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.
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 gaining 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 Aspect-J 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...
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 $M_0$ 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 $H = (N, T, q_0, Q_{end}, M)$ where $N$ is a set of nodes, $T \subseteq N \times N \times N$ 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 a transition $t$ and denoted by $\lambda_o(t)$, node $q$ will be called the goal of $t$ and denoted by $\lambda_g(t)$. 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 $M_1$ and $M_2$ can be concurrent in $M_1 \bullet M_2$. 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 $M_1 \bullet M_2 = (E_1 \cup E_2, \leq_{1+2}, A_1 \cup A_2, A_1 \cup A_2, 1 \leq_1 \leq_2$, where: $\leq_{1+2} = \leq_1 \cup \leq_2 \cup \{(e_1, e_2) \in E_1 \times E_2 | \phi_1(e_1) = \phi_2(e_2)\}$.
this definition, $\forall$ 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 $\cup_{k \geq 0} M^k$ with the convention that $M^0 = M_0$. When two bMSCs $M_1, M_2$ are such that $M_1 \cdot M_2 = M_2 \cdot M_1$, 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 $B = M \cdot N$. For a set of bMSCs $M$, we will denote by $At(M)$ the set of all atoms of bMSCs in $M$. Note that an algorithm to find a decomposition of a bMSC into atoms can be found in [7]. For a bMSC $M$ such that $M = X_1 \cdot X_2 \cdot \ldots \cdot X_k$, where all $X_i$'s are bMSCs, $X_1 \cdot X_2 \cdot \ldots \cdot X_k$ will be called a factorization of $M$.

An initial path of a HMSC $H$ is a sequence of transitions $p = t_1, t_2, \ldots, t_k$ 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, $\lambda(p) = \lambda(t_1) \cdot \lambda(t_2) \cdot \ldots \cdot \lambda(t_k)$. 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 $L(H) = \{\lambda(p) \mid p \text{ is an accepting path of } H\}$.

![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 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 $L(H)$. More formally, after weaving we should have $L(H') = (M \otimes A)M \in L(H)$. 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 $M = X_1 \cdot P \cdot X_2$. 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 $\{X_i\}_{i \in \{1, \ldots, k+1\}}$ such that

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

- $\forall i, X_i \cdot P \cdot X_{i+1}$ 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 $P \triangleright M$, i.e. if $M$ is of the form $M = X_1 \cdot P \cdot X_2 \cdot \ldots \cdot P \cdot X_{k+1}$, then the weaving of $M$ and $A$ is defined as $M \otimes A = X_1 \cdot Ad \cdot X_2 \cdot \ldots \cdot Ad \cdot X_{k+1}$. In the examples of Figures 1 and 2 the pointcut $P_1$ does not match the bMSC $Retry$ but matches the bMSC $Retry \bullet Retry$ 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 $\pi(M) \subseteq E_M$ the projection of a bMSC $M$ on a single instance, and by $\pi_F(M)$ 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 overlap 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 $P = (P_F, E_F, \leq_F, A_F, \alpha_F, \phi_F)$, 
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 $P \not\mathbin{\#} M$)

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

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

bMSC $M$ once. $M$ can be written $X_1 \bullet P_2 \bullet X_2$, but $P_2$ starts to match the "end" of $M$. Indeed, if we add bMSC $N$ to $M$, $P_2$ 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 $PM_{P_2,M}$ 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 $H_{emp}$ (i.e., $\mathcal{L}(H) = \mathcal{L}(H_{emp})$) in such a way that the pointcut $P$ only matches a finite number of paths in $H_{emp}$.

