Merging scenarios

Jacques Klein [1] Benoit Caillaud [2] Loïc Hélouët [3]

IRISA/INRIA, Campus de Beaulieu, 35042 Rennes Cedex, France

Abstract

This paper proposes a merge operator for behavioral requirements expressed by Message Sequence Charts and shows how this product can be systematically used to integrate new behaviors in an existing one. First the merge operator is defined as a fibered product of scenario descriptions. This product is then used to integrate a consensus mechanism to solve the non-local choice problem.

Key words: Scenarios, distributed systems, modeling, composition.

1 Introduction

Scenario languages define typical executions of systems. They are used to represent output traces, but also to capture requirements of distributed systems. Even if several dialects exist, all scenario languages are based on a similar idea: they depict systems runs as compositions of partially ordered sets of events. A drawback of scenarios is that the number of participants in a given interaction cannot be parameterized. With a fixed number of objects in an interaction, it is hard to describe behaviors of systems with dynamic architecture. Message Sequence Charts propose instance creation, but this mechanism needs to know a priori the name of instances that will be created.

Another drawback of scenarios is that a description of a system is usually defined by a set of scenarios, which represent typical uses of the system under design in a given situation, but comport some redundancies. The intended behavior of a system can be seen as a combination of all these views. So far, no satisfactory merge operator exists. Scenario languages are all equipped with alternative, sequential, or parallel composition, but these operators do not capture the notion of redundancy that may exist between the composed views.

A solution proposed is to gather coherently redundant views given as Live Sequence charts [8]. The main idea is to combine scenarios at runtime. LSCs are executed in parallel, and events that can be executed in several views are synchronized when possible. From an initial set of live scenarios, an event is executed, and new scenarios are "triggered" by this event execution. Two scenarios are declared inconsistent if they contradict each other on the order of common events. The main drawback of this approach is that inconsistency between scenarios may not be discovered if the simulations performed do not pass through a faulty configuration.

This paper addresses a second approach that consists in the construction of a new model, preferably using the same scenario language, which is the smallest model to contain all the

[1] Email: `jacques.klein@irisa.fr` (This work has been partially supported by Families Project, ITEA project ip02009)
[2] Email: `benoit.caillaud@irisa.fr`
[3] Email: `loic.helouet@irisa.fr`

©2004 Published by Elsevier Science B. V.

views composed. Such a model does not always exist, and in some cases it is not unique. However, when common parts in scenarios are clearly identified, this model can be computed. The language chosen is Message Sequence Charts, a scenario language standardized by ITU [11], but our approach can be adapted to any partial order based language. The merge framework proposed uses a fibered product of scenarios. This product first identifies pairs of scenarios that must be "synchronized" in two HMSCs, and realizes their union with an amalgamated sum. We then show the usefulness of this construction on a concrete application, ie, the introduction of a consensus algorithm to localize choices in a HMSC.

The benefits of scenario merging do not only concern view composition. In fact, an endogenous merge operator can be used to propose formal and well founded model transformations and design patterns for scenarios. The paper is organized as follows: Section 2 recalls some basic notions on Message Sequence Charts. Section 3 proposes a definition of amalgamated sum. Section 4 defines the complete fibered product. Section 5 shows an application of fibered product to eliminate non-local choices from HMSCs, and section 6 concludes this work.

2 Message Sequence Charts

Message Sequence Charts (or MSC for short) is a scenario language standardized by ITU [11]. MSCs are composition of very simple chronograms by means of sequence, alternative, and iteration operators. MSCs propose two specification levels. At the lowest level, basic Message Sequence Charts (or bMSCs for short) describe simple communication patterns between entities of the system called *instances*. These chronograms are then composed by several levels of High-level Message Sequence Charts (or HMSC for short), a kind of bMSC automaton.

In a bMSC, instances are represented by a vertical axis. Message exchanges are represented by arrows labeled by message names from the emitting to the receiving instance, and communications are supposed to be asynchronous. A bMSC defines a set of events, which are occurrences of actions in the system (message emissions, receptions, atomic actions or operations on timers), and a precedence relation on these events: a message emission must precede the corresponding reception, and events are totally ordered along instance axis (excepted in specific parts of the axis called coregions). Process algebra semantics for bMSCs have been proposed [16], but it seems rather natural to give bMSCs a non interleaving semantics as in [10] and [12]. In the sequel, our formal definition of bMSC will be based on the notions of preorders and partial orders. A *preorder* on a set of elements E is a relation $R \subseteq E^2$ that is reflexive (ie. $\forall e \in E, eRe$) and transitive (ie. $\forall e, f, g \in E, eRf \wedge fRg \Longrightarrow eRg$). A *partial order* is an antisymmetric preorder (ie. $\forall a, b \in E, aRb \wedge bRa \Longrightarrow a = b$).

As already mentioned, bMSCs define a causal order between message emissions and receptions, and a partial ordering on events situated on the same instance. They can then be formally defined as follows:

Definition 2.1 (Basic Message Sequence Charts) *Let I be a finite set of instances. A* bMSC *over I is a tuple $M = (E, \leq, A, \alpha, \phi, \prec)$, where E is a set of events, $\leq$ is a preorder on E, A is a set of actions, $\alpha : E \longrightarrow A$ maps events to actions, $\phi : E \longrightarrow I$ maps events to instances, $\prec \subseteq E \times E$ is a partial bijection pairing message emissions and receptions, such that $\prec \subseteq \leq$. When no instance set is specified, a bMSC is a tuple $M = (I, E, \leq, A, \alpha, \phi, \prec)$,* *where I is a finite instance set and $M = (E, \leq, A, \alpha, \phi, \prec)$ is a bMSC over I.*

A bMSC M will be called *well-formed* whenever $\leq$ is a partial order relation. For a bMSC $M = (E, \leq, A, \alpha, \phi, \prec)$, we will denote by $Min(M) = \{e \in E | \forall e' \in E, e' \leq e \Rightarrow e' = e\}$ the set of minimal events for the preorder relation, i.e the set of events that have no causal predecessor. Figure 1 shows an example of bMSC, with 3 instances (*sender*, *medium* and *receiver*), that exchange several messages (*data*, *info* and *ack*). The events of this bMSC are the emission and receptions of messages , and an atomic action *action* executed by the *sender*. Note that events e_5 and e_6, that symbolize emissions of messages *info* and *ack* are situated in a coregion, ie no order is imposed between these two events.

bMSCs alone do not have a sufficient expressive power: they can only define finite behaviors, without real alternatives (the only alternatives in the behaviors depicted by a bMSC are due to possible interleavings). For this reason, MSCs have been extended with higher-level constructs, namely HMSCs [18]. Roughly speaking, HMSCs are a kind of transition systems labeled by bMSCs.

Definition 2.2 (Labeled Transition Systems) *A* labeled transition system *(or LTS for short) is a tuple $\mathcal{S} = (S, \widehat{s}, \Sigma, T)$, where S is a set of states, $\widehat{s} \in S$ is the initial state of $\mathcal{S}$, Σ is a finite alphabet, and $T \subseteq S \times \Sigma \times S$ is a set of transitions. In the sequel, we will denote by $\alpha(t) = a$ the* label *of a transition $t = (p, a, q) \in T$. We will write $p \xrightarrow{a} q$ whenever $(p, a, q) \in T$. State p will be called the* origin *of t, denoted $\alpha_-(t)$, and q the* goal *of t, denoted $\alpha_+(t)$.*

Definition 2.3 (High level Message Sequence Charts) *Let I be a finite set of instances. A* HMSC *over I is a tuple $H = (\mathcal{S}, \mathcal{M}, \lambda)$ where: $\mathcal{S} = (S, \widehat{s}, \Sigma, T)$ is a labeled transition system called the* support automaton *of H, $\mathcal{M}$ is a finite set of bMSCs over I, $\lambda : T \to \mathcal{M}$ maps transitions to bMSCs. When no instance set is specified, HMSCs are defined as quadruples $H = (I, \mathcal{S}, \mathcal{M}, \lambda)$, where I is a finite instance set and $(\mathcal{S}, \mathcal{M}, \lambda)$ is a HMSC over I.*

An example of HMSC is given Figure 2. The notion of sequential composition (noted $\bullet$) is central to understand HMSCs. Roughly speaking, sequential composition of two bMSCs consists in gluing both diagrams along their common instance axes. Note that this sequence does only impose precedence on events situated on the same instance, but that events situated on different instance in two bMSCs $M1$ and $M2$ can be concurrent in $M1 \bullet M2$. Sequential composition can be formally defined as follows:

Definition 2.4 (Sequential Composition) *The* sequential composition *of two bMSCs M_1 and M_2 is the bMSC $M_1 \bullet M_2 = (E_1 \uplus E_2, \leq_{1\bullet 2}, \alpha_1 \cup \alpha_2, \phi_1 \cup \phi_2, A_1 \cup A_2, \prec_1 \uplus \prec_2)$, where:$\leq_{1\bullet 2} = \big(\leq_1 \uplus \leq_2 \uplus \{(e, e') \in E_1 \times E_2 | \phi_1(e_1) = \phi_2(e_2)\}\big)^*$*

As already mentioned, HMSCs are a kind of automaton labeled by partial orders. However, the automaton contained in a HMSC is only a support for sequential composition, and should not be considered as a synchronization defining a mandatory global state for all the instances. For this reason, the states of the support automaton in a HMSC H will often be called the *nodes* of H.

