Non-local Choice and Beyond: Intricacies of MSC Choice Nodes*

Arjan J. Mooij, Nicolae Goga, and Judi M.T. Romijn

Technische Universiteit Eindhoven,
Department of Mathematics and Computer Science,
P.O. Box 513, 5600 MB Eindhoven, The Netherlands
{a.j.mooij, n.goga, j.m.t.romijn}@tue.nl

Abstract. MSC is a visual formalism for specifying the behavior of systems. To obtain implementations for individual processes, the MSC choice construction poses fundamental problems. The best-studied cause is non-local choice, which e.g. is unavoidable in systems with autonomous processes. In this paper we characterize two additional problematic classes of choice nodes. Based on these three classes we point out some errors in related work. Extending our work on pragmatic implementations of non-local choice, we motivate a different choice semantics which allows a little more behavior. Finally, inspired by practical case studies, we present the first implementation approach for non-local choice nodes that can handle arbitrary numbers of processes.

1 Introduction

Message sequence chart (MSC, see [11, 17]) is a visual formalism that is used to specify the behavior of a collection of processes. An important property of MSCs is that behaviors are described from a full-system's perspective. Then an immediate question is whether an implementation can be extracted that has the same behavior but expressed in terms of the processes in the system.

To obtain such an implementation, the behavior specified for the full system must be established by the independent processes in a distributed way. The usual way to obtain an implementation for each process is to project the MSC on each single process. However, in general, implementations with exactly the same behavior do not exist (see e.g. [13]). Typical problems that arise in naive implementation attempts are additional behaviors [19] and deadlocks [20].

Then one can conclude that the MSC formalism is inappropriate for protocol specification, but there are also some approaches to really address the problem. First, the obtained implementations can be compared with their specifications to find inconsistencies [20]. A second option is to identify and detect properties of MSCs that may cause problematic implementations, and then label such MSCs

* This research is supported by the NWO under project 016.023.015: "Improving the Quality of Protocol Standards".

M. Cerioli (Ed.): FASE 2005, LNCS 3442, pp. 273–288, 2005.

as being pathological [7]. Finally, alternative semantics of the MSC constructions are studied such that these constructions become implementable [5, 15].

In this paper we address the latter two approaches for the very topical issue of choice nodes. The best-studied property leading to implementation problems is non-local choice. In addition to this property about locality of a choice, we define two classes of problems related to propagation of the choice. These three classes together arise naturally from a single process' perspective, and we use them to point out some errors in related work.

To handle non-local choice we motivate a different kind of choice semantics, viz. one that allows a little more behavior than the standardized semantics, but in a controlled way. Based on this modified semantics we address approaches to implement choice nodes. Such approaches are highly needed, since non-local choice is inevitable in MSC specifications of systems with autonomous processes. In our cooperation with protocol standardization committees (see e.g. [15]) we have noticed that currently there are insufficient applicable solutions.

We present a generalization of our approach [15] to implement non-local choice in systems with two processes. We also introduce a new implementation approach for non-local choice that, as far as we know, is the first one that can deal with arbitrary numbers of processes. The MSC patterns required for both approaches are inspired by our experience with practical case studies.

Preliminaries. Instead of hMSCs (high-level MSCs or hierarchical MSCs), in this paper we use the mathematically more convenient notion of a message sequence graph (MSG). Since these concepts are equally expressive (see [8]) this is a valid and common strategy. An MSG is a finite directed graph in which each node is labeled with a bMSC (basic MSC), and there is one initial and one terminal node. We use the term MSC to refer to an MSG together with its bMSCs.

For simplicity reasons and without loss of generality, we assume the MSG to be normalized such that if a node has more than one outgoing edge, then the bMSC associated with the node is an empty bMSC. In this way the choices in the MSG are made explicit in (choice) nodes without an associated bMSC.

We use the following nomenclature for MSCs. There are four kinds of actions (or events): an internal action, asynchronously sending a message m (denoted by $!m$), receiving a message m (denoted by $?m$) and termination. A process is said to have initiative, if a possible next action for the process is an initiating action like an internal action, sending a message or termination. Note that finding the collection of possible next actions of a process might require considering the entire MSG.

Overview. In Section 2 we give our characterization of three properties that may cause problems when implementing MSCs with choice nodes, and we discuss some related literature. In Section 3 we discuss ways to handle the best-known class, viz. non-local choice. Section 4 contains a summary of our earlier work [15] on dealing with non-local choice in systems with only two processes, and it contains a small generalization. This summary also serves as an introduction to Section 5, in which we present an approach to implement non-local choice for

an arbitrary number of processes. Finally Section 6 gives some conclusions and directions for further work.

2 Problematic Choice Node Properties

In this section we present our characterization of three problematic choice node properties. On this basis we discuss and comment on some related literature, and in Section 5 we exploit it to isolate one of the classes.

2.1 Characterization of Choice Synchronization Problems

As mentioned before, the core implementation problem is that one collective choice is specified, while it must be implemented in a distributed way. If in a choice node all possible initiating actions can be performed by only one of the processes, this single process can simply perform the system's decision about the choice. However, in general it is not sufficient to ensure that one process makes the decision. It is also important that the processes agree on the decision, so the decision must be properly *propagated* to the other processes. So far this has not really been recognized, and the propagation issue is frequently ignored.

In the remainder of this section, we study the implementation problem for a single choice node from the perspective of a single process. An important concept will be the set of successor nodes for the process, i.e. the nodes that contain the process' possible first action after the choice node. Note that the definition of successor node for a process is not restricted to the direct successors of the choice node. Namely, if the process is not involved in some direct successors in the graph, also nodes that can be reached further on must be considered.

Non-local Choice. A first question is whether the process should initiate some behavior or it should just wait to receive a message. When several processes independently decide to initiate behavior, they might start executing different successor bMSCs. This possibility easily leads to non-specified behaviors, and it is usually called *non-local choice* (NLC). An example of non-local choice can be generated with the bMSCs in Figure 1 by constructing a choice node from which only the bMSCs msc_base and msc_NLC can be chosen.

More formally, a node is a non-local choice node for the two distinct processes p and q if the following holds: there are two different successor nodes k and l for process p and q respectively, such that p has initiative in k and q has initiative in

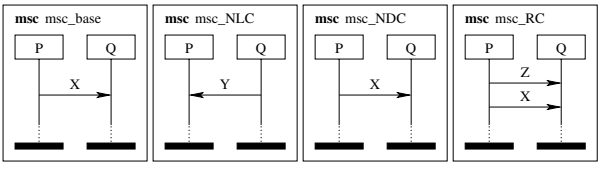


Fig. 1. bMSCs to illustrate the classification

l, and such that each (acyclic) path to node k without any action of p is disjoint with each (acyclic) path to node l without any action of q.

Non-deterministic Choice. Then assume that there is only one process that has initiative in the node, and this process performs the system's decision on the choice. Suppose in each successor node the first action of the process under consideration is a receipt, and suppose a matching message arrives. A question is whether this first receipt is sufficient to derive the decision made about the choice. In case some of the successor nodes have a common first receipt, then this is clearly not the case; we call it *non-deterministic choice* (NDC). An example of non-deterministic choice can be generated by constructing a choice node from which only the bMSCs msc_base and msc_NDC in Figure 1 can be chosen.

More formally, a node is a non-deterministic choice node for a process p if there are at least two different[1] successor nodes for p with the same receipt action as first action of p.

Race Choice. Absence of non-deterministic choice is not enough for a process to derive the choice decision on the basis of the first arriving message. Namely, in case messages *arrive* in a different order than in which their *receipt* is specified in the bMSC (which in itself is not an error, just a property of the underlying communication system), the process may incorrectly derive which decision has been made. So the first message receipt in one node, may actually have been sent according to *another* node in which the receipt is not the first action of the recipient; we call it *race choice* (RC). An example of race choice can be generated by constructing a choice node from which only the bMSCs msc_base and msc_RC in Figure 1 can be chosen.

More formally, a node is a race choice node for a process p if the following holds: there are two different[2] successor nodes k and l for process p such that p's first action in k is a receipt of message m and in l it is a receipt of a different message n, and such that starting with node l a message m may be sent to p before process p performs any action.

Examples with a combination of these properties NLC, NDC and RC can be generated with the bMSCs in Figure 1 by constructing a choice node from which only msc_base and the bMSCs for the selected properties can be chosen.

Distinguishing between the two propagation-related properties, viz. non-deterministic choice and race choice, may look somewhat arbitrary, but it is based on an essential difference. Intuitively, non-deterministic choice is a static property of the MSC, while race choice takes into account the dynamics of the communication network.

Finally it needs to be mentioned that MSCs with some of these properties are not guaranteed to give implementation problems [16]. For example, in case

[1] A successor node that can be reached from the choice node via multiple paths, is considered only once since we assume paths without actions to be irrelevant.

[2] Note that we exclude order problems that are not caused by a choice (e.g. within a bMSC), which falls under implementability of a single bMSC.

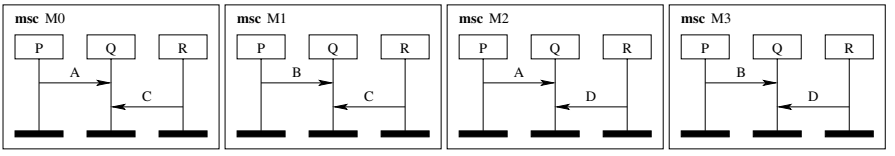


Fig. 2. Non-local choice without implied behaviors

all potential "additional" behavior has already been included in the MSC, or in case agreeing on the decision is not really important for the further execution.

2.2 Related Problems and Solutions

In this section we discuss various related issues from the literature and we point out some errors in related work.

Communication Infrastructure. Especially in small systems, the problems caused by non-local choice and race choice can be solved by extra assumptions about the underlying communication system [12, 4]. Typical properties that may help are communication synchrony, message order preservation, bounded buffer capacities and confirmed communications. In specific cases, such assumptions on the underlying system are both valid and useful.

Definitions of Non-local Choice. A frequently referenced paper for the definition of non-local choice is [2]. Although much literature suggests the equivalence of the various definitions in [2], we show that they are inconsistent. The informal introduction contains the following description:

> "When the wait-and-see strategy can be used to resolve a non-determinism within each process, we call the branching a *local branching choice*. Otherwise, when explicit synchronization between the processes is necessary to resolve a non-determinism, we call the branching a *non-local branching choice*."

After introducing a formal semantic definition and a formal syntactic characterization (equal to ours), the following informal explanation of the syntactic version is given:

> "An MSC specification has no non-local branching choice iff at each of its branching points, the first events in all bMSCs are sent by the same process."

Usually, this last version is used for definition purposes, but the first one is assumed when it comes to implementation. It is easy to see that these two definitions are different by studying a choice node with the two successor bMSCs msc_base and msc_RC from Figure 1. Since process P is the only process that can

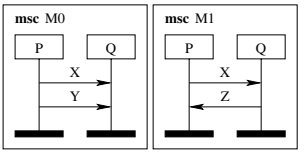


Fig. 3. Hidden non-local choice

initiate an action, it is local according to the second definition. Then according to the first definition all non-determinism should be resolved, but process Q shows the contrary.

Implied Scenarios. Implied scenarios are scenarios that are not contained in the MSC specification, but that are contained in implementations of the MSC. Although implied scenarios can result from propagation problems, only the relation with non-local choices (according to the syntactic definition of [2]) has been studied. In [18] the following two observations are made:

1. "Non-local choices are implied scenarios;"
2. "nevertheless the converse is not the case."

In contrast, [16] makes the following two observations:

3. "Notice that a non-local choice is not enough to have an implied scenario."
4. "To have an implied scenarios these conditions[3] hold: i) there is a non-local branching choice in the MSC specification so that ii) ..."

There are two contradictions here. Observation 3 falsifies observation 1, which can be shown using a choice node with as successors the bMSCs from Figure 2, where more than one process has initiative but no implied behaviors result. In turn, observation 2 falsifies observation 4, which can be shown with a choice node node with the two successor bMSCs msc_base and msc_RC from Figure 1. Implementations of this example, without non-local choice, contain implied scenarios with the prefix $!Z \cdot !X \cdot ?X$. Another example can be found in [18].

Delayed Choice. The widely accepted solution to non-deterministic choice is to use delayed choice semantics instead of ordinary choice semantics. Since this solution is effective quite often (though not always), it has become part of the MSC standard. Sometimes, it can even eliminate non-local choice by factoring out a common non-local prefix of the bMSCs after which a local choice remains.

However, we could not find any warning for its possible side-effects. Namely, delayed choice can also expose non-local choice, e.g. in a choice node with the two successor bMSCs from Figure 3. So although the MSG itself contains no non-local choice, after applying delayed choice the non-local choice pops up.

[3] This is the basis of [16]'s procedure for detecting implied behavior.

3 Dealing with Non-local Choice Nodes

In this section we address some ways to deal with the best-known choice problem from Section 2, viz. non-local choice. We motivate a class of solutions, of which only some instances have been described so far.

3.1 Traditional Approaches

Non-local choice is usually addressed by syntactically detecting (e.g. [2]) the non-local choice nodes, or by detecting the resulting implied behaviors by generating them (e.g. [16]). However, these approaches do not really address how to *solve* the problems with non-local choice. An obvious approach might be to change the MSC into a similar MSC with only local choice nodes. Since in that case at each node only one process has initiative, systems with autonomous processes cannot be specified. Another way to overcome the problems resulting from non-local choice is to explicitly include all implied behaviors in the MSC. Although this eliminates the implicit additional behaviors caused by implementing non-local choice nodes, the MSC becomes more complicated, which is definitely not desired from a practical point of view.

The problems with non-local choice nodes can also be seen as implementation issues, and hence they should not even be addressed in a specification. Then to obtain an implementation, some additional coordination protocol needs to be introduced (e.g. [2]). Although this leads to a nice layered design, it is problematic if some processes represent human beings, on which no additional protocol should be imposed. Also for protocol standardization this approach is undesired, since the additional protocol is not part of the MSC description.

3.2 Adjusted Semantics for Choice Nodes

The source of the problems with non-local choice nodes is that the underlying system is distributed. Since the processes are independent computational units, a coordination problem arises when the processes *together* need to make a transition in the MSG. Nowadays this problem is mainly noted in choice nodes, but in fact, it also arises for pure (or synchronous) sequential composition of bMSCs in an MSG. The latter issue has been solved by defining its semantics to be *weak* (or asynchronous) sequential composition, which usually corresponds to the intentions of the developer of the MSC. For choice nodes, the changes in their semantics (like delayed choice) are not (yet) sufficient.

Suppose all processes have reached a given non-local choice node. Since the processes are independent, we need to conclude that in general it cannot be avoided that the execution of several different bMSCs is initiated. This means that an *implementable* semantics of choice must allow, to some degree, parallel execution of the bMSCs. Of course, the amount of additional parallel behavior should be minimal, and as soon as possible the behavior should converge to the behavior of a conventional (or synchronous) choice.

As far as we know, this theoretical motivation for starting with parallelism and converging to synchronous choice has not been revealed before, but two of

its instances have been discovered in [5, 15]. These instances mainly differ in the way in which the additional parallel behaviors are interpreted. In [5] this behavior is ignored, while in [15] it is stored to be used at the next choice node. The main limitation of both approaches is that only systems of two processes can be addressed. In Section 5 we describe the first implementation of such a semantics for an arbitrary number of processes.

4 Approach from [15] and a Generalization

In this section we summarize the approach for implementing non-local choice nodes from [15] for two reasons. First, this approach for two processes is a nice prelude to our approach for multiple processes in Section 5. Second, we show how the MSG pattern required for [15] can be generalized.

4.1 Pattern and Its Generalization

For application of the approach of [15], the MSC must match a certain pattern both with respect to its bMSCs and to its MSG. Two special kinds of bMSCs are distinguished, viz. *RC*-like bMSCs (Request-Confirm scenario) and *A*-like bMSCs (Announce scenario). These bMSCs contain as a prefix the structure as depicted in Figure 4, in which P and Q denote the names of the two processes.

These two kinds of bMSCs can be seen as negotiation scenarios: process P can send a Request message to process Q, but Q is the arbiter process that decides whether to send a Confirm message and continue the execution of the *RC*-like bMSC or to send an Announce message and execute an *A*-like bMSC. More details are discussed together with the implementation in Section 4.2.

With respect to the MSG, the successor bMSCs of each non-local choice node M must be partitioned into *RC*-like bMSCs and *A*-like bMSCs, but such that the Request and Confirmation messages in the *RC*-like bMSCs do not occur in the *A*-like bMSCs. The main restriction of [15] is that after an *A*-like bMSC, a similar choice node as node M is reached again. This is depicted in Figure 5 (including the dashed arrows): after each *A*-like bMSC, an S node is reached that is in fact identical to node M. This MSC pattern turns out to occur frequently.

However, the MSG pattern is too strong, since it only needs to ensure that when process P has sent a Request message, it still makes sense for process Q

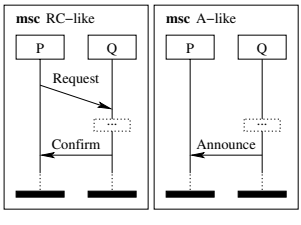


Fig. 4. Prefixes of the bMSCs for [15]

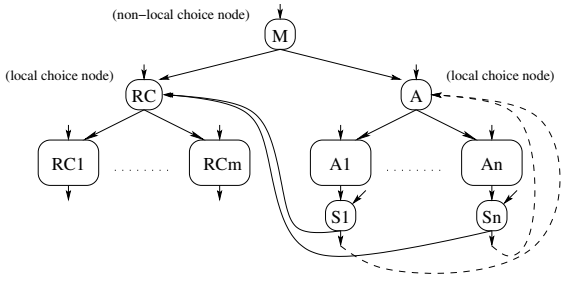


Fig. 5. MSG pattern for [15] and a generalization

to receive the message after execution of an A-like bMSC. Hence, after an A-like bMSC, only node RC is required to be reachable and we propose to *generalize* the pattern by eliminating the dashed arrows in Figure 5. After each A-like bMSC an S node is reached from which at least the previous series of RC-like bMSCs is reachable (via node RC). In addition, via the open outgoing edge also extra RC-like bMSCs and an arbitrary series of A-like bMSCs may be reachable.

4.2 Implementation

We summarize the proposed implementation for both the original pattern (see [15] for more details) and its generalization at the same time. Process Q becomes a kind of arbiter, with the usual implementation. However, the implementation for process P is slightly different: If process P receives an Announce message from process Q, it executes the corresponding A-like bMSC. Even if process P sends a Request message of an RC-like bMSC, it may still receive Announce messages from process Q that indicate that some A-like bMSC must be executed. Eventually, the Confirmation message corresponding to the Announce message will arrive, and then execution of the RC-like bMSC can be completed.

Note that in this implementation, between sending and receiving a Request message several executions of A-like bMSCs are possible. Observe also that after process P has sent a Request message, the remaining choice is in fact a local choice, viz. for arbiter process Q.