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 $H_{emp}$ such that the weaving consists of rewriting only a finite number of paths of $H_{emp}$. This transformation of $H$ into $H_{emp}$ 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 $H_{emp}$, such that each node of $H_{emp}$ 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 $H_{emp}$. These parts will be called the future matches contained in $C$. This second step is performed by algorithm 3 which transforms an HMSC $H_{emp}$ into an equivalent HMSC $H_{emp}$, such that each node of $H_{emp}$ 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 $H_{emp}$. 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 $H_{emp}$ 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 Step 1: 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 $H_{emp}$ such that each node of $H_{emp}$ 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 $(q_0, M_1), (q_1, P, M_{1.1}), (q_2, P, M_{1.2, M_2})$, and $(q_3, P, P, M_{1.3, M_2, M_3})$. These nodes can only be found one after another. The first pair (node, context) created is obviously $(q_0, M_1)$. Then we can compute $P, M_{1.1}$ and build the node $(q_1, P, M_{1.1})$. To compute the potential part $P, M_{1.1}$ of node $q_2$, it is not necessary to entirely consider $M_1$: considering the context $P, M_{1.1}$ of node $q_2$ created previously is sufficient, since $P, M_{1.1, M_2} = P, P, M_{1.1, M_2}$. Finally, to compute $P, M_{1.2, M_2, M_3}$, we also use a previously discovered context attached to node $q_2$, that is to say, $P, M_{1.2, M_2}$, and we simply compute $P, P, M_{1.2, M_2, M_3}$. 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 $H_{emp}$ 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 $P_1$ in Figure 2, algorithm 2 transforms the HMSC $H$ in Figure 5-a into the HMSC $H_{emp}$ in Figure 5-b where $M_i$ 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 $M_{1.1}$, 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 $H_{emp} = (N_{emp}, T_{emp}, q_{emp}, Q_{end_{emp}}, M_{emp})$

1: $M_{emp} = M, q_{emp} = (q_0, M_0)$
2: $CurrentStates = \{q_{emp}\}$, futureStates = $\emptyset$
3: $N_{emp} = \{q_{emp}\}$, $T_{emp} = \emptyset$
4: while CurrentStates $\neq \emptyset$ do
5: for all $q_s = (q, M_s) \in$ CurrentStates and $t = (q, M_s, q') \in T$ do
6: $q'_s = (q', P, M_{emp}, M)$
7: if $q'_s \notin N_{emp}$ then
8: futureStates = futureStates $\cup q'_s$
9: $N_{emp} = N_{emp} \cup q'_{emp}$
10: end if
11: $N_{emp} = N_{emp} \cup t_{emp}$
12: if $t_{emp} \notin T_{emp}$ then
13: $T_{emp} = T_{emp} \cup t_{emp}$
14: end if
15: end for
16: CurrentStates = futureStates, futureStates = $\emptyset$
17: end while
18: $Q_{end_{emp}} = \{(n, M) \in N_{emp} \mid n \in Q_{end}\}$$
19: return $H_{emp} = (N_{emp}, T_{emp}, q_{emp}, Q_{end_{emp}}, M_{emp})$
4.2 Step2: Future matches

For a pointcut $P$ and two bMSCs $M$ and $M'$, the context $P_{M,M'}$ is not always entirely used when $P$ matches $M \bullet M'$. Consider for example the bMSC $M$ and the pointcut $P_2$ in Figure 6, and the bMSC $M'$ in Figure 7. The bMSC $M \bullet M'$ is depicted in Figure 7 and we can see that the part of the context $C = P_{M,M'}$ 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 $FM_{P,C,M'}$.

![Diagram of Future Match](image)

**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 $q_+ \in Q_{end,+}$, 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 $q_+ \notin Q_{end,+}$, 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. $F(q_+) = M_+$ if $q_+ \in Q_{end,+}$.
2. $F(q_+) = FM_{P,C,M} \cup (C \cap F')$ if $q_+ \notin Q_{end,+}$, 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 $H_+$, such that each node of $H_+$ 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 $H_\ell$ obtained by applying algorithms 2 and 3. In $H_+$, $C_1$ is the context of node $q_1$. $C_1$ is $X$, and after each transition $(q_1,Y,q_2)$, $C_1$ remains unchanged. In $H_\ell$, 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 HMSC $H_\ell$ 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 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 $H_+$**