A *path* in a HMSC is a sequence of transitions $T = t_1.t_2. \dots .t_k$, such that the origin of t_{i+1} is equal to the goal of t_i: For every $i = 1 \dots k-1, \alpha_+(t_i) = \alpha_-(t_{i+1})$. Using sequential composition, each finite path $T = t_1.t_2. \dots .t_k$ of a HMSC defines a bMSC $O_T = \lambda(t_1) \bullet \lambda(t_2) \bullet$

$\cdots \bullet \lambda(t_k)$. A path $T = t_1.t_2.\ldots.t_k$ is a *circuit* if the origin of t_1 is equal to the goal of t_k. An *acyclic path* in a HMSC is a path $T = t_1.\ldots.t_k$ which contains no cycle, that is: $\forall i \leq j$, $\alpha_-(t_i) \neq \alpha_+(t_j)$. A *maximal acyclic path* of H is an acyclic path $T = t_1 \ldots t_k$ such that any extension of path T by one transition contains a cycle.

A *choice node* in a HMSC is a node that is the origin of two or more distinct transitions. As transitions in HMSCs are labeled by partial orders, choices can depict situations where several instances can decide to perform a scenario or another. This situation is called *non-local choice*. It was first identified in [14,2], and then refined in [9]. Consider, for instance the choice node in HMSC Figure 2. Following the description given by scenarios $M1$ and $M2$, from this node instance A can decide to send message $m1$ and then B must conform to this choice and receive $m1$, or instance B can decide to send the message $m2$, in which case instance A should receive $m2$. However, an implementation of such description without additional communications between A and B would probably lead to a situation where A sends message $m1$ while B sends message $m2$, and both instances are then deadlocked.

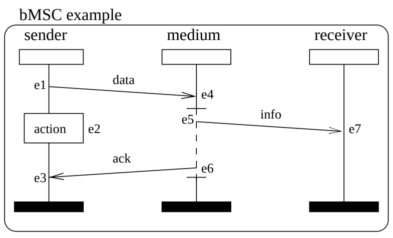

Fig. 1. An example of bMSC

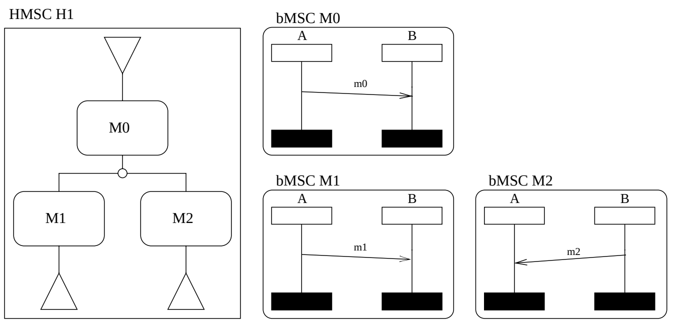

Fig. 2. non-local choice HMSC

The common understanding of choices in HMSCs is that the first instance able to perform a choice selects a behavior. The following instances reaching the same occurrence of this choice have to conform to the chosen scenario. So, the MSC semantics assumes an implicit agreement between instances (e.g., one instance decides and communicates its decision to all other instances, while the others wait until they are notified of the decision). When HMSCs are used to define a set of behaviors at a high abstraction level, this type of specification is not shocking: the only behaviors allowed are the scenarios depicted by each branch. However, when HMSCs are supposed to be precise enough to be implemented, non-local choices can have several conflicting interpretations. Indeed, the description of figure 2 can have several meanings. The first interpretation is that scenarios $M1$ and $M2$ are the only possible behaviors of the system. Communications must be added to the model to avoid message crossings. Another possible interpretation is that a third scenario where $m1$ intersects $m2$ is possible. This scenario should appear in the original HMSC.

The first definition of non-local choices in [2] assumes that any instance should communicate with other instances on each branch of a choice. This assumption limits the search for non-local choice to the set of edges leaving choice nodes. However, when considering weak sequential composition of bMSCs with disjoint set of instances, non-local choice is not a local property of choice nodes, but must be verified on the complete support automata [9]. In the sequel, we will adopt the following definition of non-local choice:

Definition 2.5 (Local Choice) *Let c be a choice node. c is* local *if and only if $\exists i \in I$ such that $\forall p$, path of H starting from c, $\phi(Min(O_p)) = \{i\}$. i will be called the* deciding instance *of choice c.*

Notice that the local choice property can be checked by considering maximal acyclic paths only. It is therefore decidable on finite HMSCs. Section 5, shows how a consensus algorithm can be inserted automatically to transform a non-local HMSC into a local one.

3 Amalgamated Sum of bMSCs

So far, bMSC composition is limited to parallel or sequential composition, iteration or choice. Other operations on bMSCs have been proposed, such as instance refinement [15], message refinement [5], virtuality [17], or more recently projections [6]. However, when two bMSCs depict different viewpoints of the same behavior, one feel the need for a merge operation that would glue the two scenarios to produce a result that contains both operands without creating copies of similar elements. This operator cannot be expressed by means of sequential nor parallel composition. We propose a merge operator for bMSCs called *amalgamated sum*. This amalgamated sum uses concepts of category theory. However, for the sake of conciseness, only what is strictly necessary has been included in the paper. More details on this topic can be found in [4,3]. First, we need to define the notion of bMSC morphisms, that will be essential to define common parts in scenarios.

Definition 3.1 (bMSC Morphism) *An* instance set morphism *is an injective mapping $\mu : I \longrightarrow I'$ from an instance set I to another instance set I'. Let I and I' be two finite sets of instances and $\mu_0 : I \rightarrow I'$ an instance set morphism. A* bMSC morphism *along μ_0, from $M = (E, \leq, A, \alpha, \phi, \prec)$, a bMSC over I, to $M' = (E', \leq', A', \alpha', \phi', \prec')$, bMSC over I', is a pair of mappings $\mu =< \mu_1, \mu_2 >$ where $\mu_1 : E \rightarrow E'$ is injective, $\mu_2 : A \rightarrow A'$ is a renaming mapping, and:*

(i) $\forall (e, f) \in E^2, e \leq f \Rightarrow \mu_1(e) \leq' \mu_1(f)$ *(ii)* $\forall (e, f) \in E^2, e \prec f \Rightarrow \mu_1(e) \prec' \mu_1(f)$

(iii) $\mu_0 \circ \phi = \phi' \circ \mu_1$ *(iv)* $\mu_2 \circ \alpha = \alpha' \circ \mu_1$

When no instance set morphism is specified, bMSC morphisms are defined by triples $\mu = (\mu_0, \mu_1, \mu_2)$ such that (μ_1, μ_2) is a bMSC morphism along μ_0.

Note that property (iii) also means that all events located on a single instance of M are sent by μ_1 on a single instance of M'.

Definition 3.2 (Amalgamated Sum of Two Sets) *Let I, J and K be three finite sets. Let $f : I \rightarrow J$ and $g : I \rightarrow K$ be two injective maps. The amalgamated sum $J\ {}_f{+}_g\ K$ is defined as $J\ {}_f{+}_g\ K = \big(J \backslash f(I)\big) \biguplus \big(K \backslash g(I)\big) \biguplus I$. The amalgamated sum yields two injections $\tilde{f} : J \rightarrow J\ {}_f{+}_g\ K$ and $\tilde{g} : K \rightarrow J\ {}_f{+}_g\ K$ defined as follows:*

$$\begin{cases} \forall i \in f(I), & \tilde{f}(i) = f^{-1}(i) \\ \forall i \in J \setminus f(I), & \tilde{f}(i) = i \end{cases} \qquad \begin{cases} \forall i \in g(I), & \tilde{g}(i) = g^{-1}(i) \\ \forall i \in K \setminus g(I), & \tilde{g}(i) = i \end{cases}$$

Note that as we use $\uplus$ (disjoint union) in our definition, the result of an amalgamated sum can contain several copies of similar elements.

Amalgamated sums of sets will be used to amalgamate sets of instances, events or actions of two bMSCs to be composed.

Definition 3.3 (Amalgamated Sum of two bMSCs) *Let $M_0 = (I_0, E_0, \leq_0, A_0, \alpha_0, \phi_0, \prec_0)$, $M_1 = (I_1, E_1, \leq_1, A_1, \alpha_1, \phi_1, \prec_1)$, $M_2 = (I_2, E_2, \leq_2, A_2, \alpha_2, \phi_2, \prec_2)$ be three bMSCs and $f =< f_0, f_1, f_2 >: M_0 \to M_1$, $g =< g_0, g_1, g_2 >: M_0 \to M_2$ be two bMSCs morphisms. The amalgamated sum of M_1 and M_2 wrt. f and g is the bMSC $M = M_1\ {}_f+_g\ M_2$ where $M = (I, E, \leq, A, \alpha, \phi, \prec)$ is defined by:*

- $I = I_1\ {}_{f_0}+_{g_0}\ I_2$; $\quad E = E_1\ {}_{f_1}+_{g_1}\ E_2$; $\quad A = A_1\ {}_{f_2}+_{g_2}\ A_2$;
- *Preorder relation $\leq$ is the transitive closure of $\tilde{f}_1(\leq_1) \cup \tilde{g}_1(\leq_2)$;*
- $\forall e \in E, \alpha(e) = \begin{cases} \alpha_1(e) \text{ if } e \in E_1 \backslash f_1(E_0) \\ \alpha_2(e) \text{ if } e \in E_2 \backslash f_2(E_0) \\ \alpha_0(e) \text{ otherwise} \end{cases}, \qquad \phi(e) = \begin{cases} \phi_1(e) \text{ if } e \in E_1 \backslash f_1(E_0) \\ \phi_2(e) \text{ if } e \in E_2 \backslash f_2(E_0) \\ \phi_0(e) \text{ otherwise} \end{cases}$
- $\prec = \tilde{f}_1(\prec_1) \cup \tilde{g}_1(\prec_2)$.

The bMSC M_0 is called the interface *of the amalgamated sum $M_1\ {}_f+_g\ M_2$.*

Let us illustrate the use of amalgamated sum on the example of Figure 3. Considering bMSCs $M_1 = (I_1, E_1, \leq_1, A_1, \alpha_1, \phi_1, \prec_1)$ and $M_2 = (I_2, E_2, \leq_2, A_2, \alpha_2, \phi_2, \prec_2)$ as two partial observations of the same system, we want to produce a behavior that contains M_1 and M_2. Let us also suppose that even if M_1 and M_2 have different instance sets, instance X in M_2 and instance *sender* in M_1 (resp. Y and *medium*) represent the same object in the system. Intuitively, merging M_1 and M_2 then amounts to inserting an atomic action between *data* emission and *ack* reception in M_1, and renaming the instances. Formally, the merge consists in the definition of an interface that identifies the common elements (events, action name, and instances) in M_1 and M_2 and renames them. For our example, this is done using a new bMSC $M_0 = (I_0, E_0, \leq_0, A_0, \alpha_0, \phi_0, \prec_0)$, and two bMSC morphisms $f : M_0 \to M_1$ and $g : M_0 \to M_2$ defined below.

- Morphism $f : M_0 \to M_1$ is a triple $f =< f_0, f_1, f_2 >$, where:
  - · $f_0 : I_0 \to I_1$ is the identity,
  - · $f_1 : E_0 \to E_1$ sends respectively $eI1, eI2, eI3, eI4$ onto $ev1, ev2, ev3, ev4$,
  - · $f_2 : A_0 \to A_1$ is the identity.
- Morphism $g : M_0 \to M_2$ is a triple $g =< g_0, g_1, g_2 >$, where:
  - · $g_0 : I_0 \to I_2$ sends *sender* onto X and *medium* onto Y,
  - · $g_1 : E_0 \to E_2$ sends respectively $eI1, eI2, eI3, eI4$ onto $evt1, evt3, evt4, evt5$,

· $g_2 : A_0 \rightarrow A_2$ sends respectively $!data, ?data, !ack, ?ack$ onto $!m1, ?m1, !m2, ?m2$.

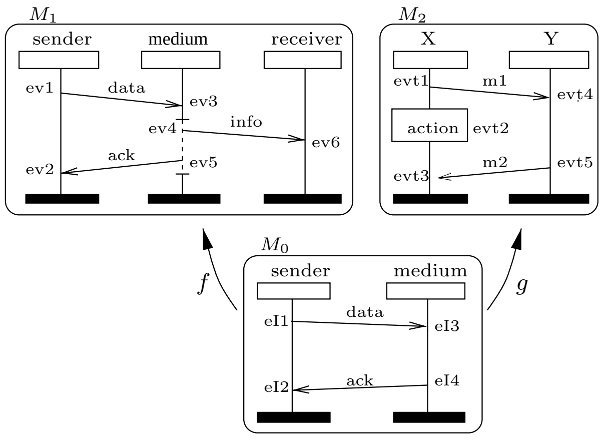

Fig. 3. An example of amalgamated sum

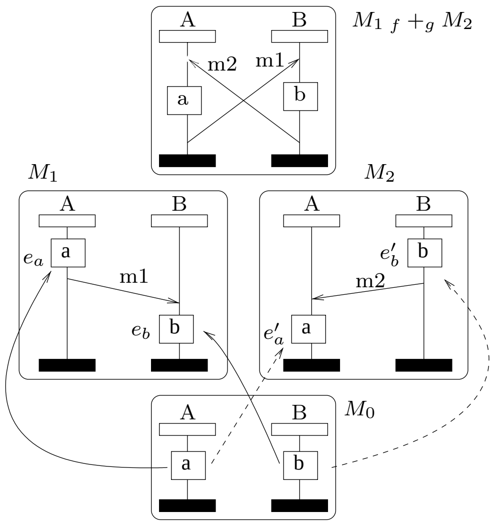

Fig. 4. An amalgamated sum that is not well-formed

The result of the amalgamated sum $M_1\ {}_f+_g M_2$ is the bMSC of Figure 1. Note that the names of the resulting instances on common parts are defined by the instance names of the interface (we could have proposed an instance name "*X* + *Sender*" instead of keeping *Sender* in the amalgamated sum.)

Note also that an amalgamated sum of two well-formed bMSCs is not always a well-formed bMSC, as the least preorder containing $\leq_1$ and $\leq_2$ may not be antisymmetric. Consider the example of Figure 4. MSC M_1 imposes that $e_a \leq e_b$ while MSC M_2 states that $e'_b \leq e'_a$. Clearly, if e_a and e_b are respectively identified with e'_a and e'_b by the interface, the amalgamated sum of M_1 and M_2 will create a symmetry, which can be easily detected. In such case, the two scenarios are incompatible, at least if one considers a matching between events symbolizing actions a and b in both operands. For more considerations about amalgamated sum properties, consult [4].

4 Fibered product of HMSCs

In many cases, merging bMSCs is not sufficient, and a question that immediately arises is how to extend the amalgamated sum approach to HMSCs. The first idea is to work on the set of bMSCs generated by a HMSC. The composition of two sets of bMSCs would be the set of coherent amalgamated sums obtained by merging pairs of bMSCs from each set. However, the result obtained is not always a set of partial orders that can be generated by a single HMSC. Consider, for example, the two HMSCs of Figure 5. Trying to merge the atomic actions labeled by a and b in both HMSCs would result in the set of bMSCs described by Figure 6, where one can say nothing about the ordering relationship between the receipt and sending of the message m, and the events corresponding to the messages labelled by n. Clearly, this set is not generated by a HMSC, as the messages of type m can cross an unbounded number of messages of type n. This kind of specification can be expressed by means of Compositional Message Sequence Charts [7], or Extended compositional Message Sequence charts [13], but not with HMSCs.

