From Automata Networks to HMSCs: a Reverse Model Engineering Perspective

Thomas Chatain
IRISA/ENS Cachan-Bretagne, Campus de Beaulieu,
F-35042 Rennes cedex, France
Thomas.Chatain@irisa.fr

Loïc Hélouët
IRISA/INRIA, Campus de Beaulieu,
F-35042 Rennes cedex, France
Loic.Helouet@irisa.fr

Claude Jard
IRISA/ENS Cachan-Bretagne, Campus de Ker-Lann,
F-35170 Bruz cedex, France
Claude.Jard@bretagne.ens-cachan.fr

Abstract. This paper considers the problem of automatic abstraction, from a low-level model given in term of network of interacting automata to a high-level message sequence chart. This allows the designer to play in a coherent way with the local and global views of a system, and opens new perspectives in reverse model engineering. Our technique is based on a partial order semantics of synchronous parallel automata and the construction of a finite complete prefix of an event-structure coding all the behaviors. We present the models and algorithms. The examples presented in the paper have been processed by a small software prototype we have implemented.

1 Introduction

Designing a distributed application is a complex task. At the final stage of the modeling, once the different architectural decisions have been made, designers usually obtain a set of communicating sequential components. During earlier stages of software development, designers use more abstract and visual representations such as scenarios. For instance, Message Sequence Charts (MSCs) [9] are an appealing visual formalism to capture system requirements. They are particularly suited for describing scenarios of distributed telecommunication software [7]. Several variants of MSCs appear in the literature (sequence diagrams , message flow diagrams, object interaction diagrams, Live Sequence Charts) and are used in a number of software engineering methodologies including UML [8]. They provide the designer with a global view of the dynamic behavior of the system, given in a declarative manner.

However, there is often a gap between the local view defined as sequential components and the more global view described by scenarios. Some scenarios cannot be implemented by sequential machines, and some compositions of sequential machines do not have finite representation in terms of MSCs. This is why a lot of recent works have been developed to automatically generate communicating automata (at least a skeleton) from MSCs [1,5] in the context of a top-down design methodology. Obviously, building a bridge in the opposite direction is also an interesting problem, as it would allow designers to play freely with any style of specification (global declarative or distributed imperative) while preserving the coherence of both views. A solution to this problem could also be the basis of another important challenge called "aspect modeling", in which a new feature described as a set of scenarios can be added safely to an already existing model of communicating machines. This will imply sophisticated formal techniques, since the required transformations modify dramatically the structure of the automata.

This context motivates our work on some "reverse distributed model engineering". We begin with simple models, which are networks of synchronous parallel finite state automata for the imperative aspect, and MSCs for the declarative aspect. The problem is thus to automatically obtain a MSC from an automata network, which codes all the runs of the system, runs being defined as partial orders of transition occurrences. The finiteness of the automata and the synchronous communication ensure that such a transformation is possible. This question has already been addressed from the theoretical point of view in term of formal languages in [3]. They show that any single Büchi automaton with a structural property, called diamond, and with all its states accepting, is able to generate the language of a bounded MSC. However, this problem is undecidable for asynchronous communicating finite state machines. This justifies our choice to consider synchronous networks and to propose an original algorithm to produce a concrete MSC, as readable as possible. Figure 1 shows an example of such network, which consists of two automata A_0 and A_1, synchronized on their common event x. Figure 1 gives the corresponding MSC we would like to compute. Notice that the MSC graph is complex due to the fact that this example was designed to show all the tricky aspects of the transformation. A more realistic example is treated in Section 4.

We will use the notion of unfolding, and the fact it can be finitely generated by a finite complete prefix. This is based on the unfolding theory, as presented in [4,2]. In the paper, we adopt nevertheless a direct approach, without using Petri nets as usual, in order to avoid to introduce a new intermediary formal model. The question of using the finite prefix as a generator of the unfolding is also new up to our knowledge.

The rest of the paper is organized as follows. Section 2 defines formally automata networks, MSCs and the notion of runs. The next section 3 is devoted to the generation of possible runs by the construction of a finite complete prefix of the unfolding. Section 4 presents how the MSC automaton and the referenced basic MSCs are extracted from the prefix. We conclude by a discussion summarizing the approach and proposing a few perspectives. All the proofs of the propositions and theorems are available in the research report [10].