5 A New Approach to Deal with Non-local Choice Nodes

This section describes a way to implement the semantics proposed in Section 3.2 for systems with an *arbitrary* number of processes. Like in [15], the additional parallel behavior is interpreted as behavior that must be stored to be used at the next choice node. As far as we know, this is the first pragmatic implementation of non-local choice for an arbitrary number of processes. Furthermore this approach can deal with bMSCs in which several processes have initiative, and it is inspired from both examples in the literature and industrial protocol standards.

The description of our formalization is based on process algebra extended with two operators. We briefly introduce them using characteristic examples, in which a and b denote actions and s and t denote terms. The first operator is the delayed choice operator $\mp$ (see [1, 17]) with as quantifier $\overline{\Sigma}$:

$$a \cdot x \mp b \cdot y \quad = \quad \begin{cases} a \neq b : & a \cdot x + b \cdot y \\ a = b : & a \cdot (x \mp y) \end{cases}$$

The second operator is the partial synchronization operator $\cap$ (see [10, 3]) with a (hidden) set of actions S on which this operator synchronizes:

$$a \cdot s \cap b \cdot t \quad = \quad \begin{cases} a \in S \wedge b \in S \wedge a = b : & a \cdot (s \cap t) \\ a \in S \wedge b \in S \wedge a \neq b : & \delta \\ a \in S \wedge b \notin S \quad : & b \cdot (a \cdot s \cap t) \\ a \notin S \wedge b \in S \quad : & a \cdot (s \cap b \cdot t) \\ a \notin S \wedge b \notin S \quad : & a \cdot (s \cap b \cdot t) + b \cdot (a \cdot s \cap t) \end{cases}$$

In the remainder of this section, we first introduce a running example. Then we address the MSC pattern we assume, followed by a description of our proposed implementation. Finally we relate it to a proposed MSC extension.

5.1 Running Example

As an example to illustrate our approach, we use a simplified version of the well-known ATM example [20]. We have restricted it to its core non-local choice problem, as depicted in Figure 6. For later use, the MSCs contain some extra annotations; in particular the bMSCs have been split by a horizontal line.

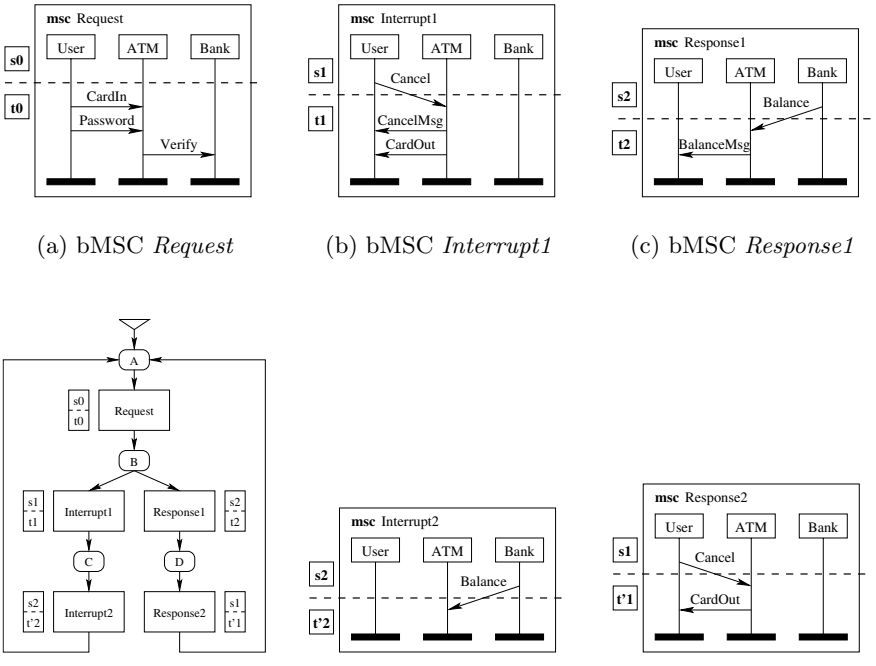
We briefly explain the functionality of this simplified ATM. In node A some repetitive behavior is started with bMSC *Request*. It consists of inserting a card and entering a password, followed by verifying the bank account. Then choice node B is reached, in which the user can choose to:

- interrupt and cancel the account verification, which corresponds to: bMSC *Interrupt1* $\rightarrow$ node C $\rightarrow$ bMSC *Interrupt2* $\rightarrow$ node A;
- wait for a balance report and press the cancel button to end the session: bMSC *Response1* $\rightarrow$ node D $\rightarrow$ bMSC *Response2* $\rightarrow$ node A.

5.2 Pattern

To keep non-local choice manageable, we isolate it from other problematic choice properties. This motivates the classification in Section 2, and from now on we ignore propagation issues. In the remainder of this section, we exploit that the MSG is normalized as discussed in Section 1 by interpreting the MSG as a graph in which the edges are labeled with (concatenated) bMSCs, and in which the nodes indicate choices.

