Realizability of Collaboration-based Service Specifications

Humberto Nicolás Castejón, Rolv Bræk
Department of Telematics
Norwegian Univ. of Sci. and Tech., Norway
{humberto.castejon, rolv.braek}@item.ntnu.no

Gregor von Bochmann
School of Inf. Technology and Engineering
University of Ottawa, Ottawa, Canada
bochmann@site.uottawa.ca

Abstract

This paper is concerned with compositional specification of services using UML 2 collaborations, activity and interaction diagrams. It addresses the problem of realizability: given a global specification, can we construct a set of communicating state machines whose joint behavior is precisely the specified one? We approach the problem by looking at how collaboration behaviors may be composed using UML activity diagrams. We classify realizability problems from the point of view of each composition operator, and discuss their nature and possible solutions. This brings a new look at already known problems: we show that given some conditions, some problems can already be detected at an abstract collaboration level, without needing to look into detailed interactions.

1. Introduction

Service engineering is a challenging task. In many cases, service behavior is not performed by a single component, but by several distributed collaborating components. This is referred to as the *crosscutting* nature of services. By structuring according to components, the behavior of each of them can be defined precisely and completely, while the behavior of a service is fragmented. In order to model the global behavior of a service more explicitly one needs an orthogonal view where the collaborative behavior is in focus. Interaction sequences such as MSCs and UML Sequence diagrams [16] are commonly used for this purpose. Normally when using interaction sequences it is very cumbersome to define all the intended scenarios. In addition, there are problems related to the realizability of interaction scenarios, i.e. finding a set of local component behaviors whose joint execution leads precisely to the global behavior specified in the scenarios. The realizability of MSC-based specifications has been extensively studied by different authors (e.g. [1, 17]). Conditions for realizability have been proposed for HMSCs [11] and Compositional MSCs [14], as well as restricted classes of HMSCs that are known to be always realizable [8]. Some authors have studied pathologies in HMSCs [4, 10] that prevent their realization. Other authors have considered realizability notions that allow additional message contents [3, 8].

A promising step forward is to adopt a collaboration-oriented approach, where the main structuring units are collaborations. In [6] we have shown the suitability of UML 2 collaborations [16] for the specification of services. Being both structural and behavioral classifiers in UML 2, collaborations can be used to define a service as a structure of roles with associated interaction behavior. Moreover, collaborations can be decomposed into smaller sub-collaborations by means of collaboration-uses (see Fig. 1(e)). *Elementary collaborations* (i.e. collaborations that are not further decomposed into sub-collaborations) are often reusable and simple enough to be completely specified using interaction sequences. The overall behavior of a composite collaboration can then be specified as a "choreography" of its sub-collaborations (i.e. a description of the execution order or causality between the sub-collaborations). For this we use UML Activity diagrams. While HMSCs describe collections of scenarios, and therefore represent incomplete and existential behavior, our choreographies describe the exact behavior of a service, according to the designer's intentions.

Interestingly, the choreography of sub-collaborations enables us to understand and classify the underlying reasons leading to realization problems. We say that a choreography is directly realizable if the joint execution of the local behaviors of all components – obtained in a straightforward manner by applying the composition ordering defined by the choreography to the local component behaviors of the sub-collaborations – leads precisely to the global behavior specified by the choreography. Note that some choreographies that are not di-

rectly realizable may still be realized by adding extra coordination messages or additional data in messages. We consider these measures as solutions to realization problems, which could be adopted by the designer depending on the application context and service domain. Note also that the realizability of a choreography depends not only on the ordering defined by the activity diagram of the choreography, but also on the characteristics of the underlying communication service used for the transmission of messages. The communication service is characterized by the type of transmission channels, and the type and number of input buffers of each component. We assume there is no message loss, and distinguish between *asynchronous* channels with *out-of-order delivery* (i.e. order of transmitted messages may not be preserved) and channels with *in-order delivery*. Components may have either a single input FIFO buffer (i.e. one buffer for all received messages) or separate input FIFO buffers (i.e. one buffer for messages received from each different peer).

In the following sections we study the direct realizability of a choreography from the point of view of the operators used to compose the sub-collaborations. In our discussion we assume that each sub-collaboration of a choreography is directly realizable. Then, for each composition operator (i.e. sequential, alternative, parallel, interruption) we study the problems that may lead to difficulties of realization. We investigate the actual nature of these problems and discuss possible solutions to prevent or remedy them.

2. Sequential Composition

Sequential composition imposes a causal dependency or partial order between the events of the composed sub-collaborations. In the following the notions of strong and weak sequential composition are discussed.

Strong Sequencing. Strong sequencing between two collaborations C_1 and C_2, written $C_1 \circ_s C_2$, requires C_1 to be completely finished, for all its components, before C_2 can be initiated. It requires a direct precedence relation between the terminating action(s) of C_1 and the initiating action(s) of C_2, so that the latter can only happen after the former are finished. This leads to the following:

Proposition 1. *The strong sequential composition of two directly realizable collaborations C_1 and C_2, $C_1 \circ_s C_2$, is directly realizable if all terminating actions of C_1 and all initiating actions of C_2 are located at the same component.*

The above proposition requires C_1 to terminate at the component initiating C_2. This is the only way the initiator of C_2 can know when C_1 is completely finished. If this condition is not satisfied, coordination messages must be added from C_1's terminating components to C_2's initiating components, in order to guarantee the strong sequencing. This could be done automatically by a synthesis algorithm [18].

Weak Sequencing. Weak sequencing of two sub-collaborations C_1 and C_2, written $C_1 \circ_w C_2$, does not require C_1 to be completely finished before C_2 can be initiated. Any component can start participating in C_2 *as soon as it has finished with* C_1 (without waiting for the other components to finish as well). This means that the actions in the two collaborations are sequenced on a per-component basis. This is the sequential composition semantics used in HMSCs and UML Interaction Overview Diagrams, but not in UML activity diagrams. We therefore mark edges with a stereotype *{weak}* whenever we want them to represent weak sequencing (see Fig. 1). For the sake of illustration, we depict activities simply as collaboration-uses. For each collaboration-use we indicate the *initiating role* (i.e. the role performing the first action) by a dot and the *terminating role* (i.e. the role performing the last action) by a bar.

Weak sequencing introduces a certain degree of concurrency, since the executions of the composed collaborations may partially overlap. Although such concurrency may be desirable for performance or timing reasons, it comes at a price, since it may lead to specifications that are not directly realizable and even counter-intuitive. This is the case for the specification in Fig. 1(a). According to the weak sequence semantics, component B may initiate collaboration C_3 as soon as it has finished with C_1. As a result, collaborations C_2 and C_3 may be executed in any order in the realized system. This is counter-intuitive to the specification, which we assume reflects the designer's intention (i.e. that C_3 should be executed after C_2, with some allowed overlapping). If the designer's intention was that the collaborations be concurrently executed, this should rather be explicitly specified by means of parallel composition.

To avoid the aforementioned problem, when two collaborations are composed in weak sequence the component initiating the second collaboration should participate in the first collaboration (e.g. as in the composition of C_1 and C_2 in Fig. 1(a)). We say a sequential composition with this property is weakly-causal:

Definition (**weak-causality**). The weak sequential composition of two collaborations, $C_1 \circ_w C_2$, is *weakly-causal* if the initiator of C_2 participates in C_1.

Weak-causality is a necessary condition for direct realizability of weak sequential composition. However, it is

