Semantic-based weaving of scenarios

Jacques Klein, Loïc Hélouët, Jean-Marc Jézéquel

Semantic-based Weaving of Scenarios*

Jacques Klein
IRISA/INRIA
Campus de Beaulieu
35042 Rennes Cedex, France
jacques.klein@irisa.fr

Loïc Hélouët
IRISA/INRIA
Campus de Beaulieu
35042 Rennes cedex, France
loic.helouet@irisa.fr

Jean-Marc Jézéquel
IRISA/ Université de Rennes 1
Campus de Beaulieu
35042 Rennes Cedex, France
jezequel@irisa.fr

ABSTRACT

The notion of aspect looks promising for handling crosscutting concerns earlier in the software life-cycle, up from programming to design, analysis and even requirements. Support for aspects is thus now raising interest also at the modeling level, including with behavioral modeling languages such as scenarios. With this kind of modeling languages, even if aspect weaving can be performed at the abstract syntax level, a weaving at the semantics level seems a far more appealing and powerful mechanism. In this paper we present a semantic-based aspect weaving algorithm for Hierarchical Message Sequence Charts (HMSCs). The algorithm proposed uses a set of transformations that take into account the compositional semantics of HMSCs to weave an initial HMSC and a behavioral aspect expressed with scenarios.

Keywords

Aspects Weaving, Scenarios, Aspect-Oriented Modeling

1. INTRODUCTION

The idea of encapsulating crosscutting concerns into the notion of aspects looks very promising for complementing the usual notion of modules available in most languages. By localizing these crosscutting concerns, the software engineer can get a greater control over variations, either in the spatial dimension (product line context) or in the temporal dimension (software evolutions). The need to isolate these crosscutting concerns has been popularized by the AspectJ programming language, but there is a growing interest in also handling them earlier in the software life-cycle, for instance at design time [2], or during requirements analysis [1], [17], [16], [10]. Beyond being able to represent aspects at requirement or design time, an automatic aspect weaver at these stages can be very useful for validation purposes (simulation or test case generation) and also for targeting non-aspect-oriented platforms (e.g. vanilla Java).

Many works address aspect weaving at the model level, including with behavioral modeling languages such as scenario languages. Within this paper, we choose High-level Message Sequence Charts (HMSCs) as the scenario model. With behavioral models, aspect weaving can be performed either at the abstract syntax level (where weaving simply consists of searching for a specific pattern that shall be replaced in the syntactic definition of some behaviors), or at the semantic level (where weaving consists of replacing a part of the behaviors defined by the semantics of the description with another behavior). This makes a real difference: syntactic weaving directly transforms a description X into a description X' with aspects, while semantic weaving transforms X into another description X' such that the semantics of X' is the semantics of X woven with an aspect.

Consider for instance the simple HMSC H of Figure 1, which contains a loop over a basic scenario with messages 'try again' and 'new attempt'. Suppose we want to weave some extra-behavior into our system every time a message 'try again' directly follows a message 'new attempt'. A simple way to define a behavioral aspect is firstly to specify a pointcut with a scenario representing the behavior to detect, and secondly to specify the expected behavior with another scenario (called the advice in Figure 2). When weaving is performed at a syntactic level, since the pointcut described by our aspect does not appear explicitly in the HMSC of Figure 1, the description would remain unchanged. At the semantic level, some executions defined by our HMSC may contain parts that are matched by our pointcut, and should then be replaced. The main idea is to compute the HMSC of Figure 4, whose semantics is exactly the semantics of H where all join points have been completed by the additional behaviors specified within the advice. Note that this kind of weaving cannot be defined as a simple syntactic replacement since it is applied on the semantics of the model, i.e. on behaviors generated by a HMSC. As we will discuss later in the paper, the weaving of behavioral aspects may even be intractable in some cases.

The rest of the paper is organized as follows. Section 2 presents the scenario language used for behavioral weaving and the general principles of our semantics-based aspect weaving algorithm. Section 3 introduces some definitions related to the detection of a pointcut within a scenario. Section 4 presents several transformations allowing the application of the weaving operation. Section 5 presents limitations of our approach. Section 6 compares our

*This work has been partially supported by the AOSD-Europe Network of Excellence.

AOSD 06, March 20–24, 2006, Bonn, Germany

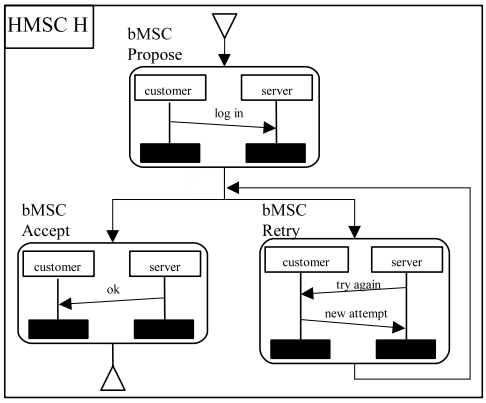

Figure 1: An example of HMSC

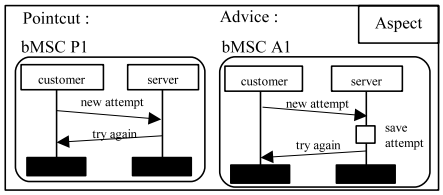

Figure 2: An example of aspect

approach with related works, and section 7 concludes this work and discusses possible extensions.

2. SCENARIOS AND WEAVING

Scenario languages are mainly used to describe behaviors of distributed systems at an abstract level or to capture requirements in early development stages with a clear, graphical, and intuitive representation. In this paper, we will use Message Sequence Charts, a scenario formalism standardized by the ITU [9]. Note however that MSC and UML 2.0's sequence diagrams are very similar languages. Indeed, UML 2.0 sequence diagrams [14] are largely inspired by MSCs (a comparison between UML2.0 and MSC-2000 is given in [6]). Therefore the approach defined in this paper could also be applied to sequence diagrams. Message Sequence Charts (MSC) propose two specification levels. At the lowest level, basic MSCs (bMSCs) describe simple communication patterns between entities of the system called instances. All events located on the same instance are totally ordered, and messages are supposed asynchronous. In Figure 2, bMSC $A1$ represents a behavior where the messages 'new attempt' and 'try again' are exchanged between the instances *customer* and *server*, and where a local event 'save attempt' is performed by the instance *server*. More formally, bMSCs can be defined as partial orders on a set of events, labeled by action names and by the names of each process executing them:

DEFINITION 1 (BASIC MSC). *A* bMSC *is a tuple* $M = (I, E, \leq, A, \alpha, \phi)$, *where* I *is a set of instances,* E *is a set of events (local event, message send, message receive),* $\leq$ *is a partial ordering on* E *imposed by instances and messages,* A *is a set of actions,* $\alpha : E \rightarrow A$ *maps events to actions,* $\phi : E \rightarrow I$ *maps events to instances. (Slightly abusing the notation, we will note* $\phi(E) = \{\phi(e) | e \in E\}$ *the set of instances appearing in any set of events* E*).*

We will also define an empty bMSC denoted by M_ϵ that does not contain events. BMSCs alone do not have sufficient expressive power: they can only define finite behaviors, without real alternatives. For this reason, MSCs have been extended with High-level MSCs, a higher level of specification that allows the composition of bMSCs with operators such as sequence, alternative and loop. Figure 1 shows a HMSC H with an alternative between the bMSCs *Accept* and *Retry*, where *Propose* and *Accept* are composed sequentially, *Retry* is in a loop, etc... HMSCs can be considered as automata labeled by bMSCs. They can be formally defined as follows:

DEFINITION 2 (HIGH-LEVEL MSC). *A* HMSC *is a graph* $H = (\mathcal{N}, T, q_0, \mathcal{Q}_{end}, \mathcal{M})$ *where:* $\mathcal{N}$ *is a set of nodes,* $T \subseteq \mathcal{N} \times \mathcal{M} \times \mathcal{N}$ *is a set of transitions,* q_0 *is the initial node,* $\mathcal{Q}_{end}$ *is a set of end nodes, and* $\mathcal{M}$ *is a finite set of bMSCs. For a transition* $t = (p, l, q) \in T$, *node* p *will be called the origin of a transition* t *and denoted by* $\lambda_-(t)$, *node* q *will be called the goal of* t *and denoted by* $\lambda_+(t)$, *and* l *will be called the label of* t *and denoted by* $\lambda(t)$. *Furthermore, we will require that no transition of* H *leaves from an end node.*

To simplify the notation, we will sometimes depict HMSCs as automata labeled by bMSC names when the exact contents of bMSCs labeling H is not useful. For example, the HMSC of Figure 1 can also be represented by the automata of Figure 5-a.

The notion of sequential composition (noted $\bullet$) is central to understanding the semantics of HMSCs. Figure 3 gives an example of this operator. Roughly speaking, sequential composition of two bMSCs consists of gluing both diagrams along their common instance axes. Note that the sequence operator only imposes precedence on events located on the same instance, but that events located on different instances in two bMSCs $M1$ and $M2$ can be concurrent in $M1 \bullet M2$. Sequential composition can be formally defined as follows:

DEFINITION 3 (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:* $\leq_{1\bullet 2} = \left(\leq_1 \uplus \leq_2 \uplus \{(e_1, e_2) \in E_1 \times E_2 \mid \phi_1(e_1) = \phi_2(e_2)\}\right)^*$

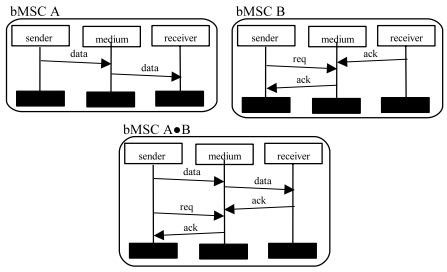

Figure 3: Sequential Composition of A and B

To calculate the new partial ordering $\leq_{1\bullet 2}$, sequential composition consists in ordering events e_1 in bMSC M_1 and e_2 in bMSC M_2 if they are situated on the same instance, and then compute the transitive closure of this ordering. In

this definition, $\uplus$ is the disjoint union of two multisets, i.e. an usual union operation where common elements of both sets are duplicated.

We will also denote by M^k the sequential concatenation of k copies of M, and by M^* the set $\bigcup_{i\in\mathbb{N}} M^i$ with the convention that $M^0 = M_\epsilon$. When two bMSCs $M1, M2$ are such that $M1 \bullet M2 = M2 \bullet M1$, we will say that $M1$ and $M2$ are independent, and write $M1||M2$ (this situation occurs only when $\phi(E_1) \cap \phi(E_2) = \emptyset$). A bMSC B will be called an *atom* iff there does not exist two bMSCs $N \neq M_\epsilon$ and $M \neq M_\epsilon$ such that $B = M \bullet N$. For a set of bMSCs $\mathcal{M}$, we will denote by $At(\mathcal{M})$ the set of all atoms of bMSCs in $\mathcal{M}$. Note that an algorithm to find a decomposition of a bMSC into atoms can be found in [7]. For a bMSC M such that $M = X_1 \bullet X_2 \bullet \cdots \bullet X_k$, where all X_i's are bMSCs, $X_1 \bullet X_2 \bullet \cdots \bullet X_k$ will be called a *factorization of M*.

An *initial path* of a HMSC H is a sequence of transitions $p = t_1.t_2...t_k$ such that the origin of t_1 is the initial node q_0. If moreover the goal of t_k is a node of $\mathcal{Q}_{end}$, then p is called an *accepting path*. The bMSC $\lambda(p)$ associated to a path p is the sequential composition of bMSCs labeling the transitions, $\lambda(p) = \lambda(t_1) \bullet \lambda(t_2) \bullet ... \bullet \lambda(t_k)$. The semantics of a HMSC H is the set of behaviors defined by concatenation of labels along all accepting paths, i.e. the set of bMSCs $\mathcal{L}(H) = \{\lambda(p) | p$ is an accepting path of $H\}$.

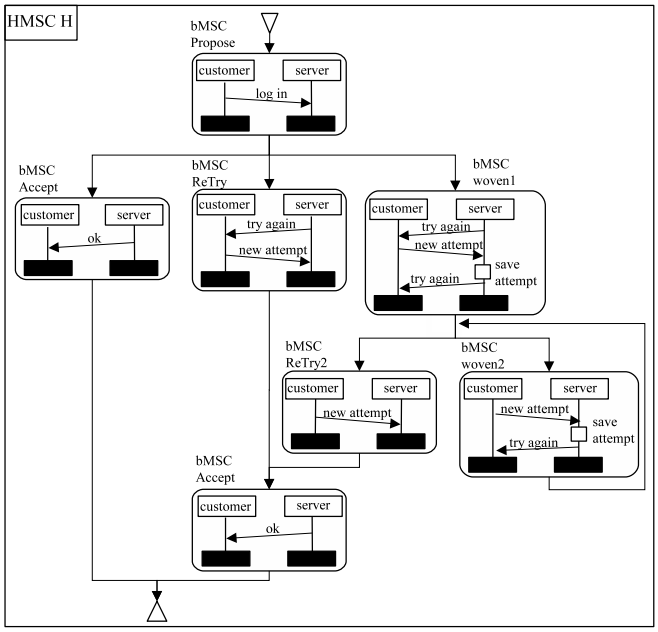

Figure 4: Result of the weaving

A *behavioral aspect* is a pair $A = (P, Ad)$ of bMSCs. P is a pointcut, i.e a bMSC interpreted as a predicate over the semantics of HMSCs satisfied by all join points. Ad is an advice, i.e. the new behavior that should replace the base behavior when it is matched by P. Similarly to Aspect-J, where an aspect can be inserted 'around', 'before' or 'after' a join point, with our approach, an advice may indifferently complete the matched behavior, replace it with a new behavior, or remove it entirely. For a given bMSC M and a given aspect A, we will denote by $M \otimes A$ the bMSC obtained by replacing sequentially every occurrence of P by Ad in M (we will detail in section 3 how this replacement mechanism is implemented).

Consider a HMSC H and a behavioral aspect $A = (P, Ad)$. The goal of the semantic-based weaver is to produce a new HMSC H' where the aspect A is woven every time that the pointcut is detected within a bMSC M of $\mathcal{L}(H)$. More formally, after weaving we should have $\mathcal{L}(H') = \{M \otimes A | M \in \mathcal{L}(H)\}$. Consider for example the HMSC H of Figure 1 and the behavioral aspect of Figure 2. The weaving should produce a HMSC such as H' in Figure 4.

3. MATCHING

Let us now describe how a pointcut can be detected within a finite bMSC.

DEFINITION 4 (MATCHING). *We will say that a pointcut P* matches *a bMSC M, which will be denoted by $P \triangleright M$, if and only if there exists two bMSCs X_1 and X_2 such that $M = X_1 \bullet P \bullet X_2$. Furthermore, if X_1 is the smallest possible bMSC allowing a decomposition of M, we will say that the matching is* minimal.

A pointcut P matches a bMSC M k times (denoted by $P \triangleright^k M$) if there are $k+1$ bMSCs $\{X_i\}_{i\in\{1..k+1\}}$ such that

- $M = X_1 \bullet P \bullet X_2 \bullet ... \bullet X_k \bullet P \bullet X_{k+1}$
- *$\forall i, X_i \bullet P \bullet X_{i+1}$ is a minimal matching. This property guarantees the uniqueness of matching: successive join points are detected in sequence at the earliest position where they appear in a bMSC.*

Using this definition of multiple matching, we can now formally define bMSC weaving. For a behavioral aspect $A = (P, Ad)$ and a bMSC M, if $P \triangleright^k M$, i.e. if M is of the form $M = X_1 \bullet P \bullet X_2 \bullet \cdots \bullet P \bullet X_{k+1}$, then the *weaving* of M and A is defined as $M \otimes A = X_1 \bullet Ad \bullet X_2 \bullet \cdots \bullet Ad \bullet X_{k+1}$. In the examples of Figures 1 and 2 the pointcut $P1$ does not match the bMSC *Retry* but matches the bMSC $Retry \bullet Retry$ once, the bMSC $Retry \bullet Retry \bullet Retry$ twice, etc... A different definition of matching based on the existence of an injective mapping from P to M was proposed in [13, 12]. We think that the pointcut detection approach proposed hereafter can be adapted to work with this definition. However, to keep the weaving process as clear as possible, we will only deal with the simple matching mechanism of definition 4. Since a bMSC is finite, an algorithm to detect a pointcut P within a bMSC M is rather straightforward. In this algorithm, we will denote by $\pi_i(M) \subseteq E_M$ the projection of a bMSC M on a single instance, and by $\pi_E(M)$ the restriction of a bMSC M to a subset of events $E \subseteq E_M$. Furthermore, we will consider that messages from an instance to another never overtake one another. Note that matching detection remains feasible without these assumptions, but that it slightly complicates the algorithm.

For a pointcut P and a bMSC M, the first part of the algorithm (lines 1 to 4) finds for each instance p of P, sequences of events of M located on p such that the labeling of these sequences is exactly the labeling of the restriction of P to instance p. In other words, P is detected locally on instance p by each sequence extracted from M. The second part of the algorithm (lines 5 to 10) checks if all the local sequences can be combined to see whether P is contained or not in M. In such a case, the minimal solution (the solution where locally matched sequences are located as early as possible on their respective instance) is returned.

Algorithm 1 Matching (P,M)

input: pointcut $P = (I_P, E_P, \leq_P, , A_P, \alpha_P, \phi_P)$,
bMSC $M = (I_M, E_M, \leq_M, , A_M, \alpha_M, \phi_M)$

output: E, subset of events matched by P
($E = \emptyset$ if $P \not\triangleright M$)

1: **for all** $i \in I_P$ **do**
2: $w_i = \pi_i(M)$ `/* a word consisting of all events on instance i */`
3: $V_i = \{v \in E^* \mid \exists u, w, w_i = u.v.w \wedge \alpha(v) = \alpha(\pi_i(P))\}$ `/*` $\forall v \in V_i$`, the events of` v `have the same action name as the events of` P `located on the instance` i `*/`
4: **end for**
5: **if** $\exists v_1, v_2, \ldots, v_{|I_P|} \in V_1 \times V_2 \times \cdots \times V_{|I_P|}$ such that $\pi_{v_1 \cup \cdots \cup v_{|I_P|}}(M) = P$ **then**
6: $V = min\{v_1, \ldots, v_{|I_P|} \in V_1 \times \cdots \times V_{|I_P|} \mid \pi_{v_1 \cup \cdots \cup v_{|I_P|}}(M) = P\}$
7: return$(\bigcup_{v_i \in V} v_i)$
8: **else**
9: return$(\emptyset)$
10: **end if**

To extend the notion of weaving to HMSCs, we have to deal with the following problem: for a path $p = t_1 \ldots t_n$ of a HMSC H, when $P \triangleright^q \lambda(p)$, all occurrences of P are not necessarily detected on a single transition. Indeed, a part of P can be detected in the label of a transition $\lambda(t_i)$, and completed later in the label of another transition $\lambda(t_{i+k})$.

Let us again consider the HMSC H in Figure 1, and its automaton representation in Figure 5-a. Consider also pointcut $P1$ in Figure 2. The weaver has to detect a message "new attempt" followed by a message "try again" within all executions of H. To explore the paths of H, we start from the initial node of H and try to see if $P1$ matches the transition labeled by bMSC *Propose*. Since $P1$ does not match *Propose* and since $P1$ does not start to match *Propose*, we can continue our exploration from node $q1$, and look at the next transitions, i.e. the transitions labeled by bMSC *Accept* and bMSC *Retry*. $P1$ does not match *Accept* and since $q2$ is an end node, we know that $P1$ does not match after node $q2$. $P1$ does not entirely match *Retry* either, but starts to match because the message "new attempt" is detected at the end of bMSC *Retry*. So, the pointcut could match a longer bMSC. As there are two transitions $t1 = (q_1, Accept, q_2)$ and $t2 = (q_1, Retry, q_1)$ in H, according to the semantics of HMSCs, we can consider a matching of $P1$ in bMSCs $Retry \bullet Accept$ and $Retry \bullet Retry$. $P1 \not\triangleright Retry \bullet Accept$, but $P1 \triangleright Retry \bullet Retry$, and a new potential matching of $P1$ starts within this second bMSC.

The matching definition indicates how many times a pointcut P was entirely detected in a bMSC M. In addition to this definition, we also need information about the matches that were started in M, and that could eventually be completed in an extension $M \bullet M'$. For this, we introduce a new notion called *potential match*. Roughly speaking, for a bMSC M and a pointcut P, the potential match $PM_{P,M}$ is the biggest concatenation of parts of M that are completely matched by a part of P and could eventually be completed to obtain a complete match of P in M.

Figure 6 shows an example where pointcut $P2$ matches

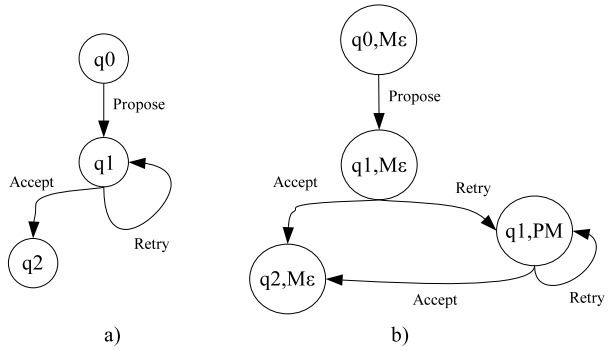

Figure 5: a) Automata-like representation b) HMSC with context

bMSC M once. M can be written $X_1 \bullet P2 \bullet X_2$, but $P2$ starts to match the "end" of M. Indeed, if we add bMSC N to M, $P2$ matches $M \bullet N$ three times, and each of these new matchings involve events of M. The part of M which participates to the two additional matches is a potential match, and is depicted by the area denoted $PM_{P2,M}$ in Figure 6.

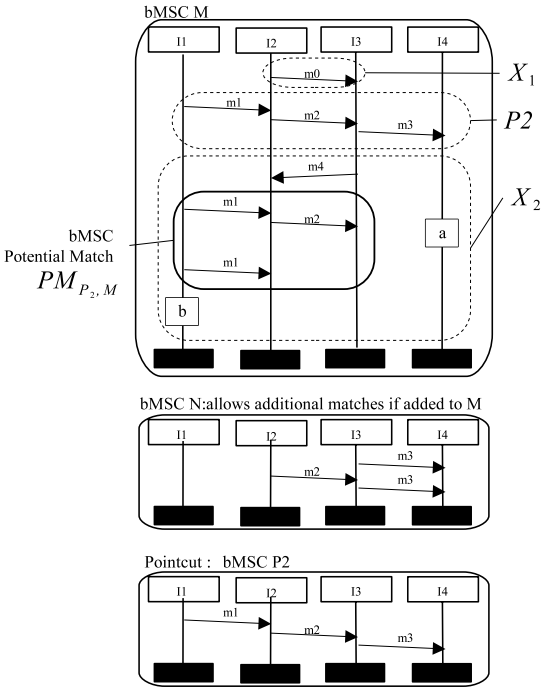

Figure 6: An example of Potential Match

A potential match indicates that currently started matchings have to be studied on longer executions, and that the paths studied can be extended to obtain a longer bMSC where P matches.

The weaving of an aspect $A = (P, Ad)$ into a base HMSC H can be decomposed into two main phases. The first phase consists of detecting join points, and the second phase rewrites them. The main difficulty of weaving resides in the detection: we have to identify all parts of the base HMSC semantics where join points may appear without unrolling

the potentially infinite set of behaviors it generates. In the approach proposed hereafter, detection mainly consists of transforming the original HMSC H into an equivalent HMSC H_{tmp} (i.e. $\mathcal{L}(H) = \mathcal{L}(H_{tmp})$) in such a way that the pointcut P only matches a finite number of paths in H_{tmp}. For the sake of clarity, the transformation process is decomposed into smaller problems. Note that with this decomposition, some parts of the algorithms may be redundant. We can however combine them to obtain a much more efficient algorithm merging all the transformations and saving exploration time. After detection, the rewriting algorithm is trivial, and consists of replacing the occurrences of P in all transitions of the transformed HMSC.

4. BUILDING MATCHED PATHS

This section presents the detection part of our weaving process. From a pointcut P and a base HMSC H, the goal is to transform H into an equivalent HMSC H_{tmp} such that the weaving consists of rewriting only a finite number of paths of H_{tmp}. This transformation of H into H_{tmp} is performed with several simple transformations.

The first step (implemented by algorithm 2) is to unfold a base HMSC H to exhibit all potential matches of a pointcut and compute an equivalent HMSC H_+, such that each node of H_+ is a pair (q, C) where q is a node of H, and C is the potential match of pointcut P for an initial path that ends at node q (this potential match will hereafter be called the **context** of q). In this way, for each node with a context, the context indicates if a currently started match has to be studied on longer paths. However, as we will see in subsection 4.2, when P matches an extension of a behavior that has lead to a context C, C is not always entirely used by the matching. For this reason, we have to compute the parts of the potential match which are actually matched by P for some paths of H_+. These parts will be called the *future matches* contained in C. This second step is performed by algorithm 3 which transforms an HMSC H_+ into an equivalent HMSC $H_\times$, such that each node of $H_\times$ is a triple (q, C, F), where (q, C) is a node of H_+ and F is a future match contained in C. So, in each node the future match indicates if a currently started match of the context will actually end on longer paths. Figure 9 shows an example of transformation from an HMSC H to another HMSC $H_\times$. Note however that this transformation has some limitations, that are described in section 5.

Algorithm 4 allows the resolution of problems related to the loops in a HMSC (subsection 4.3 and 4.4): as soon as a HMSC $H_\times$ contains at least one loop c such that all nodes of c have a non-empty future, the weaving cannot be performed without suitable permutations of bMSCs. Finally, when algorithm 4 terminates successfully, algorithm 5 (subsection 4.5) builds the set $\mathcal{P}_m$ of minimal acyclic paths where the pointcut matches. The weaving process finally consists of replacing these paths by a woven transition to obtain the woven HMSC $H \otimes A$ (algorithm 6).

4.1 Step1: Potential matches

The first transformation, implemented by algorithm 2, is roughly speaking an unfolding of the base HMSC H. This algorithm transforms a HMSC H into an equivalent HMSC H_+ such that each node of H_+ is a pair (q, PM) where q is a node of H, and PM is the potential match of pointcut P (called **context**) for an initial path that ends at node q. For instance, for a pointcut P and an initial path $p = (q0, M1, q1).(q1, M2, q2).(q2, M3, q3)$, we will build the nodes $(q0, M_\epsilon), (q1, PM_{P,M1}), (q2, PM_{P,M1 \bullet M2})$, and $(q3, PM_{P,M1 \bullet M2 \bullet M3})$. These nodes can only be found one after another. The first pair $(node, context)$ created is obviously $(q0, M_\epsilon)$. Then we can compute $PM_{P,M1}$ and build the node $(q1, PM_{P,M1})$. To compute the potential part $PM_{P,M1 \bullet M2}$ of node $q2$, it is not necessary to entirely consider $M1$: considering the context $PM_{P,M1}$ of node $q2$ created previously is sufficient, since $PM_{P,M1 \bullet M2} = PM_{P,PM_{P,M1} \bullet M2}$. Finally, to compute $PM_{P,M1 \bullet M2 \bullet M3}$, we also use a previously discovered context attached to node q_2, that is to say, $PM_{P,M1 \bullet M2}$, and we simply compute $PM_{P,PM_{P,M1 \bullet M2} \bullet M3}$. This way, all reachable contexts can be built inductively from the initial node. To build the possible contexts of a node q, it is not necessary to study all initial paths that end at node q, but only the contexts already computed that are attached to a predecessor of q. This is the approach used by algorithm 2. Since this algorithm is just an unfolding of the initial graph, it is obvious that the set of executions generated by H and $H+$ are equivalent. Note however that this algorithm terminates if and only if the number of potential matches for each node of H is finite ($\mathcal{N}$ is finite by definition), which this is not always the case. This problem will be developed in section 5.