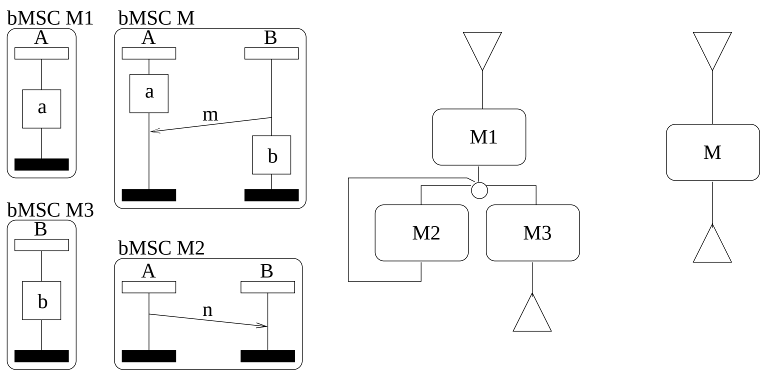

Fig. 5. Event by event matching

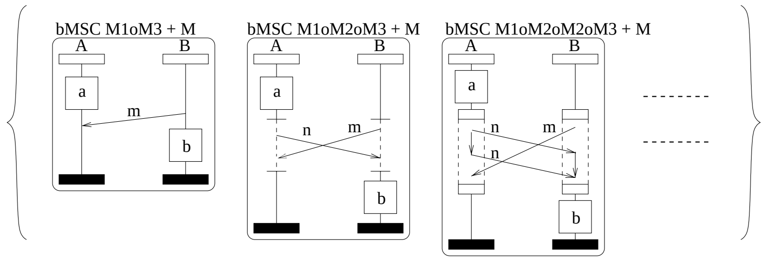

Fig. 6. Result

We propose a solution based on a partially synchronized product of HMSCs inspired by the fibered product of asynchronous transition systems proposed in [3]. The fibered product of HMSCs is twofold: First, transitions of the two support automata are synchronized partially. Then, the amalgamated sums of the bMSCs attached to the transitions being synchronized are computed to create new bMSCs. As for the amalgamated sum of section 3, the formal definition of the fibered product of HMSCs relies on a notion of morphism.

Definition 4.1 (LTS Morphism) *Let $\mathcal{S}_1 = (S_1, \widehat{s}_1, \Sigma_1, T_1)$ and $\mathcal{S}_2 = (S_2, \widehat{s}_2, \Sigma_2, T_2)$ be two labeled transition systems (LTS for short). A LTS morphism $f : \mathcal{S}_1 \to \mathcal{S}_2$ is a pair $f =< f_1, f_2 >$ where $f_1 : S_1 \to S_2$ is a total function while $f_2 : T_1 \rightharpoonup T_2$ is a partial function which satisfy:*

i) $f_1(\widehat{s}_1) = \widehat{s}_2$;

ii) $t_1 = (p, a, q)$ *in* T_1 *and* $f_2(t_1)$ *defined imply* $\exists b \in \Sigma_2, f_2(t_1) = (f_1(p), b, f_1(q))$ *in* T_2;

iii) $t_1 = (p, a, q)$ *in* T_1 *and* $f_2(t_1)$ *undefined imply* $f_1(p) = f_1(q)$ *in* $\mathcal{S}_2$.

Condition $i)$ ensures that morphisms preserve initial states. According to conditions $ii)$ and $iii)$, any transition $t \in T_1$ of $\mathcal{S}_1$ is mapped to a transition of $\mathcal{S}_2$ via f_2 if f_2 is defined in t. In other words, f_2 defines which transitions of $\mathcal{S}_1$ have observable effects in $\mathcal{S}_2$. For convenience, we can add an artificial *empty* transition $p \xrightarrow{\epsilon} p$ to each state $p \in S$. With this convention, transitions of $\mathcal{S}_1$ that are unobservable in $\mathcal{S}_2$ are mapped to empty transition. Hence, we can succinctly rewrite the second and third conditions of definition 4.1 as follows:

$$t = (p, a, q) \text{ transition of } \mathcal{S}_1 \Longrightarrow \exists a' \in \Sigma_2 \cup \epsilon, f_2(t) = (f_1(p), a', f_1(q)) \text{ transition of } \mathcal{S}_2$$

This convention is used throughout the paper. Therefore, it is now assumed that transition system are completed in every state with an empty transition of the form $p \xrightarrow{\epsilon} p$.

The partially synchronized product of labeled transition systems was first introduced by Arnold [1]. It is parameterized by a set of synchronization vectors, which is used to define which pairs of labels must be synchronized. Here, a set of pairs of transitions (instead of labels) is used for this purpose.

Definition 4.2 (Synchronous product) *Let* $\mathcal{S}_1 = (S_1, \widehat{s}_1, \Sigma_1, T_1)$ *and* $\mathcal{S}_2 = (S_2, \widehat{s}_2, \Sigma_2, T_2)$ *be two LTSs. A* synchronization constraint *C is a subset of $T_1 \times T_2$. Elements of C are called* synchronization vectors*, and indicate which transitions of $\mathcal{S}_1$ and $\mathcal{S}_2$ are synchronized. The* synchronous product *of $\mathcal{S}_1$ and $\mathcal{S}_2$ under a synchronization constraint C is the LTS $\mathcal{S} = (S, \widehat{s}, \Sigma, T)$ where:*

- $S = S_1 \times S_2$; $\widehat{s} = (\widehat{s}_1, \widehat{s}_2)$;
- $\Sigma = \big\{(\alpha(t_1), \alpha(t_2)) \mid (t_1, t_2) \in C\big\}$;
- $T = \big\{(s, \sigma, s') \in S \times \Sigma \times S \mid \exists (t_1, t_2) \in C, \sigma = (\alpha(t_1), \alpha(t_2)) \wedge s = (\alpha_-(t_1), \alpha_-(t_2)) \wedge s' = (\alpha_+(t1), \alpha_+(t_2))\big\}$.

In the previous section, we have defined a notion of morphism for bMSC. HMSC morphisms can be defined, in a similar way, as triples of morphisms or mappings: i) a morphism of instance sets, ii) a morphism of labeled transition systems and iii) a mapping that associates bMSC morphisms to transitions.

Definition 4.3 (HMSC morphism) *Let $H_1 = (I_1, \mathcal{S}_1, \mathcal{M}_1, \lambda_1)$, $H_2 = (I_2, \mathcal{S}_2, \mathcal{M}_2, \lambda_2)$ be two HMSCs. A HMSC morphism $f : H_1 \to H_2$ from H_1 to H_2 is a triple $f =< f_0, f_1, f_2 >$, where $f_0 : I_2 \to I_1$ is an instance set morphism, $f_1 =< f_{1,1}, f_{1,2} >: \mathcal{S}_1 \to \mathcal{S}_2$ is a transition system morphism, and f_2 maps transitions of T_1 to bMSC morphisms from $\lambda_2 \circ f_{1,2}(T)$ to $\lambda_1(T)$.*

Definition 4.4 (Fibered product of HMSCs) *Let H_0, H_1 and H_2 be three HMSCs, and let $f =< f_0, f_1 =< f_{1,1}, f_{1,2} >, f_2 >: H_1 \to H_0$ and $g =< g_0, g_1 =< g_{1,1}, g_{1,2} >, g_2 >: H_2 \to H_0$ be two HMSC morphisms. The fibered product of H_1 and H_2 over f and g is noted $H_1\ {}_f\!\times_g H_2$, and is the HMSC $H_1\ {}_f\!\times_g H_2 = (I, \mathcal{S}, \mathcal{M}, \lambda)$, where:*

- $I = I_1\ {}_{f_0}\!+_{g_0} I_2$;
- *$\mathcal{S} = (S, \widehat{s}, \Sigma, T)$ is the synchronous product of $\mathcal{S}_1$ and $\mathcal{S}_2$ under $C = \big\{(t_1, t_2) \in T_1 \times T_2 \mid f_{1,2}(t_1) = g_{1,2}(t_2)\big\}$;*
- $\lambda(t_1, t_2) = \lambda_1(t_1)\ {}_{f_2(t_1)}\!+_{g_2(t_2)} \lambda_2(t_2)$;
- $\mathcal{M} = \lambda(T)$.

Let us illustrate this notion of fibered product of HMSCs on an example. Figure 7 shows a product of two HMSCs $H1$ and $H2$ with two HMSC morphisms $f =< f_0, f_1, f_2 >$ that maps $H1$ to an interface HMSC H_I and $g =< g_0, g_1, g_2 >$ that maps $H2$ to H_I. The result of this product is given by HMSC *Result*. The real difficulty of HMSC product is the definition of an interface HMSC (H_I in our example) and of the corresponding HMSC morphisms (f and g). As we just want to illustrate HMSC product, we will not detail the instance sets on which these HMSCs are defined nor the corresponding morphisms (f_0 and g_0). We will not either detail f_2 and g_2 that provide the bMSC morphisms used to build amalgamated sums $loop + Nominal$ and $loop + deadlock$, and will only focus on the support automata product.