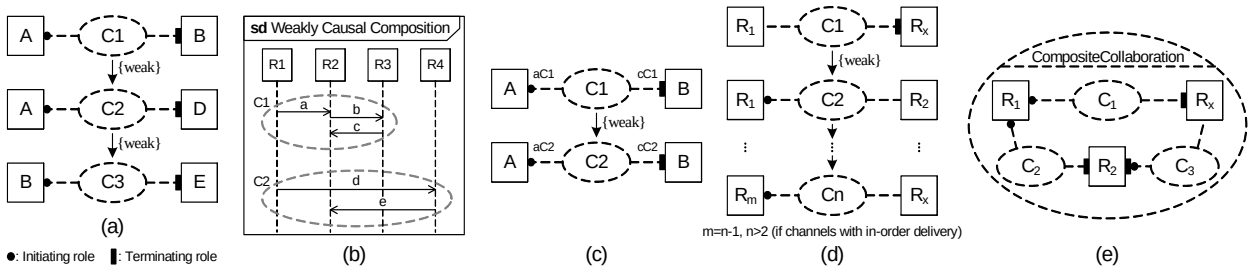

Figure 1. Problematic weak sequential compositions

not strong enough to be a sufficient condition. For example, consider the weak sequential composition of C_1 and C_2 in Fig. 1(b). This composition is weakly-causal, but it is not directly realizable. Component *R1* may initiate collaboration C_2 just after sending message *a* in C_1. Therefore, the actions in C_1 that follow the sending of message *a* may overlap with those performed in C_2 by the same components. For example, message *e* may be received at *R2* before message *c*, or even before message *a*. Obviously, this message reception order has not been explicitly specified. We note that weak-causality is enforced in the so-called local-HMSCs of [8].

In the literature about MSCs, the possibility that messages may be received in a different order from the one specified is usually called a **race condition** [2]. In general, race conditions can occur when a receiving event is specified to happen before another event (i.e. either receiving or sending), and both events are located on the same component. The reason lies in the controllability of events. While a component can always control when its sending events should happen (e.g. it can wait for one or more messages to be received before sending a message), it cannot control the timing of its receiving events. The occurrence of races highly depends on the underlying communication service that is used. If no assumption is made about the communication service, races can only be prevented if all message transmissions are strongly sequenced. This condition might be quite restrictive. We now present a less restrictive condition that does not prevent all races, but reduces their number and facilitates their detection, compared with weak-causality. This condition, which we call *send-causality*, requires all sending events to be ordered, except those that have been explicitly specified (with parallel composition) to happen concurrently.

Definition (send-causal composition). $C_1 \circ_w C_2$ is *send-causal* if (1) C_1 and C_2 are send-causal (see definition below), and (2) the component initiating C_2 is the one that performs either the last sending event of C_1 or the receiving event corresponding to that sending event[1].

Definition (send-causal elementary collaboration). An elementary collaboration is *send-causal* if it can be decomposed into a choreography of sub-collaborations, each of them consisting of exactly one message, where all sequential compositions in the choreography are send-causal.

It can be shown (see [5]) that when send-causality is enforced, races may only occur between two or more consecutive receiving events (i.e. not between a sending event and a receiving event).

Proposition 2. *In a send-causal composition, race conditions may only exist between two or more consecutive receiving events.*

Corollary 1. *A send-causal composition is directly realizable over a communication service with in-order delivery and separate input buffers.*

One of our motivations is to provide guidelines for constructing specifications with as few conflicts as possible and whose intuitive interpretation corresponds to the behavior allowed by the underlying semantics. We therefore propose, as a general specification guideline, that all elementary collaborations be send-causal. Weak sequencing of collaborations should also be send-causal, unless there is a good reason to relax this requirement. In the following we assume that all elementary collaborations are send-causal.

A *potential* race condition exists between two weakly sequenced collaborations, $C_1 \circ_w C_2$, if there is a component that participates in both collaborations and plays roles whose executions may partially overlap. Due to Proposition 2, if the sequencing is send-causal this may only happen when the role that the component plays in C_1 ends with a message reception (i.e. it is a terminating

[1] For the sake of simplicity, we assume here that each sub-collaboration has only a single initiating event and a single last sending event, but the definition could be easily generalized to consider multiple ones.

role) and the role it plays in C_2 starts with another message reception (i.e. it is a non-initiating role). Whether a potential race condition is an *actual* race or not depends on the underlying communication service, and on whether messages are received from the same or from different components. For example, in Fig. 1(c) a potential race condition exists at component B between the receptions of the last message in C_1 and the first message in C_2, but it is only actual in the case of out-of-order delivery.

We note that race conditions may not only appear between *directly* composed collaborations (e.g. Fig. 1(c)), but also between *indirectly* composed ones, as shown in Fig. 1(d). In this specification it is the weak sequencing between C_1 and C_2 that makes the potential race between C_1 and C_n possible. We therefore say that there is **indirect weak sequencing** between C_1 and C_n. This "propagation" of weak sequencing makes it more difficult to avoid races.

We have the following result:

Proposition 3. *The* send-causal *weak sequential composition of a set of directly-realizable collaborations is directly realizable*

- *over a communication service with* in-order delivery *if the following condition is satisfied: if a component plays a terminating role in a collaboration C_1 followed by a non-initiating role in another collaboration C_n, then the last message it receives in C_1 and the first one it receives in C_n are sent by the same peer-components; or*
- *over a communication service with* out-of-order delivery *only if no component plays a terminating role followed by a non-initiating role.*

Working with binary collaborations we can easily know which component sends the first and last messages of a collaboration, if we know which components play the initiating and terminating roles. Due to Proposition 3, actual races can then be detected at an early specification stage, when the detailed behavior of each collaboration has not yet been specified, but only the selection of their initiating and terminating roles has been done. In the case of n-ary collaborations, we can perform the same early analysis, but only potential races can be discovered.

One interesting aspect of the specification with collaborations is that we can get information about potential races from the diagram describing the structural composition of collaborations (see e.g. Fig. 1(e)). In such diagram we can see whether a component participates in several collaborations, and whether it plays at least one terminating and one non-initiating role in them. If that is the case, a potential race exists. This information could then be used to direct the analysis of the behavioral specification (i.e. the choreography).

Resolution of Race Conditions. Race conditions can be resolved in several ways. Some authors [13, 7] have proposed mechanisms to automatically eliminate race conditions by means of synchronization messages. We note that when the send-causality property is satisfied, the synchronization message should be used to transform the weak sequencing leading to the race into strong sequencing. If synchronization messages are added in other places new races may be introduced.

Other authors (e.g. [12, 14]) tackle the resolution of race conditions at the design and implementation levels. They differentiate between the reception and consumption of messages. This distinction allows messages to be consumed in an order determined by the receiving component, independently of their arrival order. We call this *message reordering for consumption*. In general, this reordering may be implemented by first keeping all received messages in a (unordered) pool of messages. When the behavior of the component expects the reception of one or a set of alternative messages, it waits until one of these messages is available in the message pool. Khendek et al. [12] use the SDL Save construct to specify such message reordering. This technique can be used to resolve races between messages received from the same source (i.e. in the case of channels with out-of-order delivery), as well as races between messages received from different sources. In the latter case, a communication service with separate input buffers would also resolve the races. Finally, races may also be resolved if an explicit consumption of messages in all possible orders is implemented (i.e. similar to co-regions in MSCs).

We believe that the resolution of races heavily depends on the specific application domain and requirements, as well as on the context in which they happen. In some cases the addition of synchronization messages is not an option, and a race has to be resolved by reordering for consumption. In other cases, such as when races lead to race propagation problems (see Section 3) a strict order between receptions is required, so components should be synchronized by extra messages. At any rate, all race conditions should be brought to the attention of the designer once discovered. She could then decide, first, whether the detected race entails a real problem (e.g. in Fig. 1(d) there is no race if all channels have the same latency). Then, she could decide whether reordering for consumption is acceptable or synchronization messages need to be added or the specification has to be revised.

Loops. Loops can be used to describe the repeated execution of a (composite) collaboration, which we call the body collaboration. A loop can therefore be seen as a shortcut for strong or weak sequential composition of several executions of the same body collaboration. This means that the rules for strong/weak sequencing must be applied. We note that all executions of a loop involve the same set of components (the weak-causality property is thus always satisfied). This fact makes the chances for races high when weak sequencing is used. Strong sequencing should therefore be preferred for loop bodies in the general case.

Loops may give rise to so-called *process divergence* [4], characterized by a component sending an unbounded number of messages ahead of the receiving component. This may happen if the communication between any two of the participants in the body collaboration is unidirectional.

As we will see in the next section, loops may also affect the realizability of choices.

3. Alternative Composition

Alternative composition is specified by means of choice operators, and describes alternatives between different execution paths. In a choice one or more *choosing* components decide the alternative of the choice to be executed, based on the (implicit or explicit) conditions associated with the alternatives. The other *non-choosing* components involved in the choice follow the decision made by the choosing components (i.e. execute the alternative chosen by the latter). It is thus important that:

1. The choosing components, if several, agree on the alternative to be executed. We call this the **decision-making process**.
2. The decision of the choosing components is correctly propagated to the non-choosing components. We call this the **choice-propagation process**.

In the following we study how each of these aspects affect the realizability of a choice. We assume that the set of choosing components is the union of the initiating components of all the choice alternatives.

3.1. Decision-making Process

The intuitive interpretation of a choice is that only one of the alternative behaviors is to be eventually executed. Deciding which alternative to be executed becomes simple if there is only one choosing component, and the conditions for the alternatives are local to that component (i.e. they are expressed in terms of locally observable predicates). Choices with this property are called **local**. It is easy to see that local choices are realizable (up to the decision-making process), since the decision is made by a single component based only on its local knowledge.

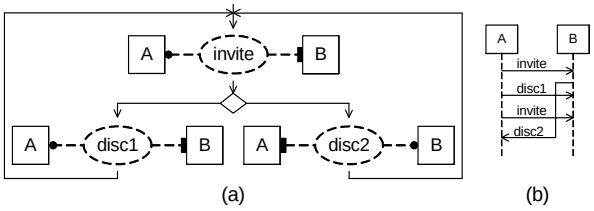

Figure 2. (a) Non-local choice and (b) implied behavior

The decision-making process gets complicated when there is more than one choosing component. This is the case in the choice of Fig. 2(a), where there are two choosing components, namely *A* and *B*. From a global perspective, we may think that once the decision node is reached, either component A initiates collaboration *disc1* with *B*, or component *B* initiates collaboration *disc2* with *A*. We are assuming then that there is an implicitly synchronization between *A* and *B*, which allows them to agree on the alternative to be executed. However, in a directly realized system, components *A* and *B* will not be able to synchronize and they may decide to initiate both collaborations simultaneously.

Choices involving more than one choosing component are usually called **non-local choices** [4]. They are normally considered as pathologies that can lead to misunderstanding and unspecified behaviors, and algorithms have been proposed to detect them in the context of HMSCs (e.g. [4, 10]). Despite the extensive attention they have received, there is no consensus on how they should be treated. We believe this is due to a lack of understanding of their nature. Some authors (e.g. [4]) consider them as the result of an underspecification and suggest their elimination. This is done by introducing explicit coordination, as a refinement step towards the design. Other authors look at non-local choices as an obstacle for realizability and propose a restricted version of HMSCs, called *local HMSCs* [11, 8], that are always realizable. These HMSCs forbid non-local choices. Finally, there are authors [9, 15] that consider non-local choices to be almost inevitable in the specification of distributed systems with autonomous processes. They propose to address them at the implementation level, and propose a generic implementation approach for non-local choices.

The problem with non-local choices is the existence of several *uncoordinated* components that have the pos-

sibility to make an independent decision in the directly realized system. As a solution, we may think of making the choice local by coordinating these components (i.e. either with additional messages or with additional message contents), so that they make a common decision. Such coordination may however not be feasible in all contexts and application domains. Consider, for example, the specification of a communication service where both end-users can take the initiative to disconnect. This could be specified as a non-local choice between two disconnection collaborations, each of them initiated by a different component (see Fig. 2(a)). The decision made by any of the components to initiate one of the disconnection collaborations is not totally controlled by that component, but it is triggered by the respective end-user. It therefore makes little sense to coordinate the components in order to obtain a local choice, since this would imply the coordination of the end-users' initiatives. Such non-local choice is simply unavoidable.

We refer to non-local choices where the coordination of the choosing components is not feasible as **competing-initiatives choices**. A characteristic of them is that all the alternative collaborations become simultaneously enabled, and they are triggered by events that cannot be controlled by the initiating components, such as an end-user initiative or a time-out. As a result, the alternative collaborations cannot be prevented from being simultaneously triggered. If this happens, it should be detected as soon as possible and the choice correctly resolved by means of a proper conflict resolution. Any component involved in two or more alternatives may be potentially used to detect the initiative conflict and initiate the resolution. For such components, the competing initiatives reveal themselves in the components' role sequences as choices between an initiating and a non-initiating role, or between two non-initiating roles played in collaborations with different peers.

A side effect of competing-initiatives choices is the existence of **orphan** messages. Consider again the specification in Fig. 2(a), which describes the repetitive execution of collaboration *invite* followed by either *disc1* or *disc2*. Now imagine that each collaboration consists of only one message. The scenario in Fig. 2(b) is then possible, where message *disc2* is sent as a response to the first *invite* message, but it is received by *A* after having sent the second *invite*. Component *A* may then consume *disc2* as a response to the second *invite* message, leading to an undesired behavior. What happens is that the conversation (i.e. exchange of messages) including *disc2* is finished while this message is still in the system (i.e. not consumed). Message *disc2* thus becomes orphan, with the danger of being consumed in a later occurrence of the same conversation. This can be avoided by marking messages (e.g. with a session id), so they are only consumed within the right collaboration instance.

Competing-initiatives choices correspond to the non-local choices discussed by Gouda et al. [9] and Mooij et al. [15]. These authors propose some resolution approaches. In the domain of communication protocols, Gouda et al. [9] propose a resolution approach for two competing alternatives (i.e. two choosing components), which gives different priorities to the alternatives. Once a conflict is detected, the alternative with lowest priority is abandoned. With motivation from a different domain, where Gouda's approach is not satisfactory, Mooij et al. [15] propose a resolution technique that executes the alternatives in sequential order (according to their priorities), and is valid for more than two choosing components. We conclude that the resolution approach to be implemented depends on the specific application domain. We therefore envision a catalog of domain specific resolution patterns from which a designer may choose the one that better fits the necessities of her system. We note that any potential resolution should also address the problem of orphan messages, which is not considered in either [9] or [15].

3.2. Choice-propagation Process

The fact that a choice is local does not guarantee its realizability. The decision made by the choosing component must be properly propagated to the non-choosing components, in order for them to execute the right alternative. In each alternative, the behavior of a non-choosing component begins with the reception of a sequence of messages, which we call the *triggering trace*. Thereafter, the component may send and receive other messages. It is the triggering traces that enable a non-choosing component to determine the alternative chosen by the choosing component. In some cases, however, a non-choosing component may not be able to determine the decision made by the choosing component. As an example, we consider the local choice in Fig. 3(a). For the component *R3*, the triggering traces for both alternatives are the same (i.e. the reception of message *x*). Therefore, upon reception of *x*, *R3* cannot determine whether *R1* decided to execute collaboration C_1 or C_2. That is, *R1*'s decision is ambiguously propagated to *R3*. We say a choice has an **ambiguous propagation** if there is a non-choosing component for which the triggering traces *specified* in two alternatives have a common prefix. Note that according to this definition, triggering traces such as (?x,?y) and (?x,?z) cause ambiguous propagation. This is true in any direct realization, since the choice cannot be made immediately after *?x*. An easy solution in this case would be to delay the choice (i.e.

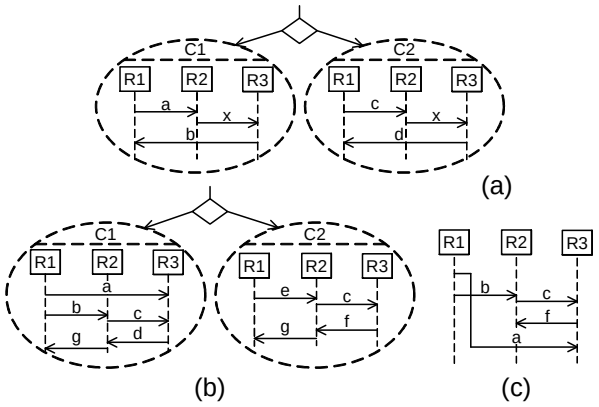

Figure 3. (a) Non-deterministic and (b) Race choice propagation; (c) Behavior implied by (b)

extract *?x* from the choice). Choices with ambiguous propagation are not directly realizable. They are similar to the non-deterministic choices defined in [14].

Now consider the choice in Fig. 3(b). It is a local choice and, according to the triggering traces specified for any of the two non-choosing components, the propagation should not be ambiguous. Still, this choice is not directly realizable. A race condition between messages *a* and *c* in C_1 may lead to the scenario of Fig. 3(c), where *R1* and *R2* execute C_1, while *R3* executes C_2. This example shows that in the presence of race conditions the triggering trace *observed* by a non-choosing component may differ from the specified one. Therefore, whenever race conditions may appear in any of the alternatives, we need to consider the potentially observable triggering traces in the analysis of choice propagation (e.g. (?a, ?c) and (?c, ?a) for *R3* in collaboration C_1 – Fig. 3(b)). We say a choice has a **race propagation** if there is ambiguous propagation due to races. Choices with race propagation are not directly realizable. They are similar to the race choices defined in [14].

To resolve the problem of race propagation we need to eliminate the race(s) that lead to it. However, if we try to remove the race conditions by means of message reordering for consumption (e.g. by means of separate input buffers), the race propagation problem may still persist. This is because, in general, a component would not be able to determine whether a received message should be immediately consumed as part of one alternative, or be kept for later consumption in another alternative (e.g. race propagation in Fig. 3(b) cannot be solved with separate input buffers). To make the message reordering work, we need to mark the messages with the collaboration *instance*[2] they belong to [18]. This not only avoids race propagation, but also ambiguous propagation in general. In [8] the realizability of local-HMSCs is studied. Although choice propagation is not explicitly discussed, the authors propose marking all messages (i.e. not only those involved in a race propagation) as we have just explained. Components have thus to check the data carried by *all* messages, and decide whether to consume them or not. We believe this unnecessarily increases the amount of processing needed upon message reception. We prefer to detect the cases of race propagation and remove the underlying race condition(s) either by transforming the responsible weak sequencing into strong sequencing, or by marking only the involved messages and applying message reordering to them.

Unfortunately, neither ambiguous nor race choice propagation can be detected at the collaboration level[3], we need to consider the detailed behavior of the subcollaborations involved in the choice.

Choices without ambiguous or race propagation are said to have **proper decision propagation**. These choices are directly realizable.

4. Interruption

The interruption semantics requires a collaboration *C* be interrupted once another preempting collaboration C_{int} is initiated. In a distributed asynchronous system the interruption may take some time to propagate to all participants in the interrupted collaboration. This means that certain components may still proceed executing their behavior in *C* for some time after C_{int} has been initiated.

As competing-initiatives choices, interruption compositions suffer from a problem of initiatives (from the interrupted and the interrupting collaborations) that compete with each other. They are therefore not directly realizable, in the general case. However, with interruptions the existence of competing initiatives is visible. The detection is thus easy at the choreography level. We refer to Section 3.1 for a discussion on resolution of competing initiatives situations and related problems.

5. Parallel Composition

A parallel composition is directly realizable as long as the composed collaborations are completely independent (i.e. their executions do not interfere with each other). Unfortunately, sometimes there are implicit dependencies that may lead to unspecified behaviors. This is the case if a component participates in several concurrent collaborations that use the same message types.

[2] If the choice is part of the body of a loop, an iteration number should be considered.

[3] We may detect the existence of a race at the collaboration level, but could not determine if that race affects the propagation.

Messages belonging to one collaboration may then be consumed within a different collaboration.

Implicit dependencies may also exist through shared resources. In this case, appropriate coordination has to be added between the collaborations, which will normally be service-specific. In [6] we discussed the automatic detection of interactions, due to shared resources, between concurrent instances of the same composite service collaboration. This detection approach makes use of pre- and post-conditions associated with sub-collaborations, and could also be used to detect interactions between collaborations composed in parallel with forks.

6. Conclusions

We have studied the realizability of service specifications given as *choreographies* of elementary sub-collaborations. A choreography describes the execution order of service sub-collaborations by means of an activity diagram.

The realizability problem has already been discussed for MSC-based specifications, where a number of specification pathologies have been identified and some "generic" resolutions proposed. We have studied such pathologies, and their solutions, from the point of view of the composition operators used in a choreography: weak and strong sequence, alternative, interruption and parallel.

The result of our study is a better understanding of the actual nature of realizability problems. Not surprisingly, we have seen that implicit concurrency and competing initiatives are at the heart of most problems. The send-causality property identified in this paper helps to build specifications that are more intuitive and less prone to conflicts, since it forces concurrency to be explicitly specified (i.e. by means of parallel composition or interruption). We have shown that some problems can already be detected at an abstract collaboration level, without needing to look into detailed interactions. We have also shown that generic solutions to the discussed problems are not valid. The same type of problem may require different resolutions in different contexts.

In [5] we present a set of algorithms for the detection of the problems discussed in this paper. We are currently working on their implementation. As future work we plan to investigate interaction patterns and domain-specific solutions to the problems we have discussed.

References

[1] R. Alur, K. Etessami, and M. Yannakakis. Realizability and verification of MSC graphs. *Theor. Comput. Sci.*, 331(1):97–114, 2005.

[2] R. Alur, G. J. Holzmann, and D. Peled. An analyzer for message sequence charts. *Software - Concepts and Tools*, 17(2):70–77, 1996.

[3] N. Baudru and R. Morin. Safe implementability of regular message sequence chart specifications. In *ACIS 4th Intl. Conf. on Soft. Eng., Artificial Intelligence, Networking and Parallel/Distr. Comp. (SNPD'03)*, 2003.

[4] H. Ben-Abdallah and S. Leue. Syntactic detection of process divergence and non-local choice in message sequence charts. In *2nd Int. WS on Tools and Algs. for the Construction and Analysis of Sys. (TACAS'97)*, 1997.

[5] H. N. Castejón, G. Bochmann, and R. Bræk. Investigating the realizability of collaboration-based service specifications. Technical report, Avantel 3/2007 ISSN 1503-4097, NTNU, 2007.

[6] H. N. Castejón and R. Bræk. Formalizing collaboration goal sequences for service choreography. In *26th Intl. Conf. on Formal Methods for Networked and Distr. Sys. (FORTE'06)*, volume 4229 of *LNCS*. Springer, 2006.

[7] C.-A. Chen, S. Kalvala, and J. Sinclair. Race conditions in message sequence charts. In *3rd Asian Symposium on Programming Languages and Systems (APLAS'05)*, volume 3780 of *LNCS*. Springer, 2005.

[8] B. Genest, A. Muscholl, H. Seidl, and M. Zeitoun. Infinite-state high-level mscs: Model-checking and realizability. *J. Comput. Syst. Sci.*, 72(4):617–647, 2006.

[9] M. G. Gouda and Y.-T. Yu. Synthesis of communicating finite state machines with guaranteed progress. *IEEE Trans. on Commun.*, Com-32(7):779–788, July 1984.

[10] L. Hélouët. Some pathological message sequence charts, and how to detect them. In *10th Intl. SDL Forum*, volume 2078 of *LNCS*. Springer, 2001.

[11] L. Hélouët and C. Jard. Conditions for synthesis of communicating automata from HMSCs. In *5th Intl. Workshop on Formal Methods for Industrial Critical Systems (FMICS'00)*. GMD FOKUS, 2000.

[12] F. Khendek and X. J. Zhang. From MSC to SDL: Overview and an application to the autonomous shuttle transport system. In *Dagstuhl Workshop on Scenarios: Models, Transformations and Tools*, 2003.

[13] B. Mitchell. Resolving race conditions in asynchronous partial order scenarios. *IEEE Trans. Softw. Eng.*, 31(9):767–784, 2005.

[14] A. Mooij, J. Romijn, and W. Wesselink. Realizability criteria for compositional MSC. In *11th Intl. Conf. on Algebraic Methodology and Software Technology (AMAST'06)*, volume 4019 of *LNCS*. Springer, 2006.

[15] A. J. Mooij, N. Goga, and J. Romijn. Non-local choice and beyond: Intricacies of MSC choice nodes. In *Fundamental Approaches to Soft. Eng. (FASE'05)*, 2005.

[16] OMG. *UML 2.1.1 Superstructure Spec.*, February 2007.

[17] S. Uchitel, J. Kramer, and J. Magee. Incremental elaboration of scenario-based specifications and behavior models using implied scenarios. *ACM Trans. Softw. Eng. Methodol.*, 13(1):37–85, 2004.

[18] G. von Bochmann and R. Gotzhein. Deriving protocol specifications from service specifications. In *ACM SIGCOMM Symposium*, 1986.