the nominal mode H^N when a switch event r_1 happen. The dashed lines represents these switch events.