To apply our approach, each bMSC must be split into a (preferably small) *front* part that may be executed in parallel, and the remaining *tail* part that



(a) bMSC *Request* (b) bMSC *Interrupt1* (c) bMSC *Response1*

(d) MSG (e) bMSC *Interrupt2* (f) bMSC *Response2*

Fig. 6. Simplified ATM example

will be part of a real choice. To solve non-local choice, in each node the choice between the successor bMSCs without their fronts must be a local choice. This can be achieved as follows:

1. *choose a process to become the "arbiter"*, which is typically a non-human process that occurs in each bMSC in an early stage;
2. *split the bMSCs* into a front and a tail, such that apart from the front, only the arbiter process has initiative, unless the bMSC can only be reached from a node with only one outgoing edge. (If the tail is empty for a process, also successor nodes are involved in deciding which processes have initiative.)

The splitting of the bMSCs must be such that the following conditions hold:

1. for each node and for each two of its outgoing edges e and f with different fronts, the node reached via edge e is no terminal node and the node has an outgoing edge with the same front as edge f;
2. for each two edges e and f with different fronts, the events in the front of edge e do not occur in the front of edge f;
3. for each two edges e and f, the events in the front of edge e do not occur in the tail of edge e nor in the tail of edge f.

The first condition reflects that additional front behaviors, which are the additional parallel behaviors, can indeed be used in next choice nodes. If it does not hold, a wrong arbiter process might have been chosen. But more likely the MSC lacks some unavoidable behavior, parts of which must be made explicit for our approach. Thus this condition can constructively help to improve the MSC without studying the process implementations. A last option is that the MSC needs to be slightly rearranged to fit our pattern.

The motivation for the last two conditions is quite technical and it will be discussed upon their use. Although our pattern contains some restrictions, it includes the patterns of Section 4 and it fits well-known examples as an ATM (see the running example), and a producer-consumer pair (see Section 5.4).

Let us apply this to our running example. All choices have been made explicit in the empty choice nodes A, B, C and D, and the only node with more than one outgoing edge, viz. node B, suffers from non-local choice.

To check the pattern, an arbiter process must be chosen. Using the heuristics from the first step, the *ATM* process should be an appropriate arbiter for node A. The next step is to split the bMSCs according to this arbiter. This is depicted by horizontal dashed lines in the bMSCs in Figure 6. This way of splitting turns out to fulfill the first condition, e.g. in node B after bMSC *Interrupt1* it is possible to execute the front of bMSC *Response1* namely as front of bMSC *Interrupt2*. The last two conditions also turn out to hold.

For reference purposes, we introduce names $s0$, $s1$, $s2$, $t0$, $t1$, $t2$, $t'1$ and $t'2$ for the bMSC parts as indicated in Figure 6. For example, for the *User* process:

$s0 = \epsilon$	$s1 = !Cancel$	$s2 = \epsilon$
$t0 = !CardIn \cdot !Password$	$t1 = ?CancelMsg \cdot ?CardOut$	$t2 = ?BalanceMsg$
	$t'1 = ?CardOut$	$t'2 = \epsilon$

5.3 Implementation

In general it is complicated to directly define our proposed implementation in terms of a finite state machine. It turns out to be easier to use techniques from constraint-oriented programming (see e.g. [3]), from which for concrete examples a finite state machine can be obtained using operational semantics (see e.g. [3]).

In the remainder of this section, we concentrate on only one of the processes since their implementations are independent. Furthermore we use V for the set of nodes, and E for the set of labeled edges. More specific, we represent each edge as a four-tuple $(v, m, n, w) \in E$ as follows: the edge is directed from node v to node w, and m and n are the front and the tail respectively of the corresponding bMSC projected on the process to be implemented.

Our implementation is described in Figure 7, where the smallest solution of $I.v$ denotes the implementation of node v for the process. It is defined as the synchronized execution of the terms $I_i.v.m$ for each individual front m, where $I_i.v.m$ expresses where front m may be executed in relation with all tails. The partial synchronization operator $\cap$ only synchronizes on the events in the tails, for which we exploit the last two conditions mentioned above.

$$
\begin{array}{|ll|}
\hline
I.v & = (\bigcap_{m:(\exists v,n,w:(v,m,n,w)\in E)} \quad I_i.v.m) \\
I_i.v.m & = \begin{cases} (\exists_{n,w} \quad (v,m,n,w) \in E) & : \quad I_a.v.m.\epsilon \\ (\forall_{n,w} \quad (v,m,n,w) \notin E) & : \quad (\overline{\Sigma}_{m',n',w':\,(v,m',n',w')\in E} \quad n' \cdot I_i.w'.m) \end{cases} \\
I_a.v.m.p & = \left(\overline{\Sigma}_{m',n',w':\,(v,m',n',w')\in E} \quad \begin{cases} m \neq m' & : \quad I_a.w'.m.(p \cdot n') \\ m = m' & : \quad (p \| m') \cdot n' \cdot I_i.w'.m \end{cases} \right) \\
\hline
\end{array}
$$

Fig. 7. Formalization of a single process implementation

The term $I_i.v.m$ describes the implementation in node v with respect to the *inactive* front m. If m is the front of a successor bMSC of node v, its execution can be started and hence it becomes active. Otherwise it remains inactive, and a usual choice is performed on the tails of the successor bMSCs of node v.

The term $I_a.v.m.p$ describes the implementation in node v with respect to the *active* front m. The additional parameter p is used to accumulate the series of executions of tails since front m's execution was allowed to start. In case the tail of a bMSC with front m is executed, then it is required that front m was executed along the path p to node v, which is expressed by the term $(p \| m')$.

To illustrate this approach on our ATM example, we first apply it to the high-level description in terms of s and t. Afterwards, the specific details can be substituted to obtain the final process implementations. First we give the instantiations of some of the formulas:

$$
\begin{array}{|ll|}
\hline
I.A & = I_i.A.s0 \cap I_i.A.s1 \cap I_i.A.s2 \\
\hline
I_i.A.s1 & = t0 \cdot I_i.B.s1 \\
I_i.B.s1 & = I_a.B.s1.\epsilon \\
I_a.B.s1.\epsilon & = (\epsilon \| s1) \cdot t1 \cdot I_i.C.s1 \quad \mp \quad I_a.D.s1.(\epsilon \cdot t2) \\
I_i.C.s1 & = t'2 \cdot I_i.A.s1 \\
I_a.D.s1.(\epsilon \cdot t2) & = ((\epsilon \cdot t2) \parallel s1) \cdot t'1 \cdot I_i.A.s1 \\
\hline
\end{array}
$$

After simplification we obtain the following implementation per process:

$$
\begin{array}{|l|}
\hline
I.A \quad = I_i.A.s0 \cap I_i.A.s1 \cap I_i.A.s2 \\
\hline
I_i.A.s0 \; = \; s0 \cdot t0 \;\cdot\; (t1 \cdot t'2 \;\mp\; t2 \cdot t'1) \;\cdot\; I_i.A.s0 \\
I_i.A.s1 \; = \; t0 \;\cdot\; (s1 \cdot t1 \cdot t'2 \;\mp\; (t2 \| s1) \cdot t'1) \;\cdot\; I_i.A.s1 \\
I_i.A.s2 \; = \; t0 \;\cdot\; ((t1 \| s2) \cdot t'2 \;\mp\; s2 \cdot t2 \cdot t'1) \;\cdot\; I_i.A.s2 \\
\hline
\end{array}
$$

By substituting the actions of the three processes and eliminating the $\cap$ operator, the following final implementations are obtained:

$$
\begin{array}{|ll|}
\hline
I^{User} & = \; !CardIn \cdot !Password \;\cdot\; (!Cancel \cdot (?CancelMsg + ?BalanceMsg) + \\
 & \qquad ?BalanceMsg \cdot !Cancel) \;\cdot\; ?CardOut \;\cdot\; I^{User} \\
I^{ATM} & = \; ?CardIn \cdot ?Password \cdot !Verify \;\cdot \\
 & \qquad (?Cancel \cdot !CancelMsg \cdot !CardOut \cdot ?Balance + \\
 & \qquad\quad ?Balance \cdot !BalanceMsg \cdot ?Cancel \cdot !CardOut) \;\cdot\; I^{ATM} \\
I^{Bank} & = \; ?Verify \cdot !Balance \;\cdot\; I^{Bank} \\
\hline
\end{array}
$$

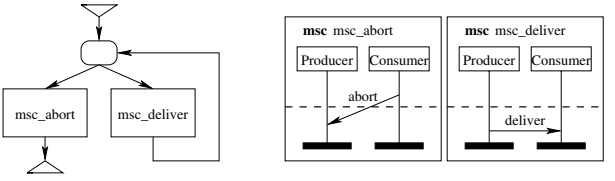


Fig. 8. Producer-consumer example

The implementation for the *ATM* (which is the arbiter process) and for the *Bank* are the usual ones. The possible behavior of the *User* has been extended, but it is intuitive in relation to Figure 6. In particular after pressing *Cancel*, the user can get a *BalanceMsg* instead of a *CancelMsg*. Using the model checker SPIN [9], we have verified that the normal implementation contains deadlocks, and that the above implementation is indeed free of deadlocks.

5.4 Relation with Compositional Message Sequence Charts

Our proposed implementation of non-local choice nodes typically contains behavior that is difficult to describe efficiently using current MSC. In [6] a syntactic extension of MSC is proposed called "compositional message sequence chart". The example in [14] to illustrate the usefulness of this extension, can also be generated using our approach and the MSC in Figure 8. Although the version of [14] gives a more precise specification (i.e. closer to an implementation), our version is simpler and more intuitive for system *specification* and still allows a (unique) implementation using the technique presented in this section.

6 Conclusions and Further Work

We have structured a number of choice node properties that may lead to implementation problems, viz. non-local choice, non-deterministic choice and race choice. This has resulted in a natural classification of these properties which covers initiative and propagation problems for choice nodes.

Further work is to address completeness of the classification. It needs to be studied whether choice nodes without any of these properties can indeed be implemented without introducing extra deadlocks or implied behaviors.

We have also focused on the best-known problematic property, viz. non-local choice, which we propose to handle by slightly changing the choice semantics. We have given the first implementation approach for non-local choice in systems with an arbitrary number of processes. This is a pure generalization of the current choice node semantics in the sense that for MSCs without choice problems it produces the normal implementation.

Further work is to study alternative formalizations of this approach. In particular the general properties of this approach need to be investigated. It would be interesting to investigate whether ignoring the additional parallel behavior

(as in [5]) can be integrated and also whether other initiative and propagation issues can be addressed.

Acknowledgements. We thank the anonymous referees for the helpful comments.

References

1. J.C.M. Baeten and S. Mauw. Delayed choice: an operator for joining Message Sequence Charts. In *Formal Description Techniques*, pages 340–354, 1995.
2. H. Ben-Abdallah and S. Leue. Syntactic detection of process divergence and non-local choice in Message Sequence Charts. In *Tools and Algorithms for the Construction and Analysis of Systems*, number 1217 in LNCS, pages 259–274, 1997.
3. H. Brinksma. Constraint-oriented specification in a constructive specification technique. In *REX Workshop on Stepwise Refinement of Distributed Systems*, volume 430 of *LNCS*, pages 130–152, 1990.
4. A.G. Engels, S. Mauw, and M.A. Reniers. A hierarchy of communication models for message sequence charts. *Science of Computer Programming*, 44:253–292, 2002.
5. M.G. Gouda and Y.T. Yu. Synthesis of communicating finite-state machines with guaranteed progress. *IEEE Transactions on Communications*, COM-32(7):779–788, July 1984.
6. E.L. Gunter, A. Muscholl, and D.A. Peled. Compositional message sequence charts. In *7th Conference on Tools and Algorithms for the Construction and Analysis of Systems*, volume 2031 of *LNCS*, pages 496–511. Springer, 2001.
7. L. Hélouët. Some pathological message sequence charts and how to detect them. In *10th SDL Forum*, number 2078 in LNCS, pages 348–364, June 2001.
8. J.G. Henriksen, M. Mukund, K. Narayan Kumar, and P. S. Thiagarajan. Towards a theory of regular MSC languages. BRICS Report RS-99-52, Department of Computer Science, Aarhus University, Denmark, 1999.
9. G.J. Holzmann. The model checker Spin. *IEEE Transactions on Software Engineering*, 23(5):279–295, May 1997.
10. International Standards Organization. *Information Processing Systems – Open Systems Interconnection – LOTOS - A Formal Description Technique Based on the Temporal Ordering of Observational Behaviour*, 1989. ISO 8807:1989.
11. ITU-T. Message sequence chart. Recommendation Z.120, ITU-T, 2000.
12. F. Khendek, G. Robert, G. Butler, and P. Grogono. Implementability of message sequence charts. In *Workshop on SDL and MSC*. SDL Forum Society, 1998.
13. M. Lohrey. Realizability of high-level message sequence charts: closing the gaps. *Theoretical Computer Science*, 309(1–3):529–554, 2003.
14. P. Madhusudan and B. Meenakshi. Beyond message sequence graphs. In *21st Conference on Foundations of Software Technology and Theoretical Computer Science*, volume 2245 of *LNCS*, pages 256–267, 2001.
15. A.J. Mooij and N. Goga. Dealing with non-local choice in IEEE 1073.2's standard for remote control. In *SAM 2004: SDL And MSC*, LNCS 3319. To appear.
16. H. Muccini. Detecting implied scenarios analyzing non-local branching choices. In *Fundamental Approaches to Software Engineering*, number 2621 in LNCS, pages 372–386. Springer Verlag, 2003.
17. M.A. Reniers. *Message Sequence Chart: Syntax and Semantics*. PhD thesis, Technische Universiteit Eindhoven, June 1999.

18. S. Uchitel. *Incremental Elaboration of Scenario-Based Specifications and Behaviour Models Using Implied Scenarios*. PhD thesis, Faculty of Engineering of the University of London, February 2003.
19. S. Uchitel, J. Kramer, and J. Magee. Detecting implied scenarios in message sequence chart specifications. In *Proceedings of the 8th European software engineering conference*, pages 74–82. ACM Press, 2001.
20. S. Uchitel, J. Kramer, and J. Magee. Synthesis of behavioral models from scenarios. *IEEE Transactions on Software Engineering*, 29(2):99–115, February 2003.