**Output:** HMSC $H_\ell = (N_\ell,T_\ell,q_{\ell,0},Q_{\ell,end},M_\ell)$

1. $M_\ell = M_+, Q_{\ell,end} = \{ (n,M,M_n) | (n,M) \in Q_{end,+} \}$
2. $CurrentStates = Q_{\ell,end}, futureStates = \emptyset$
3. $N_\ell = CurrentStates, T_\ell = \emptyset$
4. While $CurrentStates \neq \emptyset$
5. For all $q_\ell = (q,C,F) \in CurrentStates$ and $t_+ = (\langle q_+,C'_+ \rangle, M, C, q') \in T_+$
6. $F' = FM_{P,C,M} \cup (C' \cap F')$
7. $q_\ell' = (q',C',F')$
8. If $q_\ell' \notin N_\ell$ then
9. $futureStates = futureStates \cup q_\ell$
10. $N_\ell = N_\ell \cup q_\ell$
11. End if
12. $t_\ell = (q_\ell,M,q_\ell')$
13. If $t_\ell \notin T_\ell$ then
14. $T_\ell = T_\ell \cup t_\ell$
15. End if
16. End for
17. $CurrentStates = futureStates, futureStates = \emptyset$
18. End while
19. $q_{\ell,0} = (q_0,M_\ell, M_\ell)$
20. Return $H_\ell$

4.3 Unbreakable loops

From a pointcut $P$ and a HMSC $H$, we have built a HMSC $H_\ell$ 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
$H_x$ 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 $\mathcal{P}_m$ 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 $H_x$. If
$H_x$ 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: \textit{Retry} $\bullet$ \textit{Retry} $\bullet$ \textit{Accept}, \textit{Retry} $\bullet$ \textit{Retry} $\bullet$ \textit{Retry} $\bullet$ \textit{Accept},
etc...

So, before using Algorithm 5, we have to break the cy-
cles of $H_x$ that loop infinitely on nodes with non-empty
future. Let us denote by $\mathcal{C}_H$, the set of elementary cycles
of $H_x$ and $\mathcal{U}C_{H_x}$ the set of "unbreakable" element-
cycles, i.e. cycles for which algorithm 5 generates an infinite
set of paths. Formally, we have $\mathcal{U}C_{H_x} = \{c = t_1, t_2, \ldots t_k \in
\mathcal{C}_H, \forall t_i \in \{1, \ldots k\}, FM(\lambda(\lambda(t_i)) \neq M_e \}$. Sometimes,
braking a cycle just consists of splitting a bMSC labeling a tra-
nsition into smaller parts, so that an empty future appears.
For instance, in Figure 10, the transition $t = (q_0, q_1, \textit{Retry}, q_2, q_3)$
can be split into two transitions $t' = (q_0, q_1, \textit{Retry}, q_2, q_3)$
and $t'' = (q_2, q_3, \textit{Retry}, 2, q_4, q_5)$ such that $\textit{Retry}1$ is a bMSC
with a single message 'try again' and $\textit{Retry}2$ is a bMSC
containing a single message 'new attempt'. After this trans-
formation, 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 ex-
ample HMSC $H_3$ of Figure 11: pointcut $P_3$ matches an in-
finite number of bMSCs $(X1X2, X1Y1X2, X1Y11X12, \ldots)$,
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 cy-
cle 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 $H_{3'}$ in Figure 11. Note that within this new HMSC
the matching of pointcut $P$ 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 $H_x$ be a HMSC augmented with po-
tential and futures matches, $P$ be a pointcut. Let $\mathcal{U}C_{H_x}$ be
the set of unbreakable cycles of $H_x$. An unbreakable gen-
erator of $H_x$ and $P$ is an acyclic path $p = t_1 \ldots t_k$ of $H_x +$
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. $\forall i \in 2..k, F(\lambda_-(t_i)) \neq
  M_e$

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figure8.png}
\caption{A HMSC}
\end{figure}