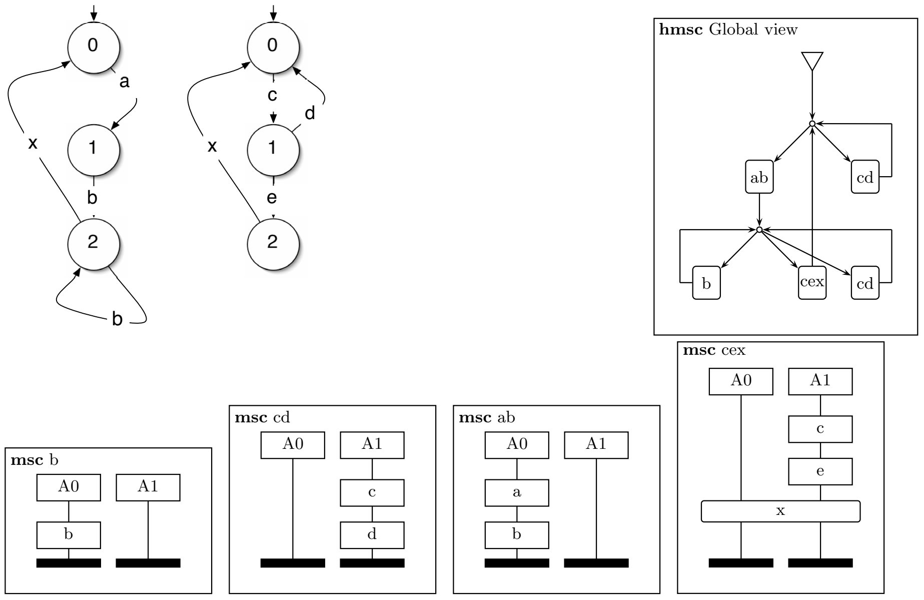

Fig. 1. A network of two synchronized automata and its scenario view.

2 Definition of Automata Networks and MSCs

2.1 Networks

An *initialized labelled automaton* is a tuple $A = \langle S, \Sigma, \rightarrow, s^0 \rangle$ where S is a finite set of *states*, Σ is a set of labels, $\rightarrow \subseteq S \times \Sigma \times S$ is a set of *labelled transitions*, and $s^0 \in S$ is the initial state. For a transition $t = (s, a, s') \in \rightarrow$, we denote $\alpha(t) \overset{\text{def}}{=} s$ its *source*, $\beta(t) \overset{\text{def}}{=} s'$ its *target*, and $\lambda(t) \overset{\text{def}}{=} a$ its *label.*

$I \overset{\text{def}}{=} \{1, \ldots, n\}$ denotes a finite set of indices. We consider the *synchronous parallel composition of the initialized labelled automata* $A_i = \langle S_i, \Sigma_i, \rightarrow_i, s_i^0 \rangle_{i \in I}$

The network of Figure 1 is formally defined by:

$$
\begin{array}{ll}
S_0 = \{0,1,2\} & S_1 = \{0,1,2\} \\
\Sigma_0 = \{a,b,x\} & \Sigma_1 = \{c,d,e,x\} \\
s_0^0 = 0 & s_1^0 = 0 \\
\multicolumn{2}{l}{\rightarrow_0 = \{(0,a,1),(1,b,2),(2,x,0),(2,b,2)\}} \\
\multicolumn{2}{l}{\rightarrow_1 = \{(0,c,1),(1,e,2),(2,x,0),(1,d,0)\}}
\end{array}
$$

In an interleaving semantics, the network behavior is defined as the (global) initialized labelled automaton $A = \langle S, \Sigma, \rightarrow, s^0 \rangle$ where:

- $S \overset{\text{def}}{=} S_1 \times \cdots \times S_n$
- $\Sigma \overset{\text{def}}{=} \bigcup_{i \in I} \Sigma_i$
- $((s_i)_{i\in I}, a, (s'_i)_{i \in I}) \in \rightarrow$ iff $\begin{cases} \forall i \in \{1,\dots,n\} & \begin{cases} (s_i, a, s'_i) \in \rightarrow_i \\ \vee\, (s_i = s'_i \ \wedge \ a \notin \Sigma_i) \end{cases} \\ \wedge\ \exists i \in \{1,\dots,n\} & (s_i, a, s'_i) \in \rightarrow_i \end{cases}$
- $s^0 \overset{\text{def}}{=} (s^0_1, \dots, s^0_n)$

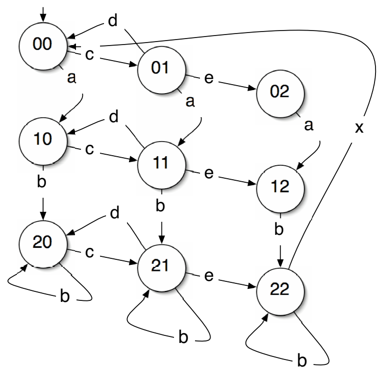

Fig. 2. The synchronized product.

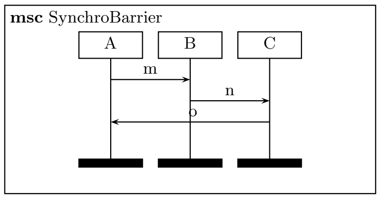

Fig. 3. bMSC representation of rendez-vous

Intuitively, we force the automata to evolve synchronously when they execute a transition labelled by the same name. In the other case, they evolve independently. Figure 2 shows the product automaton of our example. Sequential runs are the different paths in the graph of the product automaton. Unfortunately, this notion of run does not enlight the causal relations between the different occurrences of transitions (seen as atomic events), as done in MSCs. In our context, the right notion of run is the partial ordering of events that have occurred. Hence, runs of a system will be defined as basic MSCs.

2.2 Message Sequence Charts

MSCs are composed of basic scenarios (or bMSCs), that depict interactions among several objects. These interactions are then composed hierarchically by means of operators (loop, choice, sequence, ...). For the sake of simplicity, we will only consider a single hierarchical level. Interactions in the automata networks we consider are synchronous (i.e. Rendez-vous communication): they are blocking, and involve several participants. For this reason, communications in bMSCs will be represented by references to other bMSCs describing how a communication mechanism is implemented. Such Rendez-vous can be implemented using a synchronization barrier, as depicted in Figure 3. In MSCs, referencing inside a diagram is allowed by inline expressions. Here, we will only consider references to simple bMSCs depicting communications among a given set of components. We do not allow reference nesting, and will not use inline expressions with opt, alt or loop.

