Semantic-based weaving of scenarios
Jacques Klein, Loïc Hélouët, Jean-Marc Jézéquel

To cite this version:

HAL Id: hal-00921480
https://inria.hal.science/hal-00921480
Submitted on 12 Mar 2019

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
The idea of combining, crossing, and weaving the behavioral model of a system with an existing scenario to create a new one is not new. Such weaving is supported by many modeling frameworks, especially model-driven approaches. However, there is no generic way to weave scenarios, so the whole process of weaving has to be defined for each framework. This paper presents a generic weaving process that can be used for various frameworks, including model-driven approaches. The weaving process is composed of several steps: first, the scenario to be woven is defined; then, the weaving points are identified; finally, the weaving process is performed.

1. Introduction

The process of weaving scenarios is defined in the following section. The weaving process consists of two steps: identifying weaving points and performing weaving. Identification of weaving points is based on the analysis of the weaving points in the weaving process. The weaving process is performed by applying a set of transformations to the weaving points. The set of transformations is defined in the weaving process. The weaving process is performed by applying a set of transformations to the weaving points. The set of transformations is defined in the weaving process.

Keywords: Weaving, Scenarios, Aspect-Oriented Modeling.

2. Related Work

The weaving process is based on the analysis of the weaving points. The weaving points are identified by analyzing the weaving process. The weaving process is performed by applying a set of transformations to the weaving points. The set of transformations is defined in the weaving process. The weaving process is performed by applying a set of transformations to the weaving points. The set of transformations is defined in the weaving process.

3. Conclusion

The weaving process is based on the analysis of the weaving points. The weaving points are identified by analyzing the weaving process. The weaving process is performed by applying a set of transformations to the weaving points. The set of transformations is defined in the weaving process. The weaving process is performed by applying a set of transformations to the weaving points. The set of transformations is defined in the weaving process.

4. Acknowledgments

The research described in this paper was partially supported by the ANR and the European Network of Excellence.
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 UML 2.0 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 \( M_e \) 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 High-level MSC with an alternative between the bMSCs Accept and Retry, where Propose and Accept are composed sequentially, Retry is in a loop, etc... High-level MSCs can be considered as automata labeled by bMSCs. They can be formally defined as follows:

DEFINITION 2 (High-level MSC). A High-level MSC is a graph \( H = (N, T, q_0, Q_{end}, M) \) where \( N \) is a set of nodes, \( T \subseteq N \times M \times N \) is a set of transitions, \( q_0 \) is the initial node, \( Q_{end} \) is a set of end nodes, and \( M \) is a finite set of bMSCs. For a transition \( t = (p, l, q) \in T \), node \( p \) will be called the origin of transition \( t \) and denoted by \( \lambda_o(t) \), node \( q \) will be called the goal of \( t \) and denoted by \( \lambda_g(t) \), and \( l \) will be called the label of \( t \) and denoted by \( \lambda_l(t) \). Furthermore, we will require that no transition of \( H \) leaves from an end node.

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

The notion of sequential composition noted \( \circ \) is central to understanding the semantics of High-level MSCs. 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 \( M_1 \) and \( M_2 \) can be concurrent in \( M_1 \circ M_2 \). 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 \( M_1 \circ M_2 = (E_1 \cup E_2, \leq_{1+2}, \alpha_1 \cup \alpha_2, \phi_1 \cup \phi_2, A_1 \cup A_2, \leq_1 \cup \leq_2, \lambda_1 \cup \lambda_2, \lambda_1_2, \phi_1(e_1) = \phi_2(e_2)) \).

To calculate the new partial ordering \( \leq_{1+2} \), 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, $\mathcal{B}$ 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 $M^* = \bigcup_{k \in \mathbb{N}} M^k$ with the convention that $M^0 = M_\emptyset$. When two bMSCs $M_1, M_2$ are such that $M_1 M_2 = M_2 M_1$, we will say that $M_1$ and $M_2$ are independent, and write $M_1 \parallel M_2$ (this situation occurs only when $\phi(E_1) \cap \phi(E_2) = \emptyset$). A bMSC $B$ will be called an atom if there does not exist two bMSCs $N \neq M$ and $M \neq M$, such that $B = M \parallel N$. 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 $M = X_1 \cdot X_2 \cdots \cdot X_n$, where all $X_i$'s are bMSCs, $X_1 \cdot X_2 \cdots \cdot X_n$ will be called a factorization of $M$.