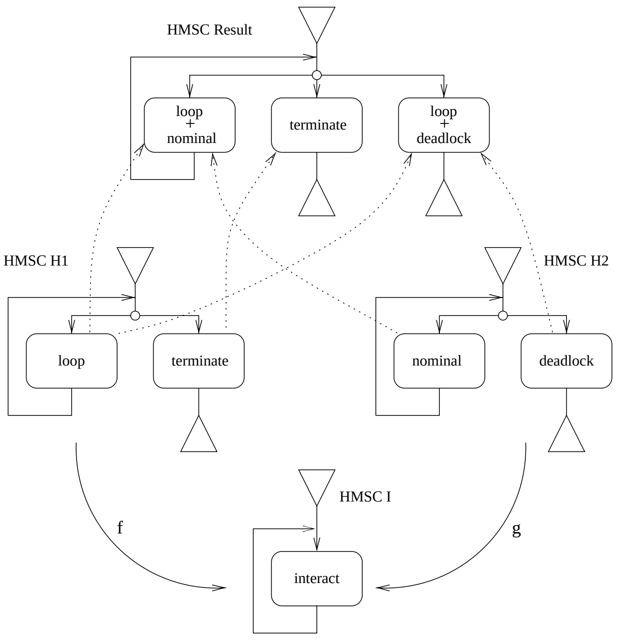

Fig. 7. product of HMSCs H1 $_f\times_g$ H2

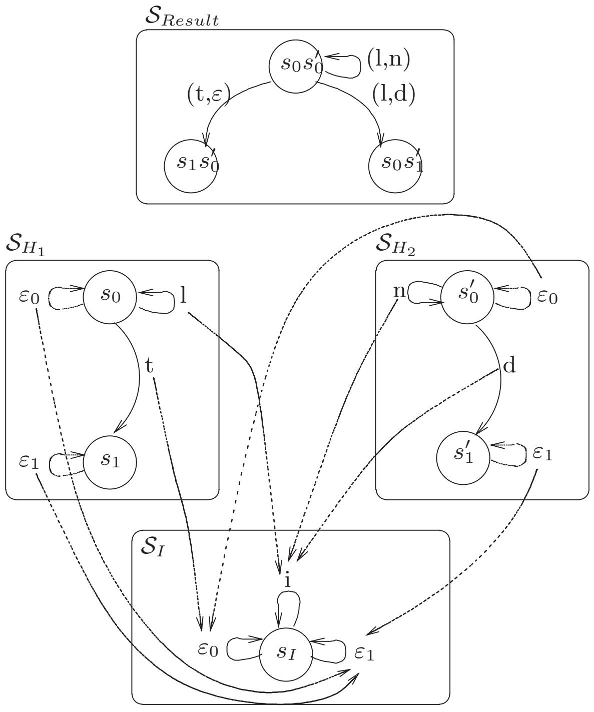

Fig. 8. Support automata for HMSCs Fig 7

Let us detail the two LTS morphisms f_1 and g_1 that are used to build the product of $H1$ and $H2$'s support automata. The support automaton S_{H1} of $H1$ comports transitions with labels l and t, that are respectively mapped to bMSCs *Loop* and *terminate* by λ_1 and the support automaton S_{H2} of $H2$ comports two transitions n and d that are respectively mapped to bMSCs *nominal* and *deadlock* by λ_2. Following the convention on LTS, additional ϵ-transitions are added to the support automata. Morphism f_1 sends respectively the transition labeled $l, t, \epsilon_0, \epsilon_1$ to those labeled $i, \epsilon_0, \epsilon_1, \epsilon_1$. In Figure 8, morphisms f_1 and g_1 are symbolized by dashed arrows from transitions of S_{H1} (respectively S_{H2}) to transitions of S_I. For this example, the synchronization constraint C built from f_1 and g_1 is defined by

$$C = \left\{ \begin{array}{l} \big((s_0, l, s_0), (s_0', n, s_0')\big); \big((s_0, l, s_0), (s_0', d, s_1')\big); \\ \big((s_0, t, s_1), (s_0', \varepsilon_0, s_0')\big); \big((s_0, \varepsilon_0, s_0), (s_1', \varepsilon_1, s_1')\big); \big((s_1, \varepsilon_1, s_1), (s_1', \varepsilon_1, s_1')\big) \end{array} \right\},$$

The synchronous product of support automata $\mathcal{S}_{H_1} \times \mathcal{S}_{H_2}$ with C is given by the support automaton $\mathcal{S}_{Result}$ of Figure 8. For the sake of clarity, empty transitions (pair of ϵ-transitions) are not represented on the product automaton. To obtain HMSC Result, the bMSCs associated to transitions of $\mathcal{S}_{Result}$ can then be amalgamated. For a given transition (t_1, t_2) of the support automaton of the fibered product, the bMSC $\lambda((t_1, t_2))$ is the amalgamated sum of the bMSC associated to transition t_1 and the bMSC associated to transition t_2. The two bMSCs morphisms needed for amalgamated sum are given by $f_2(t_1)$ and $g_2(t_2)$.

5 Non-local choices suppression

Let us illustrate our HMSC product on a concrete application. As mentioned in Section 2, some HMSCs can contain non-local choices. During implementation, this abstraction is error prone. To implement a system described by such a HMSC, an experimented programmer would certainly add some communication messages to the HMSC, so that any non-local choice would be turned into a local one. More generally, this kind of control can be implemented via well known distributed consensus protocols.

The approach proposed in this section is to integrate automatically a consensus protocol described with HMSC in a non-local description of a system. This integration is done using the fibered product between the non-local choice HMSC and another HMSC describing the consensus protocol. An interesting property of this HMSC transformation is that it can be used for an arbitrary number of instances participating to a non-local choice.

Let us consider the non-local HMSC $H1$ of figure 2. This description can be transformed into a local one by insertion of a control protocol that will tell instances A and B which branch they should follow. This protocol should have a single minimal event for each branch of the choice, and this minimal event must be performed by the same instance to ensure locality of the choice. A first solution is that an instance among all participating instances is designated as a master, and initiates a token ring protocol involving all participating instances in a certain order. This token ring is used to elect the branch to be performed. A drawback of this solution is that the symmetry of the HMSC is lost.

Another solution is to add a supervisor instance to our HMSC. The role of the supervisor instance is to ask all instances which branch they want to perform, to select an answer among the responses received, and to transmit this value to all instances. Several decision policies can be used. However, a first arrived, first served policy is assumed in this paper. Let us try to use this second solution to transform HMSC $H1$. So, the local HMSC we should obtain after transformation will look like the HMSC of Figure 9.

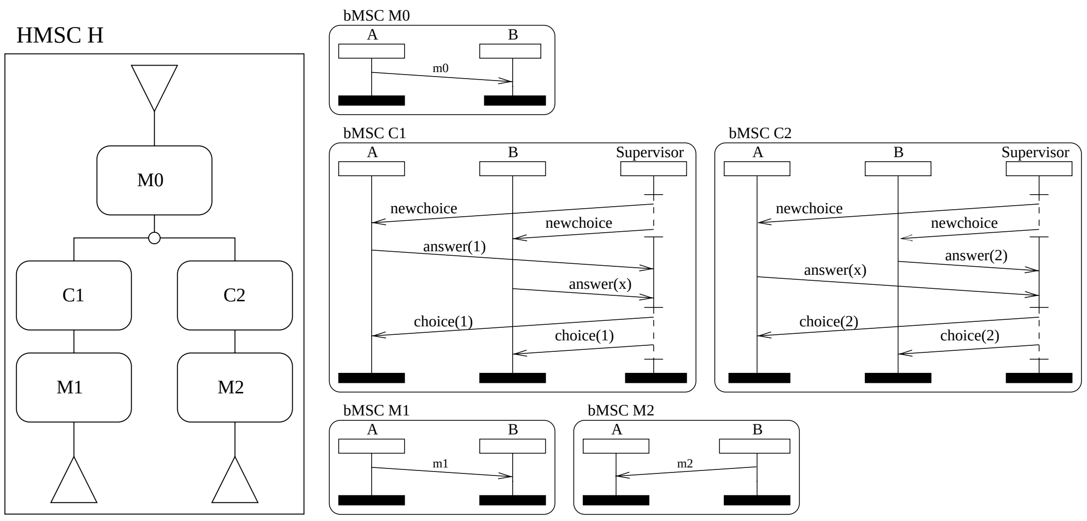

Fig. 9. Local HMSC

This transformation raises two important issues. The first one is the creation of an election protocol for an arbitrary number of instances. Indeed, to make such an approach usable, one cannot ask a user to design n scenarios when n instances participate in a non-local choice. Fortunately, building such scenarios can be accomplished as an application of the amalgamated sum of bMSCs, and can be completely automated. The second issue is the insertion of the protocols right after choice nodes. This can be accomplished using the fibered product of HMSCs described in section 4. For this application, the interface HMSC is simple and can be computed automatically.

5.1 *Consensus Protocol construction*

As mentioned before, we have chosen to add a supervisor with a first arrived, first served policy to ensure locality of choices. The main idea is to insert a preamble C_i on each branch B_i of a non-local choice. Each preamble is a protocol in which the supervisor sends messages to all instances that are participating to the non-local choice. The first instance i that answers receives an acknowledgment of its choice, and all others receive a notification of the choice of instance i. The preamble protocol is depicted in Figure 10.

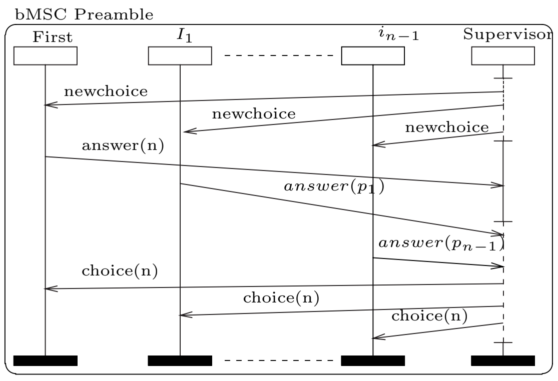

Fig. 10. Description of a preamble

However, for a non-local choice involving up to n branches, a designer would have to define up to n quasi-identical preambles, which is a tedious work, and can be error-prone. Fortunately, these preambles can be constructed automatically using the amalgamated sum. First, let us show how a preamble can be constructed for two instances. The only bMSCs needed are: a generic bMSC *C_Generic* with 3 instances (a supervisor, the first instance to answer, and the last instance) and an empty bMSC M over the participating instances. Figure 11 shows two sums with such kind of empty bMSCs. As the generic bMSC already contain 3 instances, the only task to perform to build our preamble is a renaming of $First$ and Z. This can be done with the sum $M\ {}_f{+}_g\ C\_Generic$, where $f : M\_Interface \mapsto M$ is the identity morphism, and $g = (g_0, g_1, g_2) : M\_Interface \rightarrow C\_Generic$ is a morphism where g_1 and g_2 are empty functions and g_0 is an instance set morphism that associates respectively A to $First$ and B to Z on one side and B to $First$ and A to Z on the other side. With no surprise, our sums produce bMSCs $C1$ and $C2$ of Figure 9, which are only renamings of the generic bMSC. As all our preambles will be isomorphic bMSCs, it is however important to note that renaming can be expressed using an amalgamated sum.

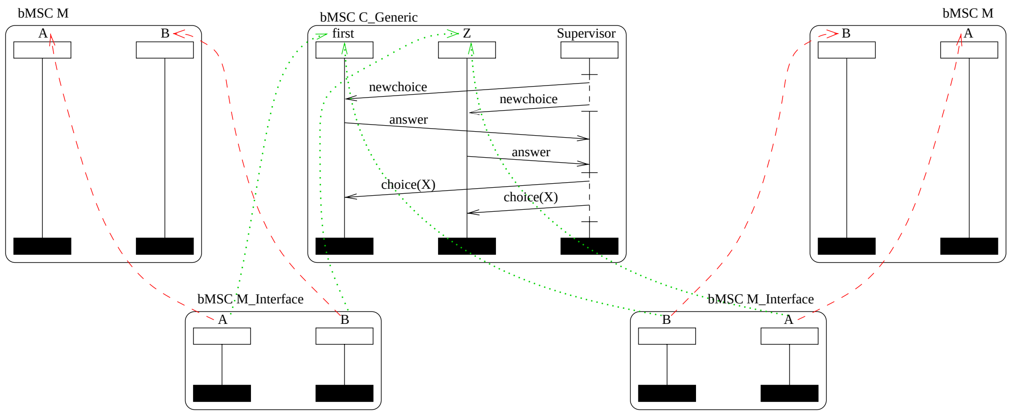

Fig. 11. Preambles for a pair of instances

5.2 Generalization

So far, it has been shown how to rename a generic preamble in the specification of a consensus protocol involving two instances and two different branches. This protocol can be generalized to an arbitrary number of instances. Let us note $C\_Generic(n)$ a preamble involving n instances as depicted in Figure 10.

First, let us show that there exists an interface MSC $C\_Interface$ and two morphisms $f : C\_Interface \longrightarrow C\_Generic(2)$ and $g : C\_Interface \longrightarrow C\_Generic(2)$ such that $C\_Generic(3) = C\_Generic(2)\ _f +_g C\_Generic(2)$. The two bMSCs $C\_Interface = (E_I, \leq_I, A_I, \alpha_I, \phi_I, \prec_I)$ over I_I and $C\_Generic(2) = (E_2, \leq_2, A_2, \alpha_2, \phi_2, \prec_2)$ over I_2 are depicted in Figure 12. $f : C\_Interface \rightarrow C\_Generic(2)$ is defined by the triple $f =< f_0, f_1, f_2 >$, where:

- $f_0 : I_I \longrightarrow I_2$ is the identity morphism;
- $f_1 : E_I \longrightarrow E_2$ respectively maps $eI_1, eI_2, eI_3, eI_4, eI_5, eI_6$ to $e2_1, e2_2, e2_5, e2_6, e2_9, e2_{10}$
- $f_2 : A_I \longrightarrow A_2$ is the identity morphism.

f and g are isomorphic morphisms, and g is defined similarly from $C\_Interface$ to another copy of $C\_Generic(2)$. Figure 12 illustrates the construction of C_Generic(3) with bMSCs C_Generic(2), C_Interface, and bMSC morphisms f and g. This construction can be generalized to build a bMSC $C\_Generic(n)$, that supervises the choices of n instances, using the bMSCs C_Interface and C_Generic of Figure 12, and two bMSC morphisms f_{n-1} and g_{n-1}. It is defined by induction: $C\_Generic(n) = C\_Generic(n-1)\ _{f_{n-1}} +_{g_{n-1}} C\_Generic(2)$, where $\forall n \in \mathbb{N}$, g_n is the bMSC morphism $g : C\_Interface \rightarrow C\_Generic$ defined for two instances, and $f_{n-1} = (id, f_2, id) : Interface \longrightarrow C\_Generic(n-1)$ is a morphism such that f_2 maps: events eI_1 and eI_2 to two events x and y associated to the emission and reception of message $newchoice$ from $Supervisor$ to $First$; events eI_3 and eI_4 to two events x' and y' associated to the emission and reception of message $answer(p)$ from $First$ to $Supervisor$; and events eI_5 and eI_6 to two events x'' and y'' associated to the emission and reception of message $Choice(X)$ from $Supervisor$ to $First$.

For a branch B_i such that $\phi(min(B_i)) = j$, the preamble C_i will be a renaming of $C\_Generic(n)$ through an instance renaming morphism that maps j to $first$ and all other

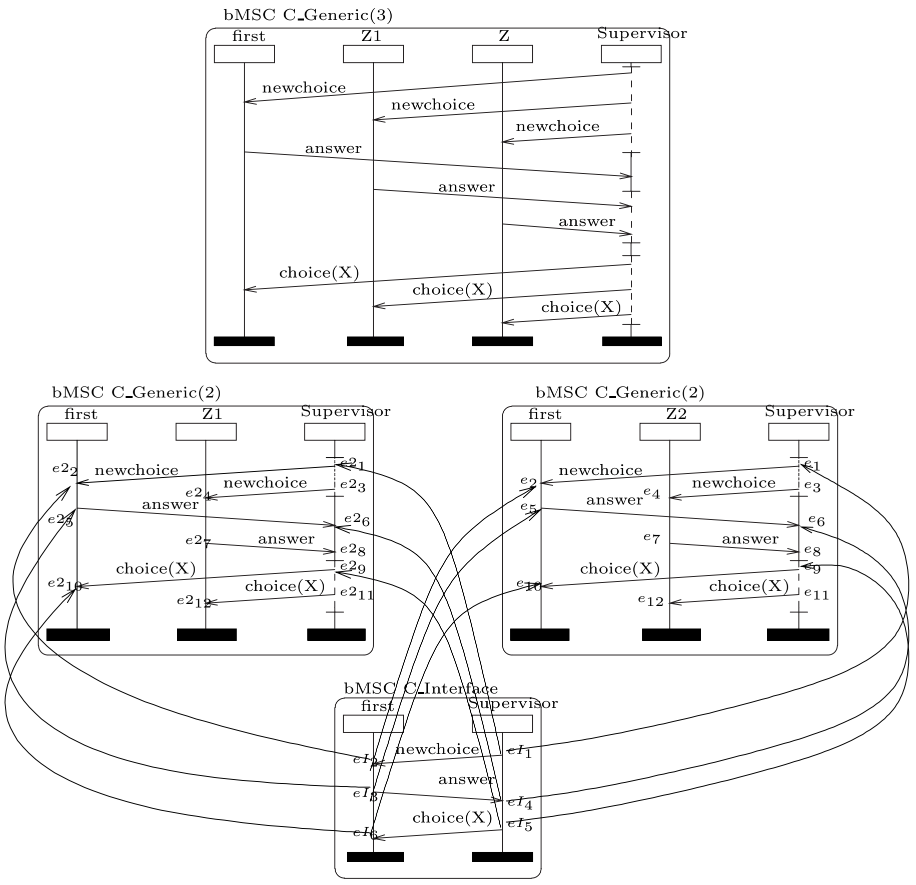

Fig. 12. the amalgamated sum $C\_Generic(2) + C\_Generic = C\_Generic(3)$

instances to any instance $I_q, q \in 1..n-1$, and through a message renaming morphisms that modifies the message names.

5.3 Insertion of preambles

Once the desired preambles have been constructed and renamed, they have to be inserted in the original HMSC. This task is performed using the fibered product operation defined in section 4. For example, building HMSC H of Figure 9 from HMSC $H1$ amounts to the insertion of bMSCs $C1$ and $C2$ created in Figure9 after the non-local choice node of $H1$. This insertion can be defined as a product $H1\ {}_f\times_g H2$ between $H1$ and the HMSC $H2$ of Figure 14.

The difficult part of insertion is the creation of an interface HMSC. Let us illustrate a simple insertion on an example. Consider the simple support automata $\mathcal{S}_1$ and $\mathcal{S}_2$ of Figure 13, and suppose we want to obtain a support automaton accepting sequence $b.a$. A way to "insert" transition b before transition a is to synchronize transition b with transition ϵ_0 and transition a with transition ϵ'_1. This synchronization is implemented via the morphisms proposed in Figure 13.

The solution proposed for simple insertions can be generalized as shown in Figure 15. Let f be the HMSC morphism from $H1 = (I_{H1}, \mathcal{S}_{H1}, \mathcal{M}_{H1}, \lambda_{H1})$ to $H0 = (I_{H0}, \mathcal{S}_{H0}, \mathcal{M}_{H0}, \lambda_{H0})$

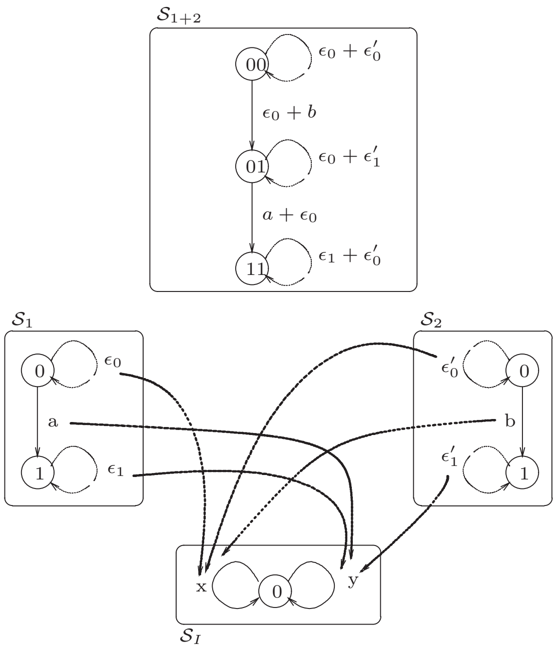

Fig. 13. A simple insertion

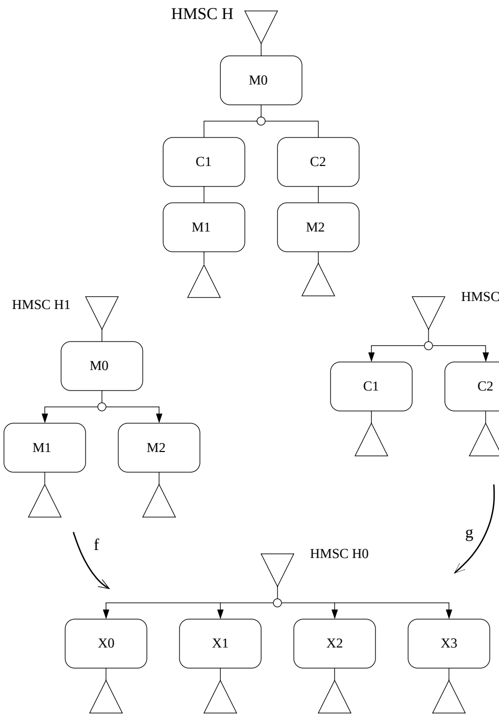

Fig. 14. Product H1 ${}_f{\times}_g$ H2

and g be the HMSC morphism from $H2 = (I_{H2}, \mathcal{S}_{H2}, \mathcal{M}_{H2}, \lambda_{H2})$ to H_0. More precisely:

- $f : H1 \rightarrow H0$ is a triple $f = \langle f_0, f_1 =< f_{1,1}, f_{1,2} >, f_2 \rangle$, where:
  - $f_0 : I_{H0} \rightarrow I_{H1}$ is the identity; $f_{1,1}$ maps all states of $\mathcal{S}_{H1}$ to S_I;
  - $f_{1,2} : T_{H1} \longrightarrow T_{H0}$ respectively maps $(s_0, m_0, s_1), (s_1, \varepsilon_0, s_1), (s_1, m_1, s_2)$ and (s_1, m_2, s_3) to $(s_I, x_0, s_I), (s_I, x_1, s_I), (s_I, x_2, s_I)$ and (s_I, x_3, s_I);
  - $f_2 : T_1 \rightarrow \mathcal{M}_0 \rightarrow \mathcal{M}_1$ associates a null bMSC morphism from $\lambda_0 \circ f_1(t)$ to $\lambda_1(t)$ to each transition $t \in T_1$.
- $g : H_2 \rightarrow H_0$ is a triple $g =< g_0, g_1, g_2 >$, where:
  - $g_0 : I_{H0} \rightarrow I_{H1}$ is the identity; $g_{1,1}$ maps all states of $\mathcal{S}_{H2}$ to S_I;
  - $g_{1,2} : T_{H2} \longrightarrow T_{H0}$ respectively maps $(s'_0, \varepsilon'_0, s'_0), (s'_0, c_1, s'_1), (s'_0, c_2, s'_2), (s'_1, \varepsilon'_1, s'_1)$ and $(s'_2, \varepsilon'_2, s'_2)$ to $(s_I, x_0, s_I), (s_I, x_1, s_I), (s_I, x_1, s_I), (s_I, x_2, s_I)$ and (s_I, x_3, s_I);
  - $g_2 : T_2 \rightarrow \mathcal{M}_0 \rightarrow \mathcal{M}_2$ which associates a null bMSCs morphism from $\lambda_0 \circ g_1(t)$ to $\lambda_2(t)$, to each transition $t \in T_2$.

Note that a transition (s, ε_s, s) is artificially added to each state. These transitions allow independent moves of transition systems. Of course, λ will associate an empty bMSC M_ϵ(bMSC with no instance or event) to each ϵ-transition. With this convention, we can define two bMSC morphisms $f : M_\epsilon \longrightarrow \lambda(t)$ and $g : M_\epsilon \longrightarrow M_\epsilon$, such that $\lambda(t) \,{}_f{+}_g\, \lambda(\varepsilon) = \lambda(t)$.

With morphisms f_1 and g_1 defined above, we can build the following synchronization constraint $C = \big(((s_0, m_0, s_1), (s_I, x_0, s_I)); ((s_1, \varepsilon_0, s_1), (s'_0, c_1, s'_1)); ((s_1, \varepsilon_0, s_1), (s'_0, c_2, s'_2));$ $((s_1, m_1, s_2), (s'_1, \varepsilon'_1, s'_1)); ((s_1, m_2, s_3), (s'_2, \varepsilon'_2, s'_2))\big)$. The partially synchronized product $\mathcal{S}_{H_1} \times \mathcal{S}_{H_2}$ under C is the support automaton $\mathcal{S}_H$. Hence, with HMSC $H0$, $H1$ and $H2$, and with the definition of trivial HMSC morphisms f and g, we obtain the local HMSC H of Figure 9.

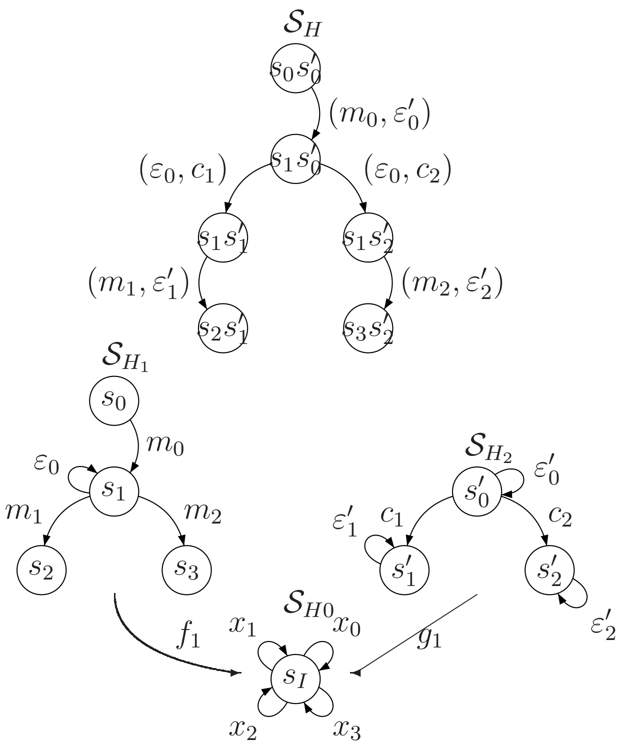

Fig. 15. labeled transition systems of HMSCs

To summarize, from a non-local HMSC H, one can produce a HMSC *Protocol* which bMSCs are built as a succession of amalgamated sums of a generic bMSC, followed by a renaming. Then, the insertion of *Protocol* into H is performed with a product. All interfaces and morphisms used for this "localization" are simple and independent of the number of instances, hence this procedure can be completely automated for an arbitrary number of instances.

6 Conclusion

This paper has proposed a formal notion of merge for bMSCs and HMSCs. This definition is based on the notions of synchronous product for transition systems and amalgamated sum. The use of this formalism on a concrete example shows the applicability of the method.

The definition of an HMSC product mainly relies on the definition of appropriate morphisms, which can be considered rather complicated at first sight. Furthermore, morphisms force some synchronization between bMSCs, which can be considered as an arbitrary solution (remember that due to the meaning of sequential composition, there are several ways to obtain similar orderings between events, and hence that decomposition of scenarios into bMSCs is not always meaningful). A clue to refine merging is to define HMSC morphisms not only as transition morphisms but rather as path morphisms, as is done in [3]. Note also that amalgamated sums define explicitely events that coincide in both views. This can be considered as a drawback, but relying on events labeling to detect similarities in views is not possible, as in the general case this approach exhibits several matching possibilities (and even an infinity when HMSCs are considered). However, we believe that for most cases where our product can be applied, the morphisms are rather simple, and can even be computed automatically. For the non local choice case, all morphisms are trivial, and do not use the full expressive power of synchronized HMSC product. Furthermore, the construction of local HMSCs from non local ones can be completely automatized.

An advantage of this composition method is that it allows the inductive definition of merging, and hence provides composition schemes for arbitrary number of instances. This product also defines a rough notion of coherence : if the HMSC obtained after composition is well formed, then the two operands are coherent on the set of events identified through the morphisms. If not, the fusion of both views are inconsistent.

Clues for future works are of practical and theoretical nature. On the practical side, we want to test our product through the definition of product-based transformation patterns for HMSCs. Of course, we do not plan to ask users to define their own fibered product of HMSCs, but rather to provide libraries of formally defined patterns, than can be reused after instantiation of a limited number of parameters. Composition can also be used to create automatically concrete scenarios involving numerous instances that cannot be designed by hand from generic scenarios. These concrete scenarios could then be used for deployment, test and simulation purposes.

On the theoretical side, an interesting work consists in studying in detail the properties of algebraic operations defined as peculiar instances of product. This can be the base for a categorically well founded algebraic framework for scenario composition. Another interesting research direction is to study merging in a category of extended MSC (compositional MSCs [7] for example).

The relation between HMSC morphisms and projections is also an interesting research topic. A question that seems quite natural is whether HMSC morphisms are inverse projections of HMSC, as defined in [6]. The answer is no, as morphisms do not preserve events labeling. When considering only label preserving morphisms, the answer is still no : interfaces are always HMSCs but HMSC projections are not HMSCs anymore as they can generate infinite connected patterns that can not be represented as compositions of a finite set of bMSCs. However, the relationship between projections and morphisms is something that is worth being studied, as finding a correct morphism is sometimes equivalent to finding an appropriate projection. The ideal case is when two HMSCs project on the same interface HMSC.

References

[1] André Arnold. *Finite transition systems.* Prentice-Hall. Prentice-Hall, 1994.

[2] H. Ben-Abdallah and S. Leue. Syntactic detection of process divergence and non-local choice in message sequence charts. In E. Brinksma, editor, *Proceedings of TACAS'97*, LNCS no 1217, pp 259 – 274, Enschede,The Netherlands, April 1997. Springer-Verlag.

[3] M.A. Bernarczyk, L. Bernardinello, B. Caillaud, W. Pawlowski, and L. Pomello. Modular

system development with pullbacks. Technical Report 4828, INRIA, May 2003.

[4] B. Caillaud and L. Hélouët. A categorical approach to hmsc composition. Research report, IRISA/INRIA, Nov. 2003. to appear.

[5] A. Engels. Message refinement: Describing multi-level protocols in MSC. In Y. Lahav, A. Wolisz, J. Fischer, and E. Holz, editors, *Proc. of SAM98:1st conference on SDL and MSC*, number 104 in Informatik-Berichte, pp. 67–74, Berlin, Germany, June 1998.

[6] B. Genest, L. Hélouët, and A. Muscholl. High-level message sequence charts and projections. In *Proceedings of CONCUR'2003*, 2003.

[7] E. Gunter, A. Muscholl, and D. Peled. Compositional message sequence charts. In *Proc. of TACAS'01*, number 2031 in Lecture Notes in Computer Science, pp. 496–511, 2001.

[8] D. Harel and R. Marelly. *Come, Let's Play : Scenario-Based Programming Using LSCs and the Play-Engine.* Springer, 2003.

[9] L. Hélouët and C. Jard. Conditions for synthesis of communicating automata from hmscs. In Axel Rennoch (Eds.) Stefania Gnesi, Ina Schieferdecker, editor, *Proc. of FMICS'2000*, Berlin, April 2000. GMD FOKUS. http://www.gmd.de/publications/report/0091.

[10] L. Hélouët, C. Jard, and B. Caillaud. An effective equivalence for sets of scenarios represented by hmscs. Technical Report 3499, INRIA, September 1998. ftp://ftp.inria.fr/INRIA/publication/RR/RR-3499.ps.gz.

[11] ITU-TS. *ITU-TS Recommendation Z.120: Message Sequence Chart (MSC).* ITU-TS, Geneva, September 1999.

[12] J.P. Katoen and L. Lambert. Pomsets for message sequence charts. In *Proc. of SAM98:1st conference on SDL and MSC*, pp. 281–290, Berlin, Juin 1998.

[13] Martin Leucker, P. Madhusudan, and Supratik Mukhopadhyay. Dynamic message sequence charts. In M. Agrawal and A. Seth, editors, *Proc. of FSTTCS'02*, LNCS no2556. Springer, December 2002.

[14] S. Leue and P. Ladkin. Four issues concerning the semantics of message flow graphs. Technical report, INRIA Lorraine, 1994.

[15] Sjouke Mauw and Michel A. Reniers. Refinement in interworkings. In *International Conference on Concurrency Theory*, pp. 671–686, 1996.

[16] M. Reniers. *Message Sequence Charts: Syntax and Semantics.* PhD thesis, Eindhoven University of Technology, 1998.

[17] E. Rudolph, P. Graubmann, and J. Grabowski. Message Sequence Chart: composition techniques versus OO-techniques - 'tema con variazioni'. In R. Bræk and A. Sarma, editors, *SDL'95 with MSC in CASE*, Proc. of 7th SDL Forum, pp 77–88, Oslo, 1995. Amsterdam, North-Holland.

[18] E. Rudolph, P. Graubmann, and J. Grabowski. Tutorial on Message Sequence Charts. *Computer Networks and ISDN Systems*, 28(12):1629–1641, 1996.