In our framework, a bMSC is defined as a finite set of events. Each event is represented as the vector of its predecessors on each instance. The absence of predecessor on an instance is denoted by the null event $\bullet$. We associate a label to each event, which will serve to note the corresponding transition of the automata. For example, considering a system with three instances, the event e_3 denoted by $((e_1,(1,a,2)),\bullet,(e_2,(3,a,4)))$ is a synchronization event between the first and the third instance, and having the events e_1 and e_2 as immediate predecessors on these instances. There is no immediate predecessor on the second instance since it does not participate in the synchronization. The labels are $(1,a,2)$ and $(3,a,4)$, denoting for instance the transitions to synchronize in an automata network. Formally, a *bMSC* over a set of instances I is a tuple $B=(E,\Sigma,A,\Theta)$, where $E=\{(e_i,\sigma)_{i\in I},\sigma\in\Sigma\}$ is a set of events such that each $e_i\in\{\bullet\}\cup E\times\Sigma$. E contains *local events* (events such that $|\{e_i\neq\bullet\}|=1$) and *interactions* (events such that $|\{e_i\neq\bullet\}|>1$). Σ is a local alphabet, A is an alphabet of local actions and interaction names, and $\Theta:\Sigma\longrightarrow A$ assigns a global name to events.

When $f_i=(e,\sigma)$, we denote $\pi_i(f)=e$. We will say that e is a predecessor of f, and write $e\rightarrow f$ when $\exists i\in I$ such that $\pi_i(f)=e$. E also contains a specific event $\perp=(\bullet,\ldots,\bullet)_{i\in I}$ called the *initial event* that has no predecessor. We will say that an event is *minimal* in a bMSC iff $\perp$ is the unique predecessor of all its components. A bMSC must also satisfy the following properties :

i) the reflexive and transitive closure $\rightarrow^*$ of $\rightarrow$ is a partial order.
ii) `(synchronization)` $\forall e=(e_i)_{i\in I}\in E$, we require that $\exists! a,\forall i\in I, e_i\neq\bullet\Longrightarrow\Theta(\sigma_i)=a$. This property means that all components participating to an event must synchronize.
iii) `(local sequencing)` $\forall i\in I,\forall e\in E, e_i\neq\bullet\Longrightarrow\pi_i(e)=\perp$ or $(\pi_i(e))_i\neq\bullet$
iv) `(no choice)` $\forall(e,e')\in E^2,\forall i\in I, e\neq e'\Longrightarrow\pi_i(e)\neq\pi_i(e')$. This property forbids the introduction of choices in a bMSC.

bMSCS are good candidates to model causal relations in runs of a distributed system. *Causality* between events is defined by $\rightarrow^*$. When neither $e\rightarrow^* e'$, nor $e'\rightarrow^* e$, we will say that e and e' are independent (or *concurrent*). The set of minimal events in B w.r.t $\rightarrow^*$ is denoted by $min(E)$. We will say that an event is minimal for an instance $i\in I$ if the predecessor event on component i is $\perp$. It is maximal for this instance if it is not a predecessor event for an event on this instance. The minimal (resp. maximal) event on instance i (when it is defined) will be denoted by $min_i(E)$(resp. $max_i(E)$). A bMSC $B1$ is a prefix of a bMSC $B2$ if and only if $E_1 \subseteq E_2$ and $\forall e \in E_1, \Theta_1(e) = \Theta_2(e)$. The empty bMSC is the tuple $B_\emptyset = (\{\perp\}, \emptyset, \emptyset, \emptyset)$. Figure 4 is an example of bMSC. This bMSC defines the behavior of 2 instances A0 and A1. Events a, b, c, e are local actions, and reference x represents a synchronous interaction between $A0$ and $A1$.

The *sequential composition* of two bMSCs $B1 = (E_1, \Sigma_1, A_1, \Theta_1)$, $B2 = (E_2, \Sigma_2, A_2, \Theta_2)$ is the bMSC $B = (E, \Sigma_1 \cup \Sigma_2, A_1 \cup A_2, \Theta)$, where :

$$\begin{aligned} E = \; & E_1 \cup \Big(E_2 \setminus \big(\{\perp\} \cup \{min_i(E_2) | i \in I\} \big) \Big) \\ & \cup \left\{ \begin{array}{l} (e'_1, \dots e'_n) | \exists i \in I, \exists (e_1, \dots, e_n) \in min_i(E_2) \\ \wedge \forall j \in I, e'_j = \begin{cases} (max_j(E_1), \sigma) \text{ if } e_j = (\perp, \sigma) \\ e_j \text{ otherwise} \end{cases} \end{array} \right\} \\ \Theta(\sigma) = \; & \Theta_1(\sigma) \text{ if } \sigma \in \Sigma_1, \Theta_2(\sigma) \text{ otherwise} \end{aligned}$$

More intuitively, sequential composition merges two bMSCs along their common instances axes by addition of an ordering between the last event on each instance of B_1 and the first event on the same instance in B_2.

A *High-level Message Sequence Chart* (HMSC) is a tuple $H = (N, \rightarrow, \mathcal{M}, n_0, F)$, where N is a set of nodes, $\rightarrow \subseteq N \times \mathcal{M} \times N$ is a transition relation, $\mathcal{M}$ is a set of bMSCs, n_0 is the initial node, and F is a set of accepting nodes. HMSCs can be considered as finite state automata labelled by bMSCs. A HMSC H defines a set of paths $\mathcal{P}_H$. For a given path $p = n_0 \xrightarrow{M_1} n_1 \xrightarrow{M_2} n_2 \dots \xrightarrow{M_k} n_k \in \mathcal{P}_H$ we can associate a bMSC $B_p = M_1 \circ M_2 \circ \dots \circ M_k$. The runs of a HMSC H are the prefixes of all bMSCs generated by paths of H. The run associated to the empty path is $B_\emptyset$.

2.3 Runs as Partial Orders

A *run* of an automata network $A_i = \langle S_i, \Sigma_i, \rightarrow_i, s_i^0 \rangle_{i \in I}$ is defined as a bMSC $M = (E, \Sigma, A, \Theta)$, with the following properties:

i) $\Sigma = \bigcup_{i \in I} \longrightarrow_i$. Hence, for an event $e = (e_i)_{i \in I}$, each e_i is of the form $e_i = (e', t)$, and we will denote $\tau_i(e) \overset{\text{def}}{=} t$, $\alpha_i(e) \overset{\text{def}}{=} \alpha(t)$ and $\beta_i(e) \overset{\text{def}}{=} \beta(t)$. We define $\beta_i(\perp) \overset{\text{def}}{=} s_i^0$.
ii) $A = \bigcup_{i \in I} \Sigma_i$.
iii) $\Theta(t) = \lambda(t)$
iv) `(local sequencing)` $\forall i \in I \quad e_i \neq \bullet \implies \alpha_i(e) = \beta_i(\pi_i(e))$

As Σ, A, Θ are implicit for a given set of events E, we will often denote a bMSC $B = (E, \Sigma, A, \Theta)$ by its set of events E. Intuitively, an event $e \neq \perp$ represents the synchronization of actions of the automaton A_i such that $e_i \neq \bullet$; and $e_i = (e', t)$ means that the local action on automaton A_i is t, and the previous action that concerned the automaton A_i was e'. Note that property *iii)* implies that for a given component $i \in I$ and for any chain $\perp \longrightarrow e^1 = (\perp, t_1) \longrightarrow e^2 = (e^1, t_2) \ldots \longrightarrow e^k = (e^{k-1}, t_k)$ such that $\forall j \in 1..k, {e^j}_i \neq \bullet$, the sequence $t_1.t_2 \ldots t_k$ is a path of automaton A_i.

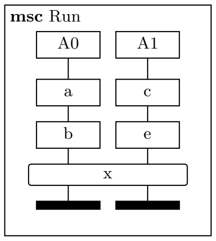

Fig. 4. A run as defined as a bMSC with inline references.

This run corresponds to the concatenation of the bMSCs AB and CEX of Figure 1. Its events are:

$$\begin{array}{ll}
0 = \perp, & 3 = ((1, (1, b, 2)), \bullet), \\
1 = ((0, (0, a, 1)), \bullet), & 4 = (\bullet, (2, (1, e, 2))), \\
2 = (\bullet, (0, (0, c, 1))), & 5 = ((3, (2, x, 0)), (4, (2, x, 0)))
\end{array}$$

The question now is to represent all the possible runs. This is the role of the *unfolding*, which superimposes all the runs, shares the common prefixes and distinguishes the different histories using the notion of *conflict*.

3 Generation of Runs

3.1 Unfolding

We consider the union of all possible runs, forming a new event set E. The absence of choices is no more guaranted. This is why we define the conflict relation $\#$ on the events as follows:

$$e \# e' \quad \text{iff} \quad \exists f, f' \in E \quad \begin{cases} f \neq f' \\ f \rightarrow^* e \\ f' \rightarrow^* e' \\ \exists i \in I \quad \pi_i(f) = \pi_i(f') \end{cases}$$

Informally, two events are in conflict if they have a common ancestor event that branches on a **same** instance.

The *unfolding* of the synchronous parallel composition of the initialized labelled automata $A_i = \langle S_i, \Sigma_i, \rightarrow_i, s_i^0 \rangle_{i \in I}$ is the set U of all events that are not in self-conflict: $U \overset{\text{def}}{=} \{e \in E \mid \neg(e \# e)\}$. Graphically, we draw a circle for each event, and an arc from e' to e, labelled by i each time $e_i = (e', t)$. Figure 5 shows the shape of the unfolding of the network of Figure 1.

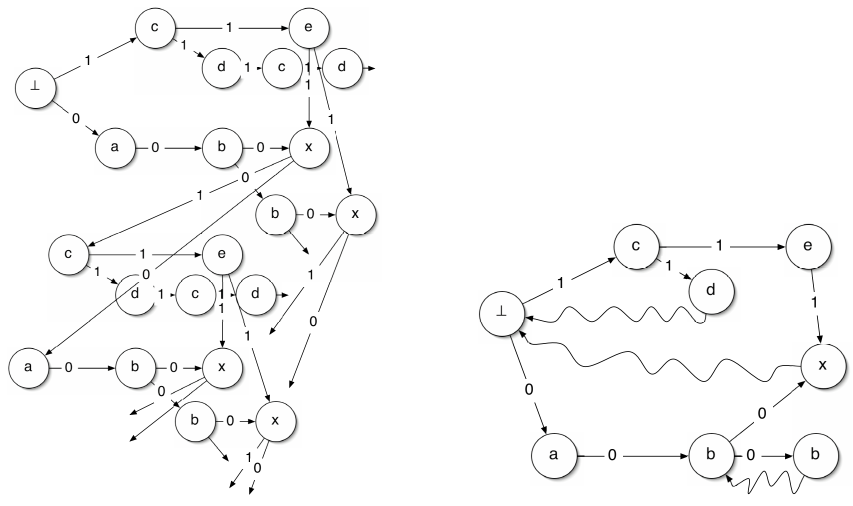

Fig. 5. The unfolding of the network of Figure 1 and its finite complete prefix.

A (finite) run (also called a *configuration*) of the unfolding is a bMSC $B = (F, \Sigma, A, \Theta)$ where Σ, A, Θ are defined as usual, and F is a finite subset of E which is conflict-free and causally closed, i.e: $\begin{cases} \forall e, f \in F \quad \neg(e \# f) \\ \forall f \in F \quad \forall e \in E \quad e \rightarrow^* f \implies e \in F \end{cases}$

Proposition 1. *The unfolding contains all the possible runs.*

3.2 A Trivial Solution for MSC Extraction

As explained previously, our goal is to compute a global declarative view defined as a MSC from a distributed imperative view of a distributed system given by a network of automata. The existence of a trivial solution to this problem is guaranteed by the following proposition.

Proposition 2. *Let $A = (S, \Sigma, \longrightarrow, s^0)$ be the global initialized labelled automaton obtained by synchronous product of automata $(A_i)_{i \in I}$. Let $H = (S, b(\Sigma), \longrightarrow', s^0, S)$ be the HMSC where $b(\sigma)$ is the bMSC containing a single local action performed by an automaton or a single interaction performed by all automata involved in a synchronous communication, and $\longrightarrow' = \{(n, b(\sigma), n') | (n, \sigma, n') \in \longrightarrow)\}$. Then, the set of runs of H and the set of runs of $(A_i)_{i \in I}$ are equivalent.*

We can imagine the resulting HMSC by having a look on Figure 2. Clearly, it does not fulfill our goal of reverse model engineering. We must try to fill as much as possible the bMSCs.

3.3 Finite Complete Prefix

The unfolding U of an automata network is an infinite structure. However, it is possible to work on a finite representation of U called a *finite complete prefix.*

For a configuration $c \subseteq U$ and for an automaton $i \in I$, we define the *last event* $\uparrow_i c$ that concerned i in c as the event $f \in c$ such that:

$$(f_i \neq \bullet \ \vee \ f = \bot) \ \wedge \ \nexists f' \in c \quad \pi_i(f') = f$$

Proposition 3. *For a configuration $c \subseteq U$ and for an automaton $i \in I$, $\uparrow_i c$ is unique.*

We denote $\uparrow c$, the vector $(\uparrow_i c)_{i \in I}$ of last events. The *global state* vector associated with a configuration c is also defined as the states of each automaton after having performed the event $\uparrow_i c$, i.e.

$$\mathit{GState}(c) \stackrel{\text{def}}{=} (\beta_i(\uparrow_i c))_{i \in I}$$

For all $e \in U$, $\lceil e \rceil \stackrel{\text{def}}{=} \{f \in E \mid f \rightarrow^* e\}$ is a configuration, called the *local configuration of* e. We define the set C of *cut-off events* of an unfolding as:

$$e \in C \quad \text{iff} \quad \exists f \in \lceil e \rceil \setminus \{e\} \quad \mathit{GState}(\lceil f \rceil) = \mathit{GState}(\lceil e \rceil)$$

Actually the event f for a cut-off event e is generally not unique. We define the *regeneration configuration*, denoted ∂e of a cut-off event $e \in C$ as the intersection of the local configurations $\lceil f \rceil$ of the events $f \in \lceil e \rceil \setminus \{e\}$ such that $\mathit{GState}(\lceil f \rceil) = \mathit{GState}(\lceil e \rceil)$:

$$\partial e \stackrel{\text{def}}{=} \bigcup_{\substack{f \in \lceil e \rceil \setminus \{e\} \\ \mathit{GState}(\lceil f \rceil) = \mathit{GState}(\lceil e \rceil)}} \lceil f \rceil.$$

Proposition 4. *For all $e \in C$, $\mathit{GState}(\partial e) = \mathit{GState}(\lceil e \rceil)$.*

The set $\{e \in U \mid \nexists f \in C \quad f \rightarrow^+ e\}$ is a *finite complete prefix* of the unfolding U.

Theorem 1. *The finite complete prefix is a finite generator of the unfolding.*

The following algorithm computes the finite complete prefix U.

```
Initialization

 1. create the initial event: $U = \bot = (\bullet)_{i \in I}$, with $GState(\{\bot\}) = (s_i^0)_{i \in I}$;
 2. $C \leftarrow \emptyset$;

Repeat until deadlock

 1. select a tuple $(x_i)_{i \in I}$ where $x_i \in \{\bullet\} \cup \rightarrow_i$, such that:
    – $\exists a \in \Sigma \quad \forall i \in I \quad \begin{cases} x_i = \bullet \implies a \notin \Sigma_i \\ x_i \neq \bullet \implies \lambda_i(x_i) = a \end{cases}$
    – $\forall i \in I \quad x_i \neq \bullet \implies \exists e'_i \in U \setminus C, \quad \beta_i(e'_i) = \alpha_i(x_i)$
 2. build the event $e = (e_i)_{i \in I}$, where $\begin{cases} e_i = (e'_i, x_i) & \text{if} \quad x_i \neq \bullet \\ e_i = \bullet & \text{otherwise} \end{cases}$
 3. if $e \notin U \;\wedge\; \neg(e \# e) \quad \text{in} \quad U \cup \{e\}$
    – $U \leftarrow U \cup \{e\}$;
    – if $\exists e' \in \lceil e \rceil$ with $GState(\lceil e' \rceil) = GState(\lceil e \rceil)$:
      then $C \leftarrow C \cup \{e\}$;
           $\partial e \leftarrow \bigcup_{\substack{f \in \lceil e \rceil \setminus \{e\} \\ GState(\lceil f \rceil) = GState(\lceil e \rceil)}} \lceil f \rceil$
```

Figure 5 (right) shows the prefix obtained from our example. Let us consider the event e, labelled by x. It is a cut-off event. Its regeneration configuration ∂e is $\{\bot\}$. This is graphically represented by an oscillating arrow.

4 MSC Extraction

MSC extraction starts with the abstraction of the prefix. Intuitively, for a given finite complete prefix, we define X as a subset of configurations that contains the local configuration of the cut-off events, their regeneration configuration, the local configuration of the terminal events, and that is closed under intersection. X can be projected on each instance in order to obtain a network of "abstract automata". The product forms the HMSC automaton. Basic MSCs are obtained by considering all the events occuring in an interval between two configurations of X, and transitions are deduced from configurations inclusion.

We denote by P the finite complete prefix of the unfolding U of an automata network. An event e is *terminal* if there exists no $f \in U$ such that $e \rightarrow f$. Let X be the set of configurations inductively defined as:

- $\{\bot\} \in X$
- for all e cut-off event, $\lceil e \rceil \in X \wedge \partial e \in X$;
- for all terminal event e, $\lceil e \rceil \in X$;
- for all $x, x' \in X$, $x \cup x'$ is a configuration $\implies x \cap x' \in X$.

We denote by $Y \stackrel{\text{def}}{=} \{\lceil e \rceil \mid e \in C\}$ the local configurations of cut-off events.

For all $x \in X$, let us define $E_x \stackrel{\text{def}}{=} x \setminus \bigcup_{\substack{x' \in X \\ x' \subsetneq x}} x'$. The sets E_x are subsets of elements that are not contained in any smaller configuration of X. They define the bMSCs that will be extracted from the prefix.

For all $x \in X$, the sets $E_{x'}$ with $x' \in X,\ x' \subseteq x$ are a partition of x. For all event $e \in x$ we denote $E^{-1}(e, x)$ the unique configuration $x' \in X$ such that $x' \subseteq x$ and $e \in E_{x'}$. Let us define an abstraction of the prefix P, where the elements of X play the role of "macro-events". For all $i \in I$ we define the set X_i of macro-events that concern i as:

$$X_i \stackrel{\text{def}}{=} \{x \in X \mid \exists e \in E_x, e_i \neq \bullet \vee e = \bot\}$$

For the example of Figure 5, we have:

- $X = \{\bot, \bot cd, \bot ab, \bot abcex, \bot abb\}$
- $E_\bot = \bot, \quad E_{\bot cd} = cd, \quad E_{\bot ab} = ab, \quad E_{\bot abcex} = cex, \quad E_{\bot abb} = b$
- $X_0 = \{\bot, \bot ab, \bot abcex, \bot abb\}, \quad X_1 = \{\bot, \bot cd, \bot abcex\}$
- $Y = \{\bot cd, \bot abcex, \bot abb\}$

For all $i \in I$ and for all $x \in X_i \setminus \{\{\bot\}\}$, the last event that concerned i in $x \setminus E_x$ is $\uparrow_i (x \setminus E_x)$. We define the macro-event that immediately precedes x on i as $\pi_i(x) \stackrel{\text{def}}{=} E^{-1}(\uparrow_i (x \setminus E_x), x)$.

Using this definition, for each $i \in I$ we can now define the initialized labelled macro-automaton

$$\mathcal{A}_i \stackrel{\text{def}}{=} \langle X_i \setminus Y, \{E_x \mid x \in X_i\}, \rightarrow_i, \{\bot\}\rangle$$

where

$$\rightarrow_i = \begin{array}{l} \{(\pi_i(x), E_x, x) \mid x \in X_i \setminus \{\{\bot\}\} \wedge x \notin Y\} \\ \cup\, \{(\pi_i(x), E_x, E^{-1}(\uparrow_i \partial e, \partial e) \mid x \in X_i \wedge x = \lceil e \rceil \text{ with } e \text{ cut-off event}\} \end{array}$$

Figure 6 shows the network of macro-automata obtained from our example. Let $\mathcal{A} = \langle S, \Sigma, \longrightarrow, s^0\rangle$ be the synchronous product $\mathcal{A}_1 \times \mathcal{A}_2 \times \cdots \times \mathcal{A}_n$. The HMSC extracted from a finite complete prefix P is defined as $H_P = (S, \longrightarrow', b(\Sigma), s^0, S)$, where $\forall \sigma \in \Sigma, b(\sigma)$ is the bMSC obtained by adding $\bot$ as predecessor of all minimal events to σ, and $\longrightarrow' = \{(s, b(\sigma), s') | \exists s, \sigma, s') \in \longrightarrow\}$. For our example, the HMSC computed from the synchronous product in Figure 6 is the resulting HMSC of Figure 1 announced in the beginning.

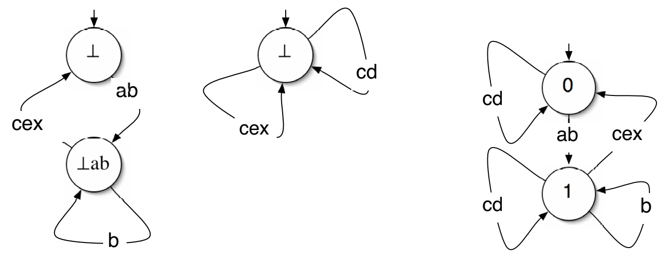

Fig. 6. The network of macro-automata and its product

Theorem 2. *Let P be a finite complete prefix of an automata network unfolding, and let $(A_i)_{i \in I}$ be the set of "macro-automata" obtained from P. Let H be the HMSC obtained from the synchronous product $(A_i)_{i \in I}$. The runs of $(A_i)_{i \in I}$ and the runs of H are equivalent.*

Let us consider the more realistic example shown in Figure 7 (left). It is a simple connection and release protocol between two peers. The two peers (sender and receiver) are presented on top of the figure. They are connected through channels of size one. The automata of channels are given at the bottom of the figure. In this protocol, the sender can initiate a connection by sending the $Creq$ message ("!" and "?" characters denote the send and receive actions respectively). After that, it can decide locally to close the connection by sending the message $Dreq$, or receives the message $Ddreq$ indicating that a distant disconnection has been made by the receiver. In case of collision (reception of $Ddreq$ in state 2), the connection is also closed. On the receiver side, after having received the $Creq$, the received may decide to close the connection by sending the distant disconnection message $Ddreq$. If not, the $Dreq$ message is received in state 1. In that case, it is required that the receiver alerts the sender by the $Dconf$ message to allow it to close locally the connection. Note that in case of collision, it is possible to receive a message $Dreq$ in state 0, which must be skipped.

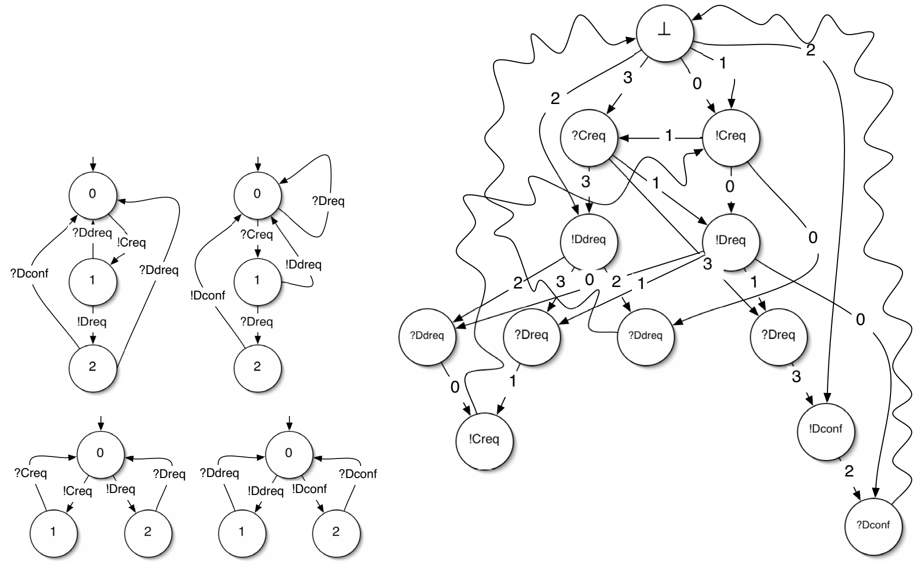

Fig. 7. The Connect-Disconnect protocol with channels of size one and its prefix.

Figure 7 (right) shows the prefix of the unfolding of this example. We show three cut-off events, corresponding to the three basic patterns of the protocol, which are local disconnection, distant disconnection and collision. The MSC view produced by our method is shown in Figure 8.

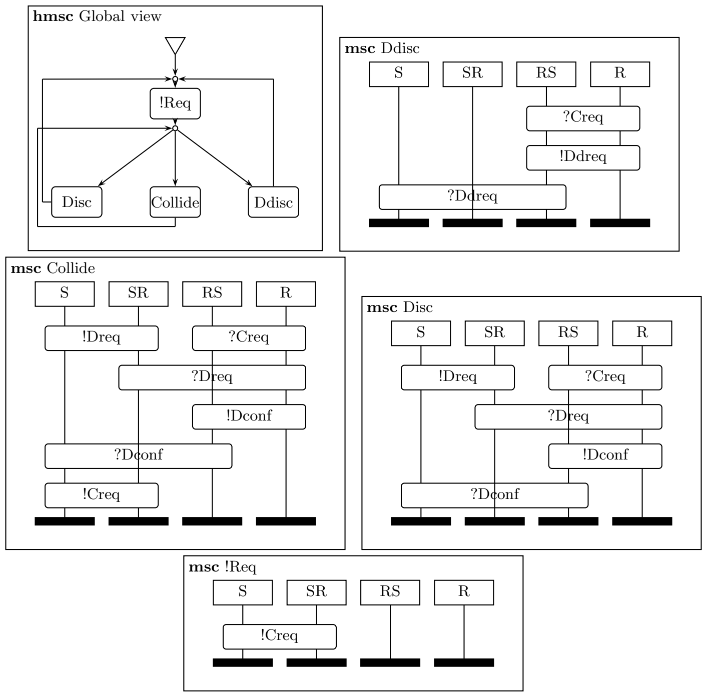

Fig. 8. MSC extracted from Automata of Figure 7.

5 Discussion

We have addressed the problem of reverse model engineering, and more precisely the automatic translation of synchronous networks of finite automata into message sequence charts. A trivial solution is to build the product automaton and to interpret transition labels as basic MSCs. Unfortunately, this degenerated MSC does not fulfill the requirements of reverse engineering, which are to present the concurrent histories of the system using as much as possible a partial order view.

This work introduces new techniques that permit to recover a global partial-order based view of a system described by composition of sequential components, and hence seems relevant for reverse model engineering. The main algorithm is the unfolding of the network of automata. It computes the set of all partial order runs. Thanks to the finiteness of the system, this set is finitely generated by a prefix. From this prefix, we showed a way to extract basic partial order patterns (bMSCs). The removal of these patterns in the prefix, followed by a local projection lead to an abstract network of “macro-automata”. A HMSC with the same behavior as the initial automata network can then be produced by computing the product of macro automata. An alternative could be to consider a parallel construct in the HMSC, as proposed for instance in netcharts [6].

The algorithms have been implemented in a software prototype (a few thousand of lines of C-code). The next step will be to be able to deal with more complex systems. First, we have to relax the synchronous assumption to take benefit of the asynchronous communication in MSCs. We think it is possible to find a class of systems in which synchronous communication can be safely replaced by an asynchronous one without changing the set of partial runs. Let us recall nevertheless that asynchronous communicating automata and MSC define uncomparable languages. This means that a translation of automata into MSC may not exists. Furthermore, deciding whether a network of asynchronous automata defines a MSC language is an undecidable problem. Hence, to be effective in an asynchronous framework, our approach will necessarily apply to a restricted class of automata. Secondly, the MSCs we obtain are dependent of two things: the definition of cut-off events and the definition of configurations that are extracted from the finite complete prefix. So far, an event is a cut off event if its configuration has already been seen in its causal past. This leads to some duplications of events in the finite complete prefix. The definition of cut-off events can be refined using the adequate orders proposed by J. Esparza in [2]. This enhancement will reduce the duplication of events. Concerning the definition of configurations to extract (the X set), we can decide to share more or less common prefixes in the bMSCs, and find a tradeoff between the number of duplications and the size of the considered bMSCs. This could be parameterized.

References

1. L. Hélouët and C. Jard. Conditions for Synthesis of Communicating Automata from HMSCs, 5th International Workshop on Formal Methods for Industrial Critical Systems (FMICS), ARE. Stefania-Gnesi, I. Schieferdecker (ed), GMD FOKUS, Apr. 2000.
2. J. Esparza and S. Römer. An Unfolding Algorithm for Synchronous Products of Transition Systems, Proc. of Concur 1999, Lecture Notes in Computer Science 1664, pp. 2-20, 1999.
3. A. Muscholl and D. Peled. From Finite State Communication Protocols to High-Level Message Sequence Charts, Proc. of ICALP'01, Lecture Notes in Computer Science 2076, pp. 720-731, 2001.
4. K. Mac Millan. A Technique of State Space Search Based on Unfolding, Journal of Formal Methods and System Design, 9, 1-22 (1992), Kluwer.
5. M. Abdallah, F. Khendec, and G. Butler. New Results on Deriving SDL Specifications from MSCs, Proc. of 9th SDL Forum, pp. 51-66, Montreal.
6. M. Mukund, K.N. Kumar, and P.S. Thiagarajan. Netcharts: Bridging the Gap between HMSCs and Executable Specifications, Proc. of Concur 2003, Lecture Notes in Computer Science 2761, pp. 296-310, 2003.
7. E. Rudolph, O. Graubmann and J. Grabowski. Tutorial on Message Sequence Charts, Computer Networks and ISDN Systems - SDL and MSC, Vol. 28, 1996.
8. G. Booch, I. Jacobson and J. Rumbaugh. Unified Modeling Language User Guide, Addison-Wesley, 1997.

9. ITU, Message Sequence Charts, standard Z.120, 2000.
10. C. Jard, L. Hélouët and T. Chatain. From Automata Networks to HMSCs: a Reverse Model Engineering Perspective, INRIA/IRISA Research Report, Aug. 2005, 22 pages.