An initial path of a bMSC $H$ is a sequence of transitions $p = t_1 t_2 \ldots t_k$ such that the origin of $t_1$ is the initial node $q_0$. If moreover the goal of $t_k$ is a node of $Q_{end}$, 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, $\lambda(p) = \lambda(t_1) \cdot \lambda(t_2) \cdots \cdot \lambda(t_k)$. The semantics of a bMSC $H$ 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](image)

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 indirectly 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 $M = X_1 \parallel P \parallel X_2$. 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 $k \geq 1$ bMSCs $\{X_i\}_{i \in \{1, \ldots, i+1\}}$ such that

- $M = X_1 \parallel P \parallel X_2 \parallel \ldots \parallel X_k \parallel P \parallel X_{k+1}$

- $\forall i, X_i \parallel P \parallel X_{i+1}$ 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 $M = X_1 \parallel P \parallel X_2 \parallel \ldots \parallel P \parallel X_{k+1}$, then the weaving of $M$ and $A$ is defined as $M \otimes A = X_1 \parallel Ad \parallel X_2 \parallel \ldots \parallel Ad \parallel X_{k+1}$. In the examples of Figures 1 and 2 the pointcut $P_1$ does not match the bMSC $Retry$ but matches the bMSC $Retry \parallel Retry$ once, the bMSC $Retry \parallel Retry \parallel Retry$ 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 $\pi(M) \subseteq E_M$ the projection of a bMSC $M$ on a single instance, and by $\pi_P(M)$ 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 override 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)


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 $P \not\succeq M$

1: for all $i \in I_P$ do
2: $w_i = \pi_i(M)$ /* a word consisting of all events on instance $i$ */
3: $V_i = \{v \in E^* | \exists u, w, w_i = u.v.w \land \alpha(v) = \alpha(\pi_i(P))\}$ /* $\forall v \in V_i$, 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
6: $\pi_{v_1}.\ldots.\pi_{v_{|I_P|}}(M) = P$ then
7: return $(\bigcup_{v \in V} w_i)$
else
9: return $(\emptyset)$
10: end if

To extend the notion of weaving to bMSCs, we have to deal with the following problem: for a path $p = t_1, \ldots, t_n$ of a bMSC $H$, when $P \not\succeq \lambda(p)$, 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 $\lambda(t_{i+1})$.

Let us again consider the bMSC $H$ in Figure 1, and its automaton representation in Figure 5-a. Consider also pointcut $P_1$ 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 $P_1$ matches the transition labeled by bMSC Propose. Since $P_1$ does not match Propose and since $P_1$ does not start to match Propose, we can continue our exploration from node $q_1$, and look at the next transitions, i.e. the transitions labeled by bMSC Accept and bMSC Retry. $P_1$ does not match Accept and since $q_2$ is an end node, we know that $P_1$ does not match after node $q_2$. $P_1$ 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 $t_1 = (q_1, Accept, q_2)$ and $t_2 = (q_1, Retry, q_1)$ in $H$, according to the semantics of bMSCs, we can consider a matching of $P_1$ in bMSCs $Retry \ast Accept$ and $Retry \ast Retry$. $P_1 \not\succeq Retry \ast Accept$, but $P_1 \succeq Retry \ast Retry$, and a new potential matching of $P_1$ 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 $M \cdot M'$. 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 $P_2$ matches bMSC $M$ once. $M$ can be written $X_1 \ast P_2 \ast X_2$, but $P_2$ starts to match the "end" of $M$. Indeed, if we add bMSC $N$ to $M$, $P_2$ matches $M \cdot N$ 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 $PM_{P_2,M}$ in Figure 6.
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 $H_{mp}$ (i.e. $\hat{L}(H) = \hat{L}(H_{mp})$) in such a way that the pointcut $P$ only matches a finite number of paths in $H_{mp}$. 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 $H_{mp}$ such that the weaving consists of rewriting only a finite number of paths of $H_{mp}$. This transformation of $H$ into $H_{mp}$ 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 $H_{pm}$, such that each node of $H_{pm}$ 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 $H_{pm}$. These parts will be called the future matches contained in $C$. This second step is performed by algorithm 3 which transforms the equivalent HMSC $H_{pm}$ into an equivalent HMSC $H_{pm}$, such that each node of $H_{pm}$ is a triple $(q, C, F)$, where $(q, C)$ is a node of $H_{pm}$ 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 $H_{pm}$. 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 $H_{pm}$ 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 hMSCs. 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 $H_{pm}$ such that each node of $H_{pm}$ is a pair $(q, P, M)$ where $q$ is a node of $H$, and $P, M$ 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 $p = (q_0, M_1, q_1, (q_1, M_2, q_2), (q_2, M_3, q_3))$, we will build the nodes $(q_0, M_1), (q_1, P, M_{p, 1}), (q_2, P, M_{p, 2})$, and $(q_3, P, M_{p, 3})$. These nodes can only be found one after another. The first pair (node, context) is obviously $(q_0, M_1)$. Then we can compute $P, M_{p, 1}$ and build the node $(q_1, P, M_{p, 1})$. To compute the potential part $P, M_{p, 1}$ of node $q_2$, it is not necessary to entirely consider $M_1$: considering the context $P, M_{p, 1}$ of node $q_2$ created previously is sufficient, since $P, M_{p, 1} = P, M_{p, 1}$. Finally, to compute $P, M_{p, 1} + M_{p, 2}$, we also use a previously discovered context attached to node $q_2$, that is to say, $P, M_{p, 1} + M_{p, 2}$, and we simply compute $P, M_{p, 1} + M_{p, 2}$. 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 3. Since this algorithm is just an unfolding of the initial graph, it is obvious that the set of executions generated by $H$ and $H_{pm}$ are equivalent. Note however that this algorithm terminates if and only if the number of potential matches for each node of $H$ is finite ( $N'$ is finite by definition ), which this is not always the case. This problem will be developed in section 5.

For the pointcut $P_1$ in Figure 2, algorithm 2 transforms the HMSC $H$ in Figure 5-a into the HMSC $H_{pm}$ in Figure 5-b where $M_1$ represents the empty bMSC and $P, M$ a bMSC with a single message "new attempt" from the instance customer to the instance server. The context associated with $q_1$ for path $p = (q_0, Propose, q_1)$ is $M_1$, since $P_1$ matches Propose neither fully nor partially. For any path ending with transition $(q_1, Retry, q_1)$, the context associated to $q_1$ is $P, M$ since $P_1$ starts to match the message "new attempt".

Algorithm 2 Context (P, H)

input: pointcut $P$, HMSC $H = (N, T, q_0, Q_{end}, M)$
output: HMSC $H_{pm} = (N', T', q_{t_0}, Q_{end'}, M_{t_1})$

1: $M_{t_1} = M$, $q_{t_0} = (q_0, M_1)$
2: $CurrentStates = \{q_{t_0}\}$, $futureStates = \emptyset$
3: $N_{t_0} = \{q_{t_0}\}$, $T_{t_0} = \emptyset$
4: while $CurrentStates \neq \emptyset$ do
5: for all $q_t = (q, M_t) \in CurrentStates$ and $t = (q, M_t, q') \in T$ do
6: $q'_{t_1} = (q', P, M_{p, 1, M})$
7: if $q_{t_1} \notin N_{t_0}$ then
8: $futureStates = futureStates \cup q_{t_1}$
9: $N_{t_1} = N_{t_0} \cup q_{t_1}$
10: end if
11: $T_{t_1} = T_{t_0} \cup T_t$
12: if $T_{t_1} \neq T_{t_0}$ then
13: $T_{t_1} = T_{t_0} \cup T_t$
14: end if
15: end for
16: $CurrentStates = futureStates, futureStates = \emptyset$
17: end while
18: $Q_{end'} = \{n, M \in N' \mid n \in Q_{end}\}$
19: return $H_{pm} = (N', T', q_{t_0}, Q_{end'}, M_{t_1})$
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 $M \cdot M'$. Consider for example the bMSC $M$ and the pointcut $P_2$ in Figure 6, and the bMSC $M'$ in Figure 7. The bMSC $M \cdot M'$ 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 $M \cdot M'$, 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](image)

Similarly, for a peculiar node $q_+ = (q, C)$ of a HMSM $H_+$, the part of $C$ that will be used to find a join point depends on the path that may start from $q$. If $q_+ \in Q_{end+d}$, it is obvious that the future match associated to the node $(q, C)$ is $M_+$ since no extension of a path ending in $q_+$ is possible. If $q_+ \notin Q_{end+d}$, 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 HMSM (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. $F(q_+) = M_+$ if $q_+ \in Q_{end+d}$.
2. $F(q_+) = FM_{P,C,M} \cup (C \cap F')$ if $q_+ \notin Q_{end+d}$, 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 HMSM immediately follows. This algorithm transforms a HMSM $H_x$ into an equivalent HMSM $H_+$ such that each node of $H_+$ is a triple $(q, C, F')$ where $(q, C)$ is a node of $H_x$, and $F'$ 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 HMSM. The main difference is that this unfolding starts from end nodes.

Consider the HMSM $H$ and the pointcut $P$ in Figure 8. Figure 9 presents $H$ as an automaton labeled by bMSC names, and the HMSMs $H_+$ and $H_+$, obtained by applying algorithms 2 and 3. In $H_+$, $C$ is the context of node $q_1$. $C_1 = X$, and after each transition $(q_i, Y, q_i)$, $C_1$ remains unchanged. In $H_+$, node $(q_1, C_1)$ 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 HMSM $H_+$ obtained from the HMSM $H_+$ 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 HMSM $H$ and a pointcut $P$ (condition for convergence of 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 $H_+ = (N_+, T_+, q_{+0}, Q_{end+d_+, M_+})$

**output:** HMSM $H_+ = (N_+, T_+, q_{+0}, Q_{end+d_+}, M_+)$

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

4.3 Unbreakable loops

From a pointcut $P$ and a HMSM $H$, we have built a HMSM $H_+$ whose nodes contain a context and a future. As we want to isolate the join points into a finite number of paths of
a transformed HMSC, the next thing to do is to rewrite $H_\pm$ into an equivalent HMSC where the number of paths starting from nodes with an empty future $M_e$ and ending on nodes with also an empty future is finite. Algorithm 5 builds the set $P_m$ of these paths and defines them more formally. This set $P_m$ ensures that $P$ matches entirely bMSC $X(t)$ associated to any path $p$ of $P_m$ a certain number of times, and that a potential matching is not initiated from this path.

However, algorithm 5 cannot be directly applied on $H_\pm$. If $H_\pm$ 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: $\text{Retry} \bullet \text{Retry} \bullet \text{Accept}, \text{Retry} \bullet \text{Retry} \bullet \text{Retry} \bullet \text{Accept}, \text{etc}.$

So, before using Algorithm 5, we have to break the cycles of $H_\pm$ that loop infinitely on nodes with non-empty future. Let us denote by $C_H$ the set of elementary cycles of $H_\pm$ and $UC_H$ the set of “unbreakable” elementary cycles, i.e. cycles for which algorithm 5 generates an infinite set of paths. Formally, we have $UC_H = \{c = t_1 t_2 ... t_k \in C_H | \forall i \in \{1...k\}, FM(\lambda - (t_i)) \neq M_e\}$. 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 $t = (q_{k-1}, \text{Retry}, q_{k})$ can be split into two transitions $t' = (q_{k-1}, \text{Retry}_1, q_{new})$ and $t'' = (q_{new}, \text{Retry}_2, q_{k})$ such that $\text{Retry}_1$ is a bMSC with a single message ‘try again’ and $\text{Retry}_2$ 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 $M_e$.

However, things are not always so easy. Consider for example HMSC $H_3$ of Figure 11: pointcut P3 matches an infinite number of bMSCs ($X1Y2, X1Y1X2, X1Y1Y1X2, ...$), 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 $H_\pm$ be a HMSC augmented with potential and futures matches, $P$ be a pointcut. Let $UC_H$ be the set of unbreakable cycles of $H_\pm$. An unbreakable generator of $H_\pm$ and $P$ is an acyclic path $p = t_1 ... t_k$ of $H_\pm$ that:

- starts from a node with an empty future, i.e., $F(\lambda - (t_1)) = M_e$,
- ends on a node with empty future, i.e., $F(\lambda + (t_k)) = M_e$,
- never passes through a node with empty future other than $\lambda - (t_1)$ and $\lambda + (t_k)$, i.e., $\forall i \in \{2 ... k\}, F(\lambda - (t_i)) \neq M_e$
atoms $\Sigma_{A1}$ is an expression of the form $E := \sigma.E \mid (E') \mid E1 + E2 \mid M$, where $\sigma \in \Sigma_{A1}$ 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 $E = X_1 \ldots X_n$ just by computing the potential match of a pointcut $P$ in $X_1 \ldots X_n$. For any sub-expression $E' = X_i \ldots X_j$ of $E$, we can also compute a future match $Future(E')$ similarly to what has already been proposed for HMSCs: $Future(E') = \{P \in P_E, P \supseteq X_{i+1} \ldots X_j\}$.

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

$$E_p = X_{n1} \ldots X_{nt1} (Y_{n1} \ldots Y_{nt_1})^* X_{n2} \ldots X_{nt_2} (Y_{n2} \ldots Y_{nt_2})^* \ldots (Y_{nq} \ldots Y_{nt_q})^* X_{n+1} \ldots X_{n+q}$$

where all $X_{ij}$ and $Y_{ij}$’s are atoms, and that generates the same set of behaviors as $\lambda(t_1) \cdots \lambda(t_i) \cdots \lambda(c_i)^* \cdots \lambda(t_q) \cdots \lambda(c_j) \cdots \lambda(t_r)\}$, where for all $i, j \in 1, k, 1 \leq i \leq j \leq r$, $\lambda(t_i) = \lambda(c_j)$. 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 $p = \{q_{x0}, X_1, q_{x1}, \ldots, q_{x2}\}$ is an unbreakable generator since the cycle $c = \{q_{x0}, X_1, q_{x1}\}$ 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 $E_p = X_1.Y_1^* \ldots X_3$.

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 $E_p = X_1 \ldots X_j Y_{j+1} \ldots Y_{t_1} X_{t_2} \ldots X_{t_r}$ is equivalent to the expression $E_p = Y_{j+1} \ldots X_{t_1} X_{t_2} \ldots X_{t_r}$ for which the path $p = \{q_{x0}, X_1, q_{x1}, \ldots, q_{x2}\}$ is no longer unbreakable.

We will use several types of permutations: Let $E = X_1 \ldots X_n$ 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 $E' = X_1 \ldots X_{n-1} X_n X_1 \ldots X_{n-1}$ whenever $X_n$ and $X_n$ are independent. Indeed, when both bMSCs $M$ and $N$ are independent, then $M \otimes N = N \otimes M$, 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.

**Figure 11: A loop problem solved with a permutation**

- crosses an unbreakable cycle of $H_k$, i.e. $\exists j \in 2, k, c = t_1 \ldots t_q \in U \cup U_{k, \omega}$ such that $\lambda(t_j) \in \bigcup_{i=1 \ldots \omega} \lambda(t_i)$
- $P \mathcal{B}^+ \lambda(p)$

Let $\mathcal{P}_H(H_k)$ be the set of unbreakable generators of $H_k$. 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, we are 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. $c = t_1 \ldots t_i, c' = t_1 \ldots t_j \in U \cup U_{\omega, \omega}$, $\bigcup_{i=1 \ldots \omega} \lambda(t_i) \cap \bigcup_{j=1 \ldots \omega} \lambda(t_j) = \emptyset$. This transformation is better described using regular expressions.

**Definition 7.** A regular expression on an alphabet of
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 $\Sigma_{at}$, we will denote by $[E]$ its equivalence class, i.e. the least set of expression on $\Sigma_{at}$ 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 $a(ba')^*bc = (ab)^* (a^*b)c$ 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 $\Sigma$ of bMSC, 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 $E_1 \setminus E_2$ that generates words of $L_1 \setminus L_2 = \{ w \in \Sigma^* | \exists v \in L_2 \wedge v.w \in L_1 \}$.

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

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 $H = (\Sigma, T, Q_0, Q_{exit}, M)$ be a HMSC, $E$ be a regular expression over $\mathcal{M}^*$, and $E'$ be an equivalent regular expression over $\Sigma = \mathcal{M} \cup \mathcal{A}(\mathcal{M})$. The HMSC obtained by replacement of $E$ by $E'$ in $H$ will be denoted $H_{E \rightarrow E'}$, and is the HMSC $H_{E\rightarrow E'}$ associated with the regular expression $E_{new} = (E_H/(E, \mathcal{M}^*)) . E'. (E_H \setminus (\mathcal{M}^*. E)) \cup (E_H \cap \mathcal{M}^*. \mathcal{M}^* - E)$.  

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 unsplittable 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 $H_p$ with all its connected unsplittable loops.

Note that for all generators $p$ of a HMSC $H_p$, 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 $A = (P, \mathcal{A})$, and a HMSC $H_{p}$ augmented with potential and future matches computed from a HMSC $H$ and the pointcut $P$, we compute the set $\mathcal{P}_p(H_{p})$ of unbreakable generators of $H_{p}$, and the set $\mathcal{P}_p$ of the associated expressions. Then, if for all $E_p \in \mathcal{P}_p$, 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 $H_{mp}$ equivalent to $H_p$ that does not contain infinite path passing infinitely often through non-empty futures. If not, the weaving cannot be performed. Finally, when $H_{mp}$ can be computed, each expression $E_p$ related to an 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($H_x, U\mathcal{C}_{in}$)

input: a HMSC $H_x$ with contexts and futures $\mathcal{U}\mathcal{C}_{in}$, set of unbreakable cycles

output: a HMSC $H_{mp}$ without unbreakable generators

1: Compute $\mathcal{P}_p(H_x)$, set of unbreakable generators
2: $\mathcal{P}_\mathcal{P} = \{ E_p | p \in \mathcal{P}_p(H_x) \}$
3: Compute $E_H$, regular expression associated to $H$
4: for all $E_p \in \mathcal{P}_\mathcal{P}$ do
5: Compute $[E_p]$
6: if $\not\exists e \in [E_p]$ 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: $Re = Rep \cup \{ (E_p, e) \}$
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: $E_H = (E_H/(E_p, \mathcal{M}^*)) . e . (E_H \setminus (\mathcal{M}^* . E)) \cup (E_H \cap \mathcal{M}^* . \mathcal{M}^* - E)$
16: end for
17: $H_{mp} = H_E *$ compute the HMSC equivalent to the expression $E_H$/*
18: return($H_{mp}$)

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 $p = (q_{01}, X_1, q_{02})(q_{02}, Y, q_{03})(q_{03}, X_2, q_{02})$ 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 $I_1$ and $Y_b$ is a bMSC with a single local event ‘b’ on instance $I_3$. 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 expression future is to consider the expression $E'_p = (Y_a)^*.X_1.Y_a(Y_a)^*.X_2$. The equivalence class of $E'_p$ contains neither a splittable nor a null-match cycle. The only way to obtain an expression future is to consider the expression $E'_p = (Y_a)^*.X_1.Y_a(Y_a)^*.X_2$. However, $E'_p \not\in [E_p]$, 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 $H_{mp}$ and woven HMSC

So far, if algorithms 2, 3 and 4 have succeeded, we have obtained a HMSC $H_{mp}$ 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 $H_{mp}$ 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 $H_{mp}$. 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 $H_{mp}$ such that for each path $p \in \mathcal{P}_m$:

i) $p$ starts from a node with future match equal to $M_1$,
ii) $p$ terminates on a node with future match equal to $M_1$,
iii) All other nodes of $p$ are labeled by future matches different from $M_1$.

The set $\mathcal{P}_m$ is then a set of paths of $H_{mp}$ such that for each path $p \in \mathcal{P}_m$:

i) $p$ starts from a node with future match equal to $M_1$,
ii) $p$ terminates on a node with future match equal to $M_1$.
iii) All other nodes of $p$ are labeled by future matches different from $M_1$.
Figure 12: Impossible weaving

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: $M_3$ and $P.M$. Consider now HMSC $H'$ and pointcut $P$ 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 $(q_0, M_3)$, the potential match obtained in $M_3 \cdot G$ contains a single local event 'a'

Algorithm 5 BuildP($P, H_{mp}$)

input: pointcut $P$.
HMSC $H_{mp} = (\lambda_{mp}, T_{mp}, q_{init_{mp}}, Q_{end_{mp}}, \mathcal{M}_{mp})$.
output: $\mathcal{P}_m$ set of matched paths.

1: $\mathcal{P}_m = \{(q_x, M, q_x) \in T_{mp} \mid F(q_x) = F(q_x) = M_x \wedge \exists k \in \mathbb{N}, P \aleph^k M_x \}$
2: $Path = \{(q_x, M, q_x) \in T_{mp} \mid F(q_x) = M_x \wedge F(q_x) \neq M_x \}$
3: while $Path \neq \emptyset$ do
4: $FuturePath = \emptyset$
5: for all $p = t_1, t_2, \ldots, t_k \in Path$ do
6: for all $(q_x, M, q_x) \in T_{mp}$, such that $q_x = \lambda_x(t_k)$
7: if $F(q_x) = M_x$ then
8: $\mathcal{P}_m = \mathcal{P}_m \cup (p.t)$
9: else
10: $FuturePath = FuturePath \cup (p.t)$
11: end if
12: end for
13: end for
14: $Path = FuturePath$
15: end while
16: return $\mathcal{P}_m$

Algorithm 6 Weave(H,A)

input: a HMSC $H = (\lambda, T, q_0, Q, M)$, a behavioral aspect $A = (P, Ad)$
output: a woven HMSC $H'$.

1: $H' = Context(P, H)$
2: $H' = Future(P, H_2)$
3: compute $\mathcal{P}_m$
4: if $\forall c, c' \in \mathcal{P}_m, c$ and $c'$ are disjoint then
5: $H_{mp} = Permutation s(H_x, \mathcal{U}C_{H_x})$
6: $\mathcal{P}_m = BuildP(P, H_{mp})$
7: for all $p \in \mathcal{P}_m$ do
8: Compute $E_p$
9: Find the minimal factorization $x_1 \cdot P \cdot x_2 \cdots \cdot P \cdot x_k$
10: $E'_p := x_1 \cdot Ad \cdot x_2 \cdots \cdot Ad \cdot x_k$
11: $H_{mp} = H_{mp} \{x_1 \cdot E'_p \}$
12: end for
13: Return $H_{mp}$
14: else
15: Weaving cannot be performed ; STOP
16: end if
(bMSC A1). From the node \((q_0, A_1)\), the potential match relating to \(A_1 \cdot G\) 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 \(An \cdot G\) 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 HMSMC HA, and aspect \(A = (P_4, Adv)\) the semantics of weaving is a set of behaviors containing scenarios of the kind \(Adv \cdot A_1^n, n \in \mathbb{N}\). This expression is not a regular expression on a set of bMSCs, and hence cannot be represented by a HMSMC.

![Figure 13: An example where algo. 2 does not terminate.](image)

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 \(M\)) such that \(M = M1 \cdot M2\) (different from \(M\)). For example, in Figure 2, the bMSC P4 is not connected (\(P4 = P9 \cdot PB\) with \(P9\) a bMSC with only a local event 'a' and \(PB\) a bMSC with only a local event 'b', and \(P9\) \(\neq PB\)).

An interesting property is that for a HMSMC \(H\) and an aspect \(A = (P, Adv)\), 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 of that 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.](image)

### 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 addressing 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 shown in [13], that it is undecidable whether the behavior of a HMSC $H$ is included inside 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 HMSCs. 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 semantics, 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