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figure9.png}
\caption{Transformation of $H$ into $H_x$}
\end{figure}

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figure10.png}
\caption{HMSC $H_x$ related to the HMSC $H_x$ in Figure 5}
\end{figure}
Atoms $\Sigma_{A1}$ is an expression of the form $E := \sigma \cdot E | \{(E')^* | E_1 + E_2 | M\}$, where $\sigma \in \Sigma_{A1}$ is an atomic bMSC, $(E')^*$ denotes iteration, and $E_1 + E_2$ 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 $E = X_1 \ldots X_k$, just by computing the potential match of a pointcut $P$ in $X_1 \ldots X_k$. For any sub-expression $E' = X_i$, $X_j$, of $E$, we can also compute a future match $Future(E')$ similarly to what has already been proposed for HMSCs: $Future(E') =$ $P M_{P,E',X_i} \ldots X_k$.

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

$$E_p = X_{i_1} \ldots X_{i_q}, (Y_{b_1} \ldots Y_{b_k})^* X_{j_1} \ldots X_{j_{k+1}} (Y_{21} \ldots Y_{2n_1})^* \ldots (Y_{q1} \ldots Y_{q_k})^* X_{c_1} \ldots X_{c_q}$$

where all $X_i$ and $Y_j$'s are atoms, and that generates the same set of behaviors as $\lambda(t_1) \cdots \lambda(t_i) \lambda(t_{i+1}) \cdots \lambda(t_j) \cdots \lambda(t_q)$, where for all $i, j \in 1, k, \lambda(t_i) = \lambda(c_j)$. Note that this expression is never of the form $E_1 + E_2$.

Consider for instance the HMSC $H3$ and the pointcut $P3$ in Figure 11. The path $p = (q_{x_1}, X_1, q_{x_2}, (q_{x_3}, X_2, q_{x_4}))$ is an unbreakable generator since the cycle $c = (q_{x_1}, Y_1, q_{x_2})$ 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 $E_p = (Y_1)^* X_1 X_2$, which is the path $p = (q_{x_1}, X_1, q_{x_2}, q_{x_3}) (q_{x_1}, X_2, q_{x_4})$ is no longer unbreakable.

We will use several types of permutations: Let $E = X_{i_1} X_{i_2} \ldots X_{i_k}$ be a regular expression on a set of bMSCs. A simple permutation is to exchange the respective order of two bMSCs when possible, $E$ can be rewritten into an equivalent regular expression $E' = X_{i_1} X_{i_2} \ldots X_{i_{k+1}} X_{i_k} X_{i_{k+2}}$ whenever $X_{i_{k+1}}$ and $X_{i_k}$ are independent. Indeed, when two bMSCs $M$ and $N$ are independent, then $M \cdot N = N \cdot M$, 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. $\exists j \in 2, k, c = t_1' \ldots t_q' \in \mathbb{U} C H_k$ such that $\lambda_- (t_j) \in \bigcup_{i \in 1, \sigma} \lambda_- (t_j')$
- $P \not\rightarrow \lambda(p)$

Let $\mathcal{P}_u(H_k)$ 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. $\forall c = t_1 \ldots t_i, c' = t_1 \ldots t_j \in \mathbb{U} C H_k$, $\bigcup_{i \in 1, \sigma} \lambda_- (t_i) \cap \bigcup_{j \in 1, \sigma} \lambda_- (t_j) = \emptyset$. This transformation is better described using regular expressions.

Definition 7. A regular expression on an alphabet of atoms $\Sigma_{A1}$ is an expression of the form $E := \sigma \cdot E | \{(E')^* | E_1 + E_2 | M\}$, where $\sigma \in \Sigma_{A1}$ is an atomic bMSC, $(E')^*$ denotes iteration, and $E_1 + E_2$ is an alternative.
When two regular expressions $E$ and $E'$ are equivalent we will note $E \approx E'$. For a given expression $E$ on a set of atoms $\Sigma_A$, we will denote by $[E]$ its equivalence class, i.e. the last set of expression on $\Sigma_A$ containing $E$ such that $\forall \epsilon \in [E], \forall \epsilon' \approx \epsilon \in [E]$. In addition to the permutations, we can use a classical property of regular expressions such as $a(b(a)')bc = ab(ab)c'$ 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 $\Sigma$ 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 $E_1 \setminus E_2$ that generates words of $L_1 \setminus L_2 = \{w \in \Sigma^* | \exists v \in L_2 \land wv \in L_1\}$.

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

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 $H = (N, T, Q_0, Q_{end}, M)$ be a bMSC, $E$ be a regular expression over $M^*$, and $E'$ an equivalent regular expression over $M = M \cup M^*$. The bMSC obtained by replacement of $E$ by $E'$ in $H$ will be denoted $H_{E \to E'}$, and the bMSC $H_{E_{new}}$ associated with the regular expression $E_{new} = (E_H \cup (E, M^*))E'\cdot(E_H \setminus (M^* \cdot E)) \cup (E_H \cap M^* \cdot E - M^*)$.

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 $E_\sigma$ 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 $E_\sigma$ rewrites only a single path of $H_{\sigma}$ with all its connected unsplittable loops.

Note that for all generators $p$ of a bMSC $H_{\sigma}$, since all expressions $e$ in $[E_\sigma]$ generate the same scenarios, all bMSCs obtained by replacing an expression $e \in [E_\sigma]$ by another expression $e' \in [E_\sigma]$ are equivalent.

For a behavioral aspect $A = (P, A_d)$, and a bMSC $H_{\sigma}$ augmented with potential and future matches computed from a bMSC $H$ and the pointcut $P$, we compute the set $P_\sigma(H_{\sigma})$ of unbreakable generators of $H_{\sigma}$, and the set $P_{exp}$ of the associated expressions. Then, if for all $e \in P_{exp}$, there is an expression $e \in [E_\sigma]$ in which all starred subexpressions of $e$ are either splittable or null-match, then there exists a bMSC $H_{\sigma_{\exp}}$ equivalent to $H_{\sigma}$ that does not contain infinite path passing infinitely often through non-empty futures. If not, the weaving cannot be performed. Finally, when $H_{\sigma_{\exp}}$ 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($H_{\sigma}, uC_{H_{\sigma}}$)

**Input:** a bMSC $H_{\sigma}$ with contexts and futures $uC_{H_{\sigma}}$, set of unbreakable cycles

**Output:** a bMSC $H_{\sigma_{\exp}}$ without unbreakable generators

1: Compute $P_{\sigma}(H_{\sigma})$, set of unbreakable generators
2: $P_{exp} = \{E_p | p \in P_{\sigma}(H_{\sigma})\}$
3: Compute $E_{\sigma}$ regular expression associated to $H$
4: for all $e \in P_{exp}$ do
5: Compute $[E_p]$,
6: if $\exists e \in [E_p]$ such that $\forall e$ 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 $c$ in $E_{\sigma}$ */
15: $E_{\sigma} = (E_H \cup (E_p, M^*))c.(E_H \cap (M^* \cdot E_p)) + (E_H \cap (M^* \cdot E_p - M^*))$
16: end for
17: $H_{\sigma_{\exp}} = H_{E_{\sigma}} /*$ compute the bMSC equivalent to the expression $E_{\sigma}$ */
18: return($H_{\sigma_{\exp}}$)

Note that some generators do not have equivalent expressions with splittable or null-match cycles. Consider the bMSC $H$ and the pointcut $P$ in Figure 12. The path $p = (q_{1,0}, X_1, q_{2})/(q_{1,0}, Y, q_{2})/(q_{1,0}, X_2, q_{3})$ 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 $I_1$ 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 empty future is to consider the expression $E_p' = (Y_a \cdot X_1 \cdot X_2 \cdot Y_b \cdot X_2)$. 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 $E_p' = Y_{a} \cdot X_{1} \cdot X_{2} \cdot Y_{b} \cdot X_{2}$. 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 bMSC of Figure 12 cannot be transformed into an equivalent bMSC to avoid unbreakable generators.

4.5 $H_{\sigma_{\exp}}$ and woven bMSC

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

More specifically, $P_{m}$ is the set of paths of a bMSC $H_{\sigma_{\exp}}$ such that for each path $p$ of $P_{m}$:

i) $p$ starts from a node with future match equal to $M_{i}$,

ii) $p$ terminates on a node with future match equal to $M_{f}$,

iii) All other nodes of $p$ are labeled by future matches different from $M_{e}$.
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 $H_{m,p}$ obtained using Algorithm 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 $H_{m,p}$ and identified all join points in a finite set $\mathcal{P}_m$ of paths of $H_{m,p}$, 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 3 builds a HMSC $H_{R}$ 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: $M_0$ and $P.M$. Consider now HMSC $H_4$ 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 $(g_0, M_0)$, the potential match obtained in $M_0 \bullet G$ contains a single local event 'a'

![Figure 12: Impossible weaving](image)

Algorithm 5 BuildP($P, H_{m,p}$)

**input:** pointcut $P$.

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

1. $\mathcal{P}_m = \{ (q_x, M, q_y) \in T_{m,p} \mid F(q_x) = F(q_y) = M \land \exists k \in \mathbb{N}, P^k M \}$
2. Path = $\{ (q_x, M, q_y) \in T_{m,p} \mid F(q_x) = M \land F(q_y) \neq M \}$
3. while Path $\neq \emptyset$ do
4. $FuturePath = \emptyset$
5. for all $p = t_1, t_2, \ldots, t_s$ in Path do
6. for all $(q_x, M, q_y) \in T_{m,p}$ such that $q_x = \lambda_x(t)$ do
7. if $F(q_y) = M$ 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,$\Lambda$)

**input:** a HMSC $H_i = (N, T, g_0, Q_{end}, M)$, a behavioral aspect $\Lambda = (P, Ad)$.

**output:** a woven HMSC $H_i'$.

1. $H_i' = Context(P, H_i)$
2. $H_i = Future(P, H_i)$
3. compute $UC_{h_i}$
4. if $\forall c, c' \in UC_{h_i}, c$ and $c'$ are disjoint then
5. $H_{m,p} = Permutation(H_i, UC_{h_i})$
6. $\mathcal{P}_m = BuildP(P, H_{m,p})$
7. for all $p \in \mathcal{P}_m$ do
8. Compute $E_p$
9. Find the minimal factorization $x_1 \bullet P \bullet x_2 \cdot \cdot \cdot P \bullet x_k$ of $E_p$
10. $E_p' := x_1 \bullet Ad \bullet x_2 \cdot \cdot \cdot Ad \bullet x_k$
11. $H_{m,p} = H_{m,p} \{ e_p \}$
12. end for
13. Return $H_{m,p}$
14. else
15. 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 \((q_0, An)\), \(An \bullet G\) 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 HA, and aspect \(A = (P4, Ad4)\) the semantics of weaving is a set of behaviors containing scenarios of the kind \(Ad^n.A1^n, n \in \mathbb{N}\). 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 \(M_k\)) such that \(M = M1 \bullet M2\) (and \(M4\)) \(M2\). For example, in Figure 2, the bMSCs P1 and A1 are connected, whereas in Figure 13, the bMSC P4 is not connected (\(P4 = P0 \bullet Pb\) with \(P0\) a bMSC with only a local event 'a' and \(Pb\) a bMSC with only a local event 'b', and \(P0\) \(\bullet Pb\)).

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 of that 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 (IP’s introduced in [5]) and are composed with specification scenarios. The weaving process defined in [17] composed 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 states 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 et al. [12], the pointcuts have been shown to have 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 modelings 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 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 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 ⊗ 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 work. 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