For the pointcut $P1$ in Figure 2, algorithm 2 transforms the HMSC H in Figure 5-a into the HMSC H_+ in Figure 5-b where M_ϵ represents the empty bMSC and PM a bMSC with a single message "new attempt" from the instance *customer* to the instance *server*. The context associated with $q1$ for path $p = (q_0, Propose, q1)$ is M_ϵ, since $P1$ matches *Propose* neither fully nor partially. For any path ending with transition $(q1, Retry, q1)$, the context associated to $q1$ is PM since $P1$ starts to match the message "new attempt".

Algorithm 2 Context(P,H)

input: pointcut P, HMSC $H = (\mathcal{N}, T, q_0, \mathcal{Q}_{end}, \mathcal{M})$
output: HMSC $H_+ = (\mathcal{N}_+, T_+, q_{+0}, \mathcal{Q}_{end_+}, \mathcal{M}_+)$

1: $\mathcal{M}_+ = \mathcal{M}$, $q_{+0} = (q_0, M_\epsilon)$
2: $CurrentStates = \{q_{+0}\}, futureStates = \emptyset$,
3: $\mathcal{N}_+ = \{q_{+0}\}, T_+ = \emptyset$
4: **while** $CurrentStates \neq \emptyset$ **do**
5: **for all** $q_+ = (q, M_p) \in CurrentStates$ and $t = (q, M, q') \in T$ **do**
6: $q'_+ = (q', PM_{P,M_p \bullet M})$
7: **if** $q'_+ \notin \mathcal{N}_+$ **then**
8: $futureStates = futureStates \cup q'_+$
9: $\mathcal{N}_+ = \mathcal{N}_+ \cup q'_+$,
10: **end if**
11: $t_+ = (q_+, M, q'_+)$
12: **if** $t_+ \notin T_+$ **then**
13: $T_+ = T_+ \cup t_+$
14: **end if**
15: **end for**
16: $CurrentStates = futureStates, futureStates = \emptyset$
17: **end while**
18: $\mathcal{Q}_{end_+} = \{(n, M) \in \mathcal{N}+ \mid n \in \mathcal{Q}_{end}\}$
19: return $H_+ = (\mathcal{N}_+, T_+, q_{+0}, \mathcal{Q}_{end_+}, \mathcal{M}_+)$

4.2 Step2: Future matches

For a pointcut P and two bMSCs M and M', the context $PM_{P,M}$ is not always entirely used when P matches $M \bullet M'$. Consider for example the bMSC M and the pointcut $P2$ in Figure 6, and the bMSC M' in Figure 7. The bMSC $M \bullet M'$ is depicted in Figure 7 and we can see that the part of the context $C = PM_{P,M}$ which is actually used when P matches $M \bullet M'$, only consists of a message '$m1$'. This part of a context C, will be called the *future match* of context C when M' follows, and it will be denoted $FM_{P,C,M'}$.

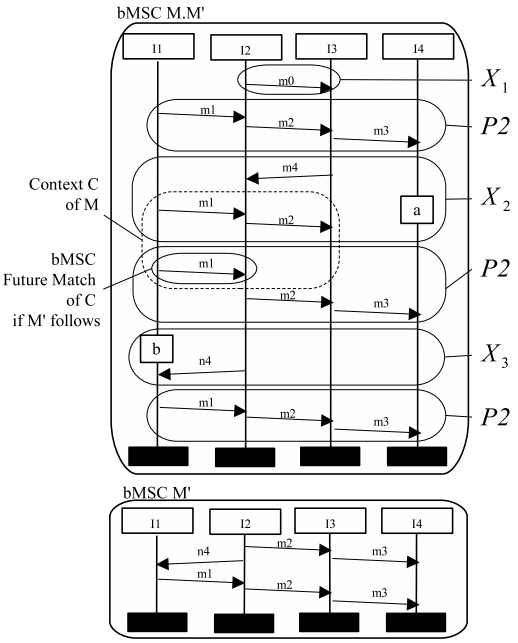

Figure 7: An example of Future Match

Similarly, for a peculiar node $q_+ = (q, C)$ of a HMSC H_+, the part of C that will be used to find a join point depends on the path that may start from q. If $q_+ \in \mathcal{Q}_{end_+}$, it is obvious that the future match associated to the node (q, C) is M_ϵ, since no extension of a path ending in q_+ is possible. If $q_+ \notin \mathcal{Q}_{end_+}$, then the part of C that is matched by a pointcut depends on the transitions that follow. Even if the number of paths starting from a node q_+ and ending on an end node can be infinite, the set of possible future matches can be defined inductively as follows:

DEFINITION 5. *Let H_+ be a HMSC (with context), P be a pointcut, and $q_+ = (q, C)$ be a node of H_+. A* future match *associated to node q_+ is a subset $F(q_+)$ of C defined as follows:*

1. $F(q_+) = M_\epsilon$ *if* $q_+ \in \mathcal{Q}_{end_+}$,
2. $F(q_+) = FM_{P,C,M} \cup (C \cap F')$ *if* $q_+ \notin \mathcal{Q}_{end_+}$, *and if there exists a node* $q'_+ = (q', C')$ *and a transition* $t = (q_+, M, q'_+) \in T_+$ *such that F' is a future match of* q'_+.

Let us discuss the second item of this definition. The element that appears in $FM_{P,C,M}$ is the part of the context C which is actually matched if the bMSC M follows. However, some other events of C can be matched with a longer bMSC, but these events necessarily appear in the future match F' associated to q'_+. So these other events are contained in $C \cap F'$.

According to this definition, algorithm 3 that computes all possible futures matches for a contextual HMSC immediately follows. This algorithm transforms a HMSC H_+ into an equivalent HMSC $H_\times$ such that each node of $H_\times$ is a triple (q, C, FM) where (q, C) is a node of H_+, and FM is a future match for context C. Like algorithm 2, the transformation implemented by algorithm 3 is roughly speaking an unfolding of H_+, and hence produces an equivalent HMSC. The main difference is that this unfolding starts from end nodes.

Consider the HMSC H and the pointcut P in Figure 8. Figure 9 presents H as an automaton labeled by bMSC names, and the HMSCs H_+ and $H_\times$ obtained by applying algorithms 2 and 3. In H_+, $C1$ is the context of node q_1. $C1 = X$, and after each transition (q_1, Y, q_1), $C1$ remains unchanged. In $H_\times$, node $(q1, C1)$ is duplicated with two different future matches, as the part of bMSC X that will be used to match P is not the same when Z immediately follows X and when at least a copy of Y is inserted between X and Z. Figure 10 shows another example of a HMSC $H_\times$ obtained from the HMSC H_+ in Figure 5.

Note that if algorithm 2 terminates, then algorithm 3 also terminates. Indeed, if the number of potential matches is bounded for a HMSC H and a pointcut P (condition of convergence for algorithm 2), then the number of future matches is also bounded because a future match is always a subset of a potential match.

Algorithm 3 Future(P,H_+)

input: pointcut P,
HMSC $H_+ = (\mathcal{N}_+, T_+, q_{+_0}, \mathcal{Q}_{end_+}, \mathcal{M}_+)$
output: HMSC $H_\times = (\mathcal{N}_\times, T_\times, q_{\times_0}, \mathcal{Q}_{end_\times}, \mathcal{M}_\times)$

1: $\mathcal{M}_\times = \mathcal{M}_+$, $\mathcal{Q}_{end_\times} = \{(n, M, M_\epsilon) | (n, M) \in\in \mathcal{Q}_{end_+}\}$
2: $CurrentStates = \mathcal{Q}_{end_\times}, futureStates = \emptyset$,
3: $\mathcal{N}_\times = CurrentStates, T_\times = \emptyset$
4: **while** $CurrentStates \neq \emptyset$ **do**
5: **for all** $q_\times = (q, C, F) \in CurrentStates$ and $t_+ = ((q', C'), M, (q, C)) \in T_+$ **do**
6: $F' = FM_{P,C',M} \cup (C' \cap F)$
7: $q'_\times = (q', C', F')$
8: **if** $q'_\times \notin \mathcal{N}_\times$ **then**
9: $futureStates = futureStates \cup q'_\times$
10: $\mathcal{N}_\times = \mathcal{N}_\times \cup q'_\times$,
11: **end if**
12: $t_\times = (q'_\times, M, q_\times)$
13: **if** $t_\times \notin T_\times$ **then**
14: $T_\times = T_\times \cup t_\times$
15: **end if**
16: **end for**
17: $CurrentStates = futureStates, futureStates = \emptyset$
18: **end while**
19: $q_{\times_0} = (q_0, M_\epsilon, M_\epsilon)$,
20: return $H_\times$

4.3 Unbreakable loops

From a pointcut P and a HMSC H, we have built a HMSC $H_\times$ whose nodes contain a context and a future. As we want to isolate the join points into a finite number of paths of

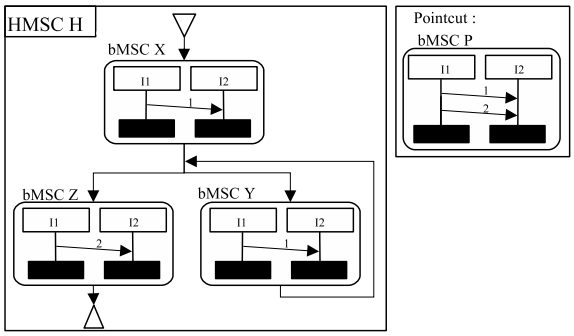

Figure 8: A HMSC

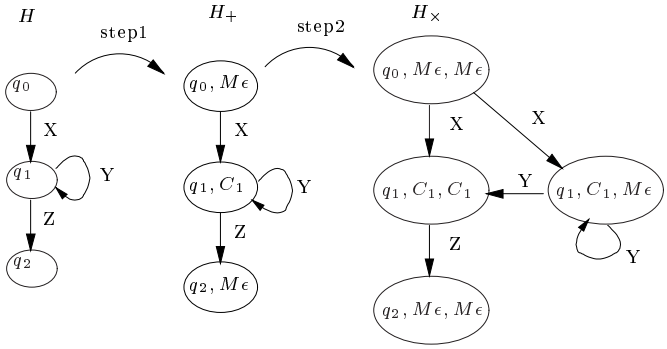

Figure 9: Transformation of H into $H_\times$

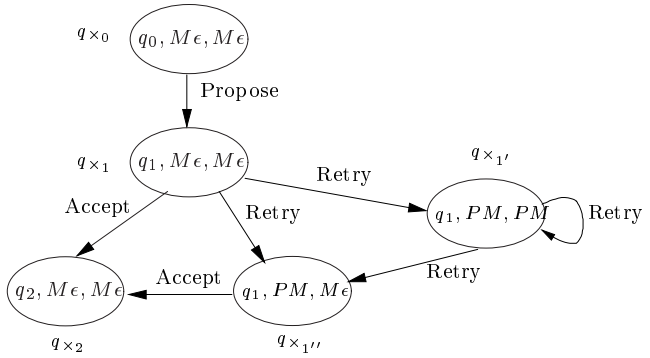

Figure 10: HMSC $H_\times$ related to the HMSC H_+ in Figure 5

a transformed HMSC, the next thing to do is to rewrite $H_\times$ into an equivalent HMSC where the number of paths starting from nodes with an empty future M_ϵ and ending on nodes with also an empty future is finite. Algorithm 5 builds the set $\mathcal{P}_m$ of these paths and defines them more formally. This set $\mathcal{P}_m$ ensures that P matches entirely bMSC $\lambda(p)$ associated to any path p of $\mathcal{P}_m$ a certain number of times, and that a potential matching is not initiated from this path.

However, algorithm 5 cannot be directly applied on $H_\times$. If $H_\times$ contains at least one loop c such that all nodes of c have a non-empty future, then the algorithm diverges. Consider, for example, the HMSC in Figure 10: there is an infinite number of paths starting and terminating with an empty future: $Retry \bullet Retry \bullet Accept$, $Retry \bullet Retry \bullet Retry \bullet Accept$, etc...

So, before using Algorithm 5, we have to break the cycles of $H_\times$ that loop infinitely on nodes with non-empty future. Let us denote by $\mathcal{C}_{H_\times}$ the set of elementary cycles of $H_\times$ and $\mathcal{UC}_{H_\times}$ the set of "unbreakable" elementary cycles, i.e. cycles for which algorithm 5 generates an infinite set of paths. Formally, we have $\mathcal{UC}_{H_\times} = \{c = t_1.t_2...t_k \in \mathcal{C}_{H_+} | \forall i \in \{1...k\}, FM(\lambda_-(t_i)) \neq M_\epsilon\}$. Sometimes, breaking a cycle just consists of splitting a bMSC labeling a transition into smaller parts, so that an empty future appears. For instance, in Figure 10, the transition $t = (q_{\times_{1'}}, Retry, q_{\times_{1'}})$ can be split into two transitions $t' = (q_{\times_{1'}}, Retry1, q_{new})$ and $t'' = (q_{new}, Retry2, q_{\times_{1'}})$ such that $Retry1$ is a bMSC with a single message 'try again' and $Retry2$ is a bMSC containing a single message 'new attempt'. After this transformation, one can easily note that the future of node q_{new} is the empty bMSC M_ϵ.

However, things are not always so easy. Consider for example HMSC H_3 of Figure 11: pointcut $P3$ matches an infinite number of bMSCs $(X1X2, X1Y1X2, X1Y1Y1X2, ...)$, but for each of these matched bMSCs, the events of $Y1$ associated to the loop are never matched by $P3$. Algorithm 5 cannot be applied to a HMSC containing this kind of cycle because there is an infinite number of paths starting and ending at nodes with empty future. Since bMSCs $X1$ and $Y1$ are independent, a possible solution is to rewrite the HMSC of Figure 11 into an equivalent one, where the loop on $Y1$ is performed before bMSC $X1$. We obtain the HMSC $H3'$ in Figure 11. Note that within this new HMSC the matching of pointcut $P3$ no longer involves an infinite number of occurrences of $Y1$. Let us now try to formalize this transformation, and see whether it can be applied in a systematic way to solve loop problems.

Definition 6. *Let $H_\times$ be a HMSC augmented with potential and futures matches, P be a pointcut. Let $\mathcal{UC}_{H_\times}$ be the set of unbreakable cycles of $H_\times$. An* unbreakable generator *of $H_\times$ and P is an acyclic path $p = t_1 \dots t_k$ of H_+ that:*

- *starts from a node with an empty future, i.e., $F(\lambda_-(t_1)) = M_\epsilon$*
- *ends on a node with empty future, i.e. $F(\lambda_+(t_k)) = M_\epsilon$*
- *never passes through a node with empty future other than $\lambda_-(t_1)$ and $\lambda_+(t_k)$, i.e. $\forall i \in 2..k$, $F(\lambda_-(t_i)) \neq M_\epsilon$*

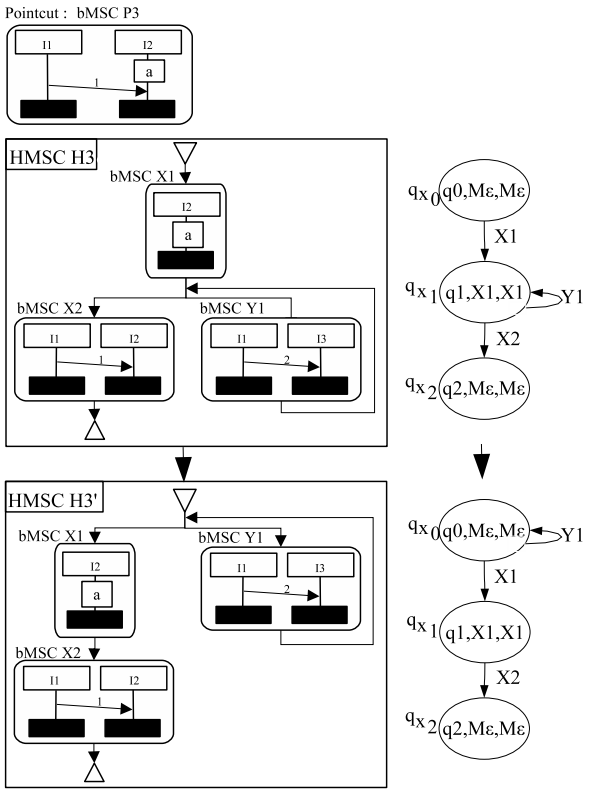

Figure 11: A loop problem solved with a permutation

- *crosses an unbreakable cycle of* H_+ *i.e* $\exists j \in 2..k, c = t'_1 \dots t'_q \in \mathcal{UC}_{H_\times}$ *such that* $\lambda_-(t_j) \in \bigcup_{i\in 1..q} \lambda_-(t'_i)$
- $P \triangleright^k \lambda(p)$

Let $\mathcal{P}_u(H_\times)$ be the set of unbreakable generators of $H_\times$. Any unfolding of an unbreakable loop in an unbreakable generator produces a new path where P matches, and that does not pass through a node with empty future. The main idea behind the algorithm that follows is to check whether there is an equivalent HMSC *up to* permutation of independent bMSCs that does not contain unbreakable generators.

4.4 Breaking generators with permutations

So far, we have identified a problem related to acyclic paths that are connected to a loop where a pointcut matches but that do not pass through a node with an empty future. In this situation, weaving is not just a simple replacement of a finite set of paths by a woven expression. The main objective is then to find if there is a transformation of problematic paths into equivalent sequences of transitions such that loops systematically pass through a node with empty future. We will say that a loop is *splittable* if it passes through a node with empty future, and we will say that a loop is $null - match$ if an initial node of the loop has an empty future. This section provides a solution for cases where unbreakable cycles are *disjoint*, i.e. $\forall c = t_1 \dots t_k, c' = t'_1 \dots t'_p \in \mathcal{UC}_{H_\times}$, $\bigcup_{i\in 1..k} \lambda_-(t_i) \cap \bigcup_{i\in 1..p} \lambda_-(t'_i) = \emptyset$. This transformation is better described using regular expressions.

DEFINITION 7. *A regular expression on an alphabet of atoms* Σ_{At} *is an expression of the form* $E ::= \sigma.E \mid (E)^* \mid E1 + E2 \mid M_\epsilon$, *where* $\sigma \in \Sigma_{At}$ *is an atomic bMSC,* $(E)^*$ *denotes iteration, and* $E1 + E2$ *is an alternative.*

Regular expression and finite automata have the same expressive power. Moreover, from a regular expression we can compute an automaton generating the same language (using Brzozowski's algorithm [8]), and from an automaton, we can compute an equivalent expression (using a reduction algorithm). Hence, we can manipulate indifferently regular expressions on bMSCs or HMSCs, and get back to the other model. For a given regular expression E, we will call H_E the HMSC generating the same set of scenarios, and for a given HMSC H, we will call E_H the equivalent regular expression on bMSCs. For details about this well-known relation between automata and regular expressions, interested readers may consult [8], chapter 2.

Similarly to what is done for HMSCs, we can associate a context to an expression $E = X_1 \dots X_k$, just by computing the potential match of a pointcut P in $X_1 \bullet \dots \bullet X_k$. For any sub-expression $E' = X_1 \dots X_i$ of E, we can also compute a future match $Future(E')$ similarly to what have already been proposed for HMSCs: $Future(E') = FM_{P,E',X_{i+1}\bullet\dots X_k}$.

Let $p = t_1.t_2.\dots.t_q$ be an unbreakable generator, connected to a set of disjoint unbreakable cycles $C = \{c_1, c_2, \dots c_k\}$. We can associate to p an expression

$$E_p = X_{11} \dots X_{1k_1}(Y_{11} \dots Y_{1q1})^*.X_{21} \dots X_{2k_2}.(Y_{21} \dots Y_{2q_2})^* \dots (Y_{q1} \dots Y_{qq_q})^*.X_{q1} \dots X_{qk_q}$$

where all X_{ij} and Y_{ij}'s are atoms, and that generates the same set of behaviors as $\lambda(t_1) \bullet \dots \bullet \lambda(t_{i1}) \bullet \lambda(c_1)^* \bullet \lambda(t_{i1+1}) \bullet \dots \bullet \lambda(c_k)^* \bullet \lambda(t_{ik}) \bullet \dots \bullet \lambda(t_q)$, where for all $i_j, j \in 1..k, \lambda_+(t_{i_j}) = \lambda_-(c_j)$. Note that this expression is never of the form $E1 + E2$.

Consider for instance the HMSC $H3$ and the pointcut $P3$ in Figure 11. The path $p = (q_{\times_0}, X_1, q_{\times_1})(q_{\times_1}, X_2, q_{\times_2})$ is an unbreakable generator since the cycle $c = (q_{\times_1}, Y_1, q_{\times_1})$ is connected to p and it does not contain a node with empty future. Each bMSC of $H3$ is an atom. So, we can associate to p the expression $E_p = X_1.(Y_1)^*.X_2$.

The idea behind the following algorithms is to rewrite these expressions up to permutation of independent atoms so that cycles become splittable or null-match. The new expressions can then be used to compute a new HMSC without problematic paths. Consider again the HMSC $H3$ and the pointcut $P3$ in Figure 11. The expression $E_p = X_1.(Y_1)^*.X_2$ is equivalent to the expression $E_p = (Y_1)^*.X_1.X_2$ for which the path $p = (q_{\times_0}, X_1, q_{\times_1})(q_{\times_1}, X_2, q_{\times_2})$ is no longer unbreakable.

We will use several types of permutations: Let $E = X_1.X_2 \dots X_k$ be a regular expression on a set of bMSCs. A simple permutation is to exchange the respective order of two bMSCs when possible, i.e. E can be rewritten into an equivalent regular expression $E' = X_1.X_2 \dots X_{i+1}.X_i \dots X_k$ whenever X_{i+1} and X_i are independent. Indeed, when two bMSCs M and N are independent, then $M \bullet N = N \bullet M$, and this property extends to any sequence of bMSCs. In the same way, if E contains loops, a loop can be moved upward (or forward) in a string if all the bMSCs it contains are independent from all atoms that are overtaken. Finally, two loops can be permuted if they are independent.

When two regular expressions E and E' are equivalent we will note $E \simeq E'$. For a given expression E on a set of atoms Σ_{At}, we will denote by $[E]$ its equivalence class, i.e. the least set of expression on Σ_{at} containing E such that $\forall e \in [E], \forall e' \simeq e, e' \in [E]$. In addition to the permutations, we can use a classical property of regular expressions such as $a(ba)^*bc = (ab)^*abc = ab(ab)^*c$ to find the equivalence class of an expression E. Finally, When two expressions are equivalent, then their iterations are also equivalent. Let note that for any expression E, since the definition of equivalence never unfolds any starred expression, $[E]$ is finite and can be effectively computed.

DEFINITION 8. *Let E_1, E_2 be two regular expressions over an alphabet Σ of bMSCs, generating words in two languages $L_1 \subseteq \Sigma^*$ and $L_2 \subseteq \Sigma^*$. The* left quotient *of E_1 by E_2 is the regular expression E_1/E_2 that generates words of $L_1/L_2 = \{w \in \Sigma^* | \exists v \in L_2 \wedge w.v \in L_1\}$.*

The right quotient *of E_1 by E_2 is the regular expression $E_1 \backslash E_2$ that generates words of $L_1 \backslash L_2 = \{w \in \Sigma^* | \exists v \in L_2 \wedge v.w \in L_1\}$.*

The complement *of an expression E is an expression $\overline{E}$ that generates words of $\Sigma^* - L_E$.*

These definitions of quotients and complements are useful for the definition of the replacement of a sub-expression by another, which is central in the weaving process.

DEFINITION 9. *Let $H = (\mathcal{N}, T, q_0, \mathcal{Q}_{end}, \mathcal{M})$ be a HMSC, E be a regular expression over $\mathcal{M}^*$, and E' be an equivalent regular expression over $\Sigma = \mathcal{M} \cup At(\mathcal{M})$. The HMSC obtained by* replacement *of E by E' in H will be denoted $H_{E|E'}$, and is the HMSC H_{Enew} associated with the regular expression*

$$Enew = \big(E_H/(E.\mathcal{M}^*)\big).E'.\big(E_H \backslash (\mathcal{M}^*.E)\big) \cup \big(E_H \cap \overline{\mathcal{M}^*.E.\mathcal{M}^*}\big)$$

This definition of replacement can be used to replace an unsplittable expression E by a splittable expression E' everywhere in the semantics of H. We can also use this definition to replace one specific occurrence of an expression E_p corresponding to an unbreakable generator p. To do so, it is sufficient to label differently the bMSC associated with each transition (i.e ensure that $\forall t, t' \in T, \lambda(t) \neq \lambda(t')$ even if the bMSCs are isomorphic). With this convention, the replacement of an expression E_p rewrites only a single path of $H_\times$ with all its connected unsplittable loops.

Note that for all generators p of a HMSC $H_\times$, since all expressions e in $[E_p]$ generate the same scenarios, all HMSCs obtained by replacing an expression $e \in [E_p]$ by another expression $e' \in [E_p]$ are equivalent.

For a behavioral aspect $A = (P, Ad)$, and a HMSC $H_\times$ augmented with potential and future matches computed from a HMSC H and the pointcut P, we compute the set $\mathcal{P}_u(H_\times)$ of unbreakable generators of $H_\times$, and the set $Pexp$ of the **associated** expressions. Then, if for all $E_p \in Pexp$, there is an expression $e \in [E_p]$ in which all starred subexpressions of e are either splittable or null-match, then there exists a HMSC H_{tmp} equivalent to $H_\times$ that does not contain infinite path passing infinitely often through non-empty futures. If not, the weaving cannot be performed. Finally, when H_{tmp} can be computed, each expression E_p related to a unbreakable generator p is replaced by the splittable or null-match expression found. The replacement of unbreakable generators is implemented by Algorithm 4.

Algorithm 4 Permutations($H_\times, \mathcal{UC}_{H_\times}$)

input: a HMSC $H_\times$ with contexts and futures
$\mathcal{UC}_{H_\times}$, set of unbreakable cycles
output: a HMSC H_{tmp} without unbreakable generators

1: Compute $\mathcal{P}_u(H_\times)$, set of unbreakable generators
2: $Pexp = \{E_p | p \in \mathcal{P}_u(H_\times)\}$
3: Compute E_H, regular expression associated to H
4: **for all** $E_p \in Pexp$ **do**
5: Compute $[E_p]$.
6: **if** $\nexists e \in [E_p]$ such that $\forall c$ connected to e, c is splittable or null-match **then**
7: Weaving cannot be performed ; STOP
8: **else**
9: Choose e with splittable or null-match cycles
10: $Rep = Rep \cup \{(E_p, e)\}$
11: **end if**
12: **end for**
13: **for all** $(E_p, e) \in Rep$ **do**
14: `/* replace` E_p `with` e `in` E_H `*/`
15: $E_H = \big(E_H/(E_p.\mathcal{M}^*)\big).e.\big(E_H \backslash (\mathcal{M}^*.E_p)\big) + \big(E_H \cap \overline{\mathcal{M}^*.E_p.\mathcal{M}^*}\big)$
16: **end for**
17: $H_{tmp} = H_{E_H}$ `/* compute the HMSC equivalent to the expression` E_H `*/`
18: return(H_{tmp})

Note that some generators do not have equivalent expressions with splittable or null-match cycles. Consider the HMSC H and the pointcut P in Figure 12. The path $p = (q_{\times 0}, X_1, q_{\times 1})(q_{\times 1}, Y, q_{\times 2})(q_{\times 2}, X_2, q_{\times 3})$ is clearly an unbreakable generator. The bMSCs X_1 and X_2 are atoms and the bMSC Y can be split into two atoms Y_a and Y_b where Y_a is a bMSC with a single local event 'a' on instance $I1$ and Y_b is a bMSC with a single local event 'b' on instance $I3$. An expression associated to path p is $E_p = X_1.Y_a.Y_b.(Y_a.Y_b)^*.X_2$. The equivalence class of E_p contains neither a splittable nor a null-match cycle. The only way to obtain an empty future is to consider the expression $E'_p = (Y_b)^n.X_1.X_2.(Y_a)^n$. However, $E'_p \notin [E_p]$, and even if it is equivalent to E_p, E'_p is not a regular expression anymore. So, the HMSC of Figure 12 cannot be transformed into an equivalent HMSC to avoid unbreakable generators.

4.5 H_{tmp} and woven HMSC

So far, if algorithms 2, 3 and 4 have succeeded, we have obtained a HMSC H_{tmp} such that all loops pass through a node labeled with an empty future. The goal of Algorithm 5 is to build a finite set $\mathcal{P}_m$ of paths of H_{tmp} such that $\forall p \in \mathcal{P}_m$, the pointcut P matches exactly k times $\lambda(p)$. $\mathcal{P}_m$ contains all join points that may appear in the semantics of H_{tmp}. Hence, the weaving only consists of a rewriting of paths in $\mathcal{P}_m$.

More specifically, $\mathcal{P}_m$ is the set of paths of a HMSC H_{tmp} such that for each path p of $\mathcal{P}_m$:

i) p starts from a node with future match equal to M_ϵ,

ii) p terminates on a node with future match equal to M_ϵ,

iii) All other nodes of p are labeled by future matches different from M_ϵ

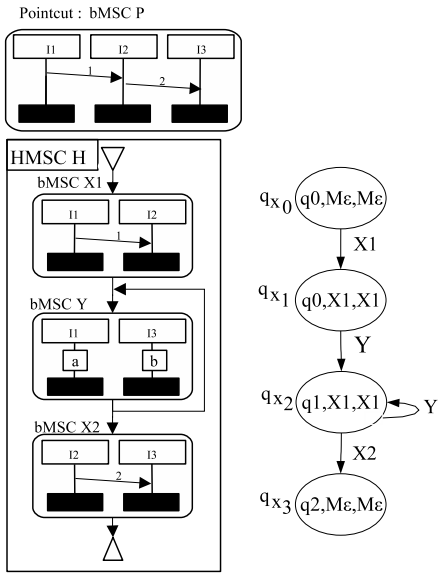

Figure 12: Impossible weaving

iv) $\exists k \in \mathbb{N}$ such that P matches the bMSC $\lambda(p)$ associated to p k times.

We know that after algorithm 4, all loops participating in a match pass through at least one node labeled with an empty future. So, for a HMSC H_{tmp} obtained using Algorithms 2, 3 and 4, each path of $\mathcal{P}_m$ is acyclic, and $\mathcal{P}_m$ is finite. The construction of $\mathcal{P}_m$ is performed by algorithm 5.

Now that we have detailed how to transform a base HMSC H into a new equivalent HMSC H_{tmp} and identified all join points in a finite set $\mathcal{P}_m$ of paths of H_{tmp}, the only remaining work is to replace all parts matched by the pointcut with the advice. As for permutations, this can be performed through a rewriting of a regular expression. Algorithm 6 below describes the overall weaving process.

5. LIMITATIONS

The algorithms described in section 4 have some inherent limitations. Semantic weaving is not always a tractable problem: for some HMSCs and some pointcuts, there might be no way to build a HMSC generating the woven semantics. Hence, the algorithms described in section 4 do not always work. This section identifies sufficient conditions on pointcuts and HMSCs so that our algorithms terminate with a correct result.

The first limitation of our approach is a termination problem for Algorithm 2. From a HMSC H, Algorithm 2 builds a HMSC H_+ where the nodes of H are transformed into nodes containing a context (a potential match). Since the number of nodes of the HMSC H is finite, the algorithm terminates if and only if the number of potential matches is finite.

Applying algorithm 2 on the HMSC of Figure 5-b generates a new HMSC that exhibits only two different contexts: M_ϵ and PM. Consider now HMSC $H4$ and pointcut $P4$ in Figure 13. For this case, the size of contexts can grow infinitely, and it is impossible to build a finite HMSC with a finite set of contexts. From node (q_0, M_ϵ), the potential match obtained in $M_\epsilon \bullet G$ contains a single local event 'a'

Algorithm 5 BuildP(P,H_{tmp})

input: pointcut P,
HMSC $H_{tmp} = (\mathcal{N}_{tmp}, T_{tmp}, q_{0_{tmp}}, \mathcal{Q}_{end_{tmp}}, \mathcal{M}_{tmp})$.

output: $\mathcal{P}_m$ set of matched paths.

1: $\mathcal{P}_m = \{(q_\times, M, q'_\times) \in T_{tmp} \mid F(q_\times) = F(q'_\times) = M_\epsilon \wedge \exists k \in \mathbb{N}, P \triangleright^k M\}$
2: $Path = \{(q_\times, M, q'_\times) \in T_{tmp} \mid F(q_\times) = M_\epsilon \wedge F(q'_\times) \neq M_\epsilon\}$
3: **while** $Path \neq \emptyset$ **do**
4: $\quad FuturePath = \emptyset$
5: $\quad$ **for all** $p = t_1.t_2...t_k \in Path$ **do**
6: $\quad\quad$ **for all** $(q_\times, M, q'_\times) \in T_{tmp}$ such that $q_\times = \lambda_+(t_k)$ **do**
7: $\quad\quad\quad$ **if** $F(q'_\times) = M_\epsilon$ **then**
8: $\quad\quad\quad\quad \mathcal{P}_m = \mathcal{P}_m \cup (p.t)$
9: $\quad\quad\quad$ **else**
10: $\quad\quad\quad\quad FuturePath = FuturePath \cup (p.t)$
11: $\quad\quad\quad$ **end if**
12: $\quad\quad$ **end for**
13: $\quad$ **end for**
14: $\quad Path = FuturePath$
15: **end while**
16: return $\mathcal{P}_m$

Algorithm 6 Weave(H,A)

input: a HMSC $H_=(\mathcal{N}, T, q_0, \mathcal{Q}_{end}, \mathcal{M})$,
a behavioral aspect $A = (P, Ad)$
output: a woven HMSC H'.

1: $H+ = Context(P, H)$
2: $H_\times = Future(P, H_+)$
3: compute $\mathcal{UC}_{H_\times}$
4: **if** $\forall c, c' \in \mathcal{UC}_{H_\times}$, c and c' are disjoint **then**
5: $\quad H_{tmp} = Permutations(H_\times, \mathcal{UC}_{H_\times})$
6: $\quad \mathcal{P}_m = BuildP(P, H_{tmp})$
7: $\quad$ **for all** $p \in \mathcal{P}_m$ **do**
8: $\quad\quad$ Compute E_p
9: $\quad\quad$ Find the minimal factorization $x_1 \bullet P \bullet x_2 \cdots \bullet P \bullet x_k$ of E_p
10: $\quad\quad E'_p := x_1 \bullet Ad \bullet x_2 \cdots \bullet Ad \bullet x_k$
11: $\quad\quad H_{tmp} = H_{tmp\{E_p|E'_p\}}$
12: $\quad$ **end for**
13: $\quad$ Return H_{tmp}
14: **else**
15: $\quad$ Weaving cannot be performed ; STOP
16: **end if**

(bMSC $A1$). From the node $(q_0, A1)$, the potential match relating to $A1 \bullet G$ contains two local events 'a' (bMSC $A2$), and from the node (q_0, An), (An contains n local events 'a') the potential match relating to $An \bullet G$ contains $n+1$ local events 'a' (bMSC $An+1$). From a node and a context, there is always a way to find an extension to transform a potential match into a complete match, but this extension creates a new and bigger potential match than previously.

Note that the problem is not due to the algorithms, and is really intractable: for HMSC $H4$, and aspect $A = (P4, Ad4)$ the semantics of weaving is a set of behaviors containing scenarios of the kind $Ad4^n.A1^n, n \in \mathbb{N}$. This expression is not a regular expression on a set of bMSCs, and hence cannot be represented by a HMSC.

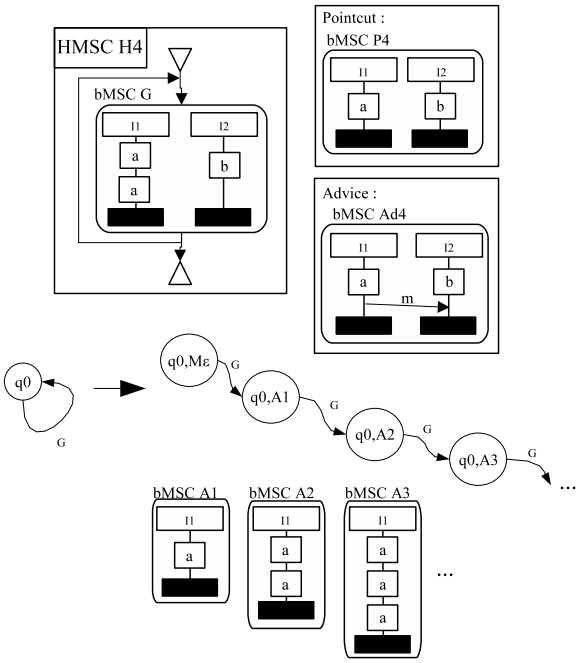

Figure 13: An example where algo. 2 does not terminate.

To avoid the problem of infinite number of contexts, we will only use connected bMSCs as pointcuts, for which the possible number of bMSC potential match is finite. A bMSC M is *connected* if and only if there does not exist two bMSCs $M1$ and $M2$ (different from M_ϵ) such that $M = M1 \bullet M2$ and $M1||M2$. For example, in Figure 2, the bMSCs $P1$ and $A1$ are connected, whereas in Figure 13, the bMSC $P4$ is not connected ($P4 = Pa \bullet Pb$ with Pa a bMSC with only a local event 'a' and Pb a bMSC with only a local event 'b', and $Pa||Pb$).

An interesting property is that for a HMSC H and an aspect $A = (P, Ad)$, if the pointcut P is connected, then the number of potential matches generated by H is bounded. We do not give here a complete formal proof that of this property but only the main idea of the demonstration. Consider Figure 14 where a pointcut P is depicted (the same pointcut as in Figure 6). It is clear that for this pointcut and for any H, there exist only three possible potential matches noted PM_1, PM_2 and PM_3. Let us call *potential part* a beginning of match into a potential match. For instance, the potential match PM_1 contains two potential parts: one, noted PP_1 in Figure 14, with the two first messages 1 and 2; the other, noted PP_2, with the last message 1. The main idea is to show that, in the general case, the size of a potential part PP_{i+1} is always strictly smaller than a potential part PP_i. In your example, the potential part PP_2 contains only one message whereas PP_1 contains two messages. Since the size of potential parts in a potential match is strictly decreasing, when the pointcut is connected, the number of potential matches is bounded.

Another limitation is that the treatment of unbreakable generators is restricted to situations where unbreakable cycles are disjoint. With this situation, loops have the same properties as atoms in an expression, and the permutation are more or less permutations over finite words. The permutation algorithm could be extended as long as the equivalence class for an expression remains a finite set of expressions.

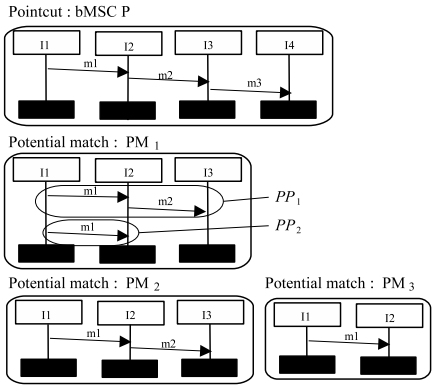

Figure 14: A bounded number of potential matches.

6. RELATED WORK

Clarke [2] uses UML templates to define aspects. The question addressed by her work is more the specification of aspects than the weaving process into non-aspectual models, but our semantic-based weaver can be easily adapted to this work to keep the advantages of the two approaches.

Similarly to our approach, Whittle and Araujo ([17] and [1]) represent behavioral aspects with scenarios. Aspectual scenarios are modeled as interaction pattern specifications (IPSs introduced in [5]) and are composed with specification scenarios. The weaving process defined in [17] composes scenarios with instantiation and ad-hoc composition operators. The weaving process proposed by [1] is performed into two steps. The first step is to generate state machines from the aspects and from the specification. The weaving process is then a composition of these state machines. However, in these two approaches, the detection of join points is not automated: users have to specify them with a binding relation between an aspect and a specification. Moreover, they do not address specification scenarios defining infinite behavior. Another difference is that none of these scenario-based weaving address the weaving at the semantic level as we propose here.

More generally, in [3] and [4], Douence et al. are interested in event pattern matching, which is close to our approach. A significant difference is that they use a monitor to perform event pattern matching at runtime on a program execution,

whereas our weaving is made statically at a modeling level.

The aspect model and in particular the mechanism to identify join points (the pointcut definition language) plays a critical role in the applicability of the aspect-oriented methodology. According to Kiczales [11], the pointcuts definition language probably has the most relevant role in the success of the aspect-oriented technology but most of the solutions proposed so far are too tied to the syntax of the programs manipulated.

Ostermann et al. [15] try to address this problem by proposing a static joint point model that exploits information from different models of program semantics. They show that this model of joint points increases the abstraction level and the modularity of pointcuts.

Note however that the semantics matching is not only a problem studied in the aspect community, but has also met considerable interest in the community of concurrency and verification. Muscholl and Al. ([12],[13]) have studied the problem of matching for Message Sequence Charts, and shown some limits to the matching problem. For example, they have shows in [13], that it is undecidable whether the behavior of a HMSC H is included within the behavior of a HMSC H'. This result means that a behavioral weaving using the full HMSC language as a definition of pointcut is not possible. This is why we had to limit the expressiveness of our pointcut language as explained in Section 5.

7. CONCLUSIONS

This paper has proposed a definition of behavioral aspect weaving. Behaviors and aspects are defined using Message Sequence Charts and the weaving is performed at the semantic level. More precisely, an aspect defines a part of behavior that should be replaced by another one every time it appears in the semantics of the base specification. This approach suffers from several limitations: the matching process can only be performed if each join point appears inside a bounded fragment of a behavior. We have shown that this is always the case when pointcuts are defined with connected bMSCs. Another limitation is that the base HMSC should not exhibit two non-disjoint cycles where the pointcut matches. This second property is not a structural property of the base HMSC, and can only be checked during the weaving process. When it holds, we have a procedure to rewrite the base HMSC H into an equivalent HMSC H' using unfolding and commutations. The HMSC H' obtained is such that the semantics weaving $H \otimes A$ simply consists of a syntactic weaving in transitions of H'. Note that the property on loops is only a sufficient condition to apply the proposed algorithm. It does not mean that weaving is impossible for specifications that do not fulfill the loop condition. This has to be considered for further research. Another extension of this work is an improvement of the aspect description to deal with more abstract scenarios, or with different definition of matching (using for example the definition of [13]). Finally, we are currently implementing our matching and weaving algorithm within the UMLAUT model transformation framework.

8. REFERENCES

[1] J. Araujo, J. Whittle, and Kim. Modeling and composing scenario-based requirements with aspects. In *Proceedings of RE 2004*, Kyoto, Japan, September 2004.

[2] S. Clarke. *Composition of Object-Oriented Software Design Models*. PhD thesis, Dublin City University, 2001.

[3] R. Douence, P. Fradet, and M. Südholt. A framework for the detection and resolution of aspect interactions. In *Proceedings of GPCE'02*, LNCS. Springer, 2002.

[4] R. Douence, O. Motelet, and M. Südholt. A formal definition of crosscuts. In *Reflection'01*, pages 170–186, 2001.

[5] R. B. France, D.-K. Kim, S. Ghosh, and E. Song. A uml-based pattern specification technique. *IEEE Transaction on Software Engineering, vol.30(3), 193-206, March 2004*, 2004.

[6] O. Haugen. Comparing uml 2.0 interactions and msc-2000. In *Proceedings of SAM 2004*, pages 69–84. LNCS 3319, 2004.

[7] L. Hélouët and P. Le Maigat. Decomposition of Message Sequence Charts. In *Proceedings of SAM2000*, Grenoble, Juin 2000.

[8] J. Hopcroft and J. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley, Reading, Massachusetts, 1979.

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

[10] I. Jacobson and P.-W. Ng. *Aspect-Oriented Software Development with Use Cases*. Addison-Wesley, 2004.

[11] G. Kiczales. The fun has just begun. Keynote of AOSD'03, 2003.

[12] A. Muscholl. Matching specifications for Message Sequence Charts. In *Proceedings of FoSSaCS'99*, LNCS 1578, pages 273–287, 1999.

[13] A. Muscholl, D. Peled, and Z. Su. Deciding properties for message sequence charts. In *Proceedings of FOSSACS'98*, pages 226–242. Springer-Verlag, 1998.

[14] OMG. Uml superstructure specification, v2.0. OMG Document number formal/05-07-04, 2005.

[15] K. Ostermann, M. Mezini, and C. Bockisch. Expressive pointcuts for increased modularity. In *Proceedings of ECOOP'05*. Springer LNCS, 2005.

[16] A. Rashid, A. M. D. Moreira, and J. Araújo. Modularisation and composition of aspectual requirements. In *proceedings of AOSD'03*, pages 11–20, 2003.

[17] J. Whittle and J. Araújo. Scenario modelling with aspects. *IEE Proceedings - Software*, 151(4):157–172, 2004.