Realizability of Service Specifications

Mohammad F. Al-hammouri(✉) and Gregor von Bochmann

School of Electrical Engineering and Computer Science (EECS),
University of Ottawa, Ottawa, ON, Canada
{m.alhammouri,bochmann}@uottawa.ca

Abstract. This paper considers a global requirements model in the form of partially ordered actions of UML collaborations, or a high-level MSC (UML interaction sequences), and then studies the derivation of a distributed design model which may include coordination messages exchanged between the different system components. Different problems for the direct realization (without coordination messages) of a design model for special cases of alternatives followed by strict or weak sequence are discussed and solutions provided. Then the case of a weak while loop is considered. While previous work proposes the addition of sequence numbers in the involved messages, we show that in most cases such sequence numbers are not required. We consider message FIFO transmission or without order, and identify two potential problems: loop termination race, and message overtaking. A proposition is given which states under which conditions the directly realized distributed design model does not have these problems and therefore does not need additional sequence numbers. Another proposition provides certain modifications (including the addition of sequence numbers) that can be applied to the design model when these problems are present, and such that the resulting design model conforms to the requirements. These results can be viewed as an improvement of the previous work in [1] by minimizing the number of additional sequence numbers that must be included in the messages of a weak while loop collaboration.

Keywords: Direct realizability
Deriving a distributed implementation · Distributed applications
Partial order specifications · Distributed design models
Weak sequencing

1 Introduction

In this paper, we are concerned of the transformation from a global requirements model, which describe the behavior of a distributed system in an abstract manner by defining the local actions to be performed by the different system components, to a distributed design model, which defines the behavior of each system component separately, including its local actions plus the exchange of coordination messages which are necessary to assure that the actions of the different components are performed in the required order. This problem is often

F. Khendek and R. Gotzhein (Eds.): SAM 2018, LNCS 11150, pp. 127–143, 2018.
https://doi.org/10.1007/978-3-030-01042-3_8

called realizability of service specifications where the service specification is the global requirements model and the specification is said to be directly realizable if a design model can be constructed without any coordination messages. Many difficulties are associated with the realizability of service specifications like non-local choice [2], non-deterministic choice [3], and race conditions [4]. Most of the work in this area uses Message Sequence Charts (MSC) [5] or UML interaction sequences [6] as a modeling paradigm for the global requirements model. We have proposed to use the concept of collaborations for defining the requirements model [6–9]. A collaboration identifies a certain number of system components that play certain roles in the collaboration and defines a global behavior to be performed by these roles. The behavior is defined in terms of actions to be performed by the roles, and a partial order that defines constraints on the order in which these actions may be performed. Normally, the behavior of a collaboration is defined in terms of sub-collaborations that are performed in a specified order. The ordering relationships are strict or weak sequential order, alternatives, concurrency and looping behavior. This formalism is similar to HMSC [10] and UML [11].

In [1], an algorithm is proposed to derive a distributed design model from a global requirements model with a behavior defined by sub-collaborations in sequential, alternative, concurrent or looping composition. As in [12,13], the algorithm may introduce coordination messages for strict sequencing. It also deals with weak sequencing and introduces a choice indication (*cim*) message if one of the roles does not participate in all alternatives of a choice, and introduces sequence numbers in all the messages in the body of a loop with weak sequencing between the repetitions. The algorithm assumes that each component has a message pool where received messages are stored until they are consumed in the order required by the component.

In this paper we investigate this problem in more detail. The main contributions are as follows:

- We show that in many cases the *cim* message is not required.
- We discuss the reception of coordinating messages if an alternative with different sets of terminating roles is followed in strict sequence by another sub-collaboration; a case which was not covered in [1].
- We discuss in detail under which conditions sequence numbers in the messages of a weak while loop are required for coordination. In particular, we have a proposition which states under which condition a weak while loop is *directly realizable* (without sequence numbers nor additional coordination messages); we distinguish between networks message delivery with and without FIFO order; and we point out that, in general, not all messages need sequence numbers when the loop is not *directly realizable*.
- We show that the distributed design model for a weak while loop may be constructed such that for certain components the direct realization approach can be taken, while the approaches of [1] or of [14] may be used independently for the other components. The approach of [1] assumes that the message pool has an interface that allows to wait for a message with a specific sequence

number. If such a function is not available, as for instance in typical programming languages interfaces, the approach of [14] (which is more complex) can be used.
– We discuss in detail the functions that the interface of the message pool of a component should provide for the different behavior composition rules.

The paper is organized as follows: In Sect. 2, we present the concept of collaboration which is used for the modeling the behavior of systems. The order of execution of actions is defined in terms of partial orders. We first give an intuitive explanation and some simple examples, then we discuss the formalization of these concepts following Pratt [15] and complement this formal model with the concept of roles which represent the different system components. We also define what it means that a more detailed behavior model (e.g. a distributed design model) conforms to a more abstract requirement model. In Sect. 3, we discuss the some issues related to the realizability of a global requirements model, give a short literature review, discuss two issues related to alternatives, and propose an interface for the message reception pool. Then in Sect. 4, we discuss in detail the derivation of a distributed design model for a weak while loop behavior. Section 5 is the conclusion.

2 Definitions and Notations

2.1 The Concept of Collaboration

As mentioned above, a collaboration is a collection of actions that are performed by a distributed system. In the requirements model, a certain number of roles are identified, and each action is performed by one of these roles such that their execution satisfies a given partial order. Figure 1(a) shows the example of a collaboration $\mathbb{X}$ using a graphical notation borrowed from [9]. Collaboration $\mathbb{X}$ has three roles, x, y, and z, and includes 6 actions, a_i ($i = 1, 2, \ldots 6$). The partial order for the execution of these actions is shown in Fig. 1(b), where $a_i \rightarrow a_j$ means that the execution of action a_i is performed before the execution of action a_j.

For the composition of two collaborations A and B in strict sequence, written "A $;_s$ B", any action of B may only executed when all actions of A have completed. Therefore it is important to identify the initial actions (those for which there are no earlier actions in the partial order) and the final actions (those for which there are no later actions in the partial order). As discussed in [13], for ensuring strict sequencing between A and B, it is sufficient to ensure that all initial actions of B start after all final actions of A have completed. Figure 1(c) shows a more abstract view of collaboration $\mathbb{X}$ showing only the initial and final actions of the collaboration. The order of execution of the actions of collaboration $\mathbb{X}$ is also be presented in Fig. 1(d) using the notation proposed in [8] (adapted from Activity Diagrams). Here the A_i ($i = 1, 2, \ldots 6$) are sub-collaborations, and each A_i contains a single action, namely a_i.

Figure 2 shows two other compositions of collaborations. Figure 2(a) shows a collaboration including two decision points: the possible execution orders are $C_0 ;_s C_1 ;_s C_3$, $C_0 ;_s C_2 ;_s C_3$, and $C_0 ;_s C_2 ;_s C_4 ;_s C_3$, where the C_i are arbitrary sub-collaborations. Figure 2(b) shows a weak while loop where sub-collaboration C_2 is performed after zero, one or more executions of sub-collaboration C_1. The sequencing between successive executions of C_1 and the final execution of C_2 are weak sequences, which means that sequencing is only enforced locally by each role, but not globally.

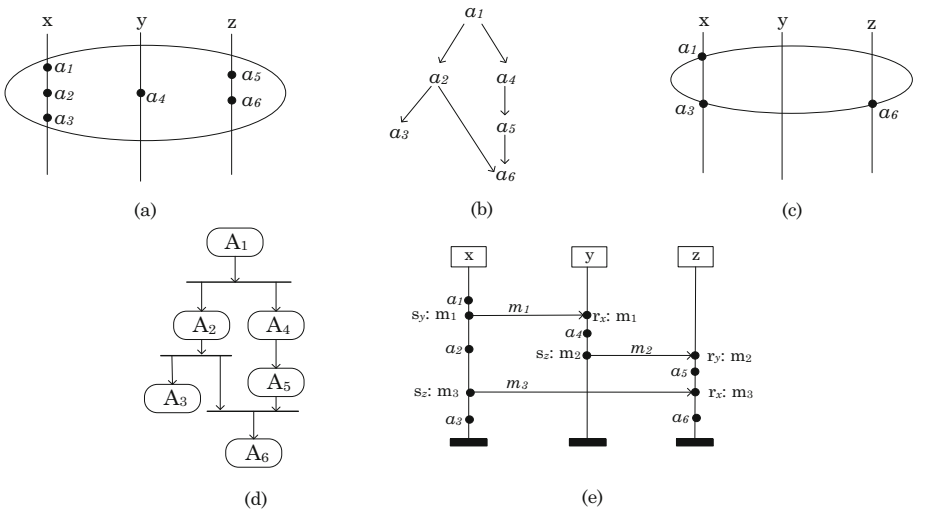

Fig. 1. (a) Example of a collaboration $\mathbb{X}$. (b) The partial order for the execution of the actions of $\mathbb{X}$. (c) An abstract view of the collaboration $\mathbb{X}$, showing only initial and final actions. (d) Another representation for the order of actions executions (adapted from Activity Diagrams). (e) Distributed implementation for collaboration $\mathbb{X}$ using MSC notation

2.2 Behavior of Collaborations: A Formalization

As pointed out by Lamport [16], partial order is a natural concept for describing the execution of distributed systems. Pratt [15] proposed to use labelled partially ordered set (lposet) [17] for this purpose. A (strict) lposet, also called pomset (partially ordered multi-set) is a tuple $(E, \sum, <, l)$, where E is a set of events, $\sum$ is a set of action labels, $< \subseteq E \times E$ is a irreflexive, asymmetric, and transitive order on E (where "$e_1 < e_2$" means that event e_2 is after event e_1, or $e_1 \rightarrow e_2$ and l is a labeling function l: $E \rightarrow \sum$.

The behavior of the collaboration $\mathbb{X}$ shown in Fig. 1(a) can be modelled by a lposet $(E, \sum, <, l)$, where $E = \{e_i |\ i = 1, 2, ... 6\}$, $\sum = \{a_i |\ i = 1, 2, ... 6\}$, $<$ is as shown in Fig. 1(b), and $l(e_i) = a_i$ for $|\ i = 1, 2, ... 6$. Note that the events in the figure are labelled with the action labels, not with the event names.

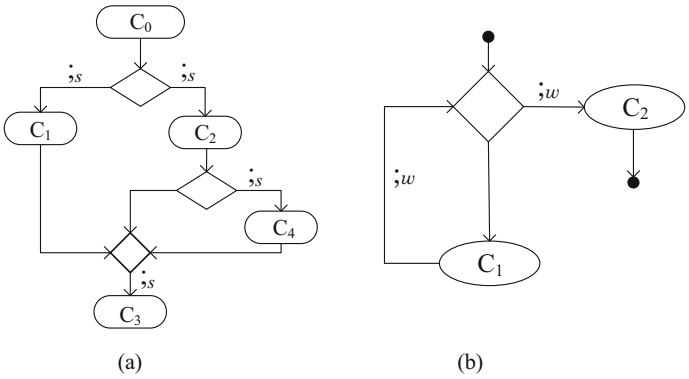

Fig. 2. (a) An example of a composition of collaborations including two decision points. (b) Weak while loop

The collaboration $\mathbb{X}$ shown above is a special case of a requirements model which has a behavior defined by a single pomset. However, in general, the behavior of a collaboration consists of several pomsets. Gischler [18] uses the term "process" to designate a behavioral model, such as a collaboration, and the term "behavior" for the set of pomsets that are allowed for the execution by that model. The following rules are defined for the behavior of process (or collaboration) compositions (see for instance [18]):

The **strict sequence** of two processes C_1 and C_2, written $C_1; s\ C_2$, has the following behavior: the set of all strict concatenations of one pomset in the behavior of C_1 with one pomset in the behavior of C_2 (where the strict combination of two pomsets P_1 and P_2 means that all events of P_2 are after all events of P_1.

The **concurrent** execution of two processes C_1 and C_2, written $C_1 || \ C_2$, has the following behavior: the set of all concurrent combinations of one pomset in the behavior of C_1 with one pomset in the behavior of C_2 (where the concurrent combinations of two pomsets P_1 and P_2 is the pomset that contains the union of events and actions, and no order dependencies between the events of P_1 and P_2.

A **choice** between two alternative processes C_1 and C_2, written $C_1 + \ C_2$, has the following behavior: it is the union of the pomsets in the behavior C_1 and the behavior of C_2.

For an arbitrary number of repetitions of a process C, the Kleene star operator is defined as usual. The behavior of C^{*s} is defined to be strict sequence of zero, one or more repetitions of C in strict sequence.

As an example, the behavior of the collaboration shown in Fig. 2(a) can be defined by the expression $C_0\ ;_s (C_1 + C_2\ ;_s (C_4 + 1));_s\ C_3$, where 1 represents the pomset with an empty set of events. We note that the literature on pomsets usually only considers strict sequencing, which makes abstraction of system components and roles. These concepts are necessary for the derivation of a distributed system design model, and for the definition of weak sequencing

(which was first introduced for MSCs). Therefore we introduce the concept of a collaboration-pomset, which is an extension of a pomset as follows:

Definition 1 (Collaboration-Pomset). *A collaboration-pomset is a tuple* $(E, \sum, <, l, \mathbb{R}, \rho)$*, where* $(E, \sum, <, l)$ *is a pomset,* $\mathbb{R}$ *is a set of roles, and* ρ *is a mapping* $\rho : E \rightarrow \mathbb{R}$*.* ρ *assigns a role to each event.*

Definition 2 (Collaboration-Behavior). *A collaboration-behavior is a set of collaboration-pomsets which have a common set of roles* $\mathbb{R}$*.*

We consider in the following mainly collaborations that have a behavior (i,e, a collaboration-behavior) which can be defined by regular expressions, such as discussed above, or by diagrams, such as shown in Figs. 1 and 2.

Definition 3 (Weak Sequence). *The weak sequence of two collaborations* C_1 *and* C_2*, written* $C_1;_w C_2$*, has the following behavior: the set of all weak concatenations of one collaboration-pomset in the behavior of* C_1 *with one collaboration-pomset in the behavior of* C_2 *(where the weak concatenation of two collaboration-pomsets* P_1 *and* P_2 *means that, for any role r, all events of* P_2 *that are assigned to the role r are after all events of* P_1 *that are assigned to r.*

Note: It was shown in [19] that associativity does not always hold for multiple strict and weak sequencing.

Like the Kleene operator for strict sequencing C^{*s} (mentioned above), we also define arbitrary, multiple weak sequencing using the notation C^{*w} . We consider in Sect. 4 the distributed design model for a weak while loop, as shown in Fig. 2(b), which is defined by the expression "$C_1{}^{*w}$ $;_w C_2$".

2.3 Comparing Two Behavior Models

We use the same modeling concepts for requirement models and distributed system design models, namely collaborations (as defined in Sect. 2.2). In this subsection we ask the question: Does a given design model C_2 conform to a given requirement model C_1? The conformance relation should be defined such that if C_2 conforms to C_1, then any implementation that conforms to C_2 will also conform to C_1.

We assume that a design model C_2 conforming to a more abstract model C_1 should include all events of C_1 associated with the same actions and roles as in C_1, but it may contain additional events that are introduced during the refinement process. We also assume that the partial order defined by C_1 should be realized by C_2, but the order of C_2 may be stronger. Therefore we provide the following definitions.

Definition 4 (Conformance of Pomsets). *Given two collaboration-pomsets* P_1 *and* P_2*, we say that* P_2 *conforms to* P_1 *if the events of* P_2 *include the events of* P_1*, the order of* P_2 *is a refinement of the order of* P_1 *and the restriction of the labelling and role mapping functions of* P_2*, restricted to the events of* P_1*, are equal to the functions of* P_1*.*

Definition 5 (Conformance of Collaborations). *Given two collaborations C_1 and C_2, we say that C_2 conforms to C_1 if for each collaboration-pomset P in the behavior of C_2, there is a collaboration-pomset in the behavior of C_1 to which P conforms.*

3 Deriving Conforming Distributed Design Models

3.1 Basic Ideas

Since the early work in this area [13,20], the following basic ideas were proposed for the derivation of a distributed design model from a global requirements model:

1. The distributed design consists of processes for each role. The processes performed by different roles communicate through the exchange of messages.
2. The process of a given role r is obtained from the global requirements collaboration by projecting its behavior onto role r, that is, by deleting all events that are associated with other roles $r' \neq r$.
3. If an order should be introduced between two actions $a_1 \rightarrow a_2$ associated with different roles r_1 and r_2, respectively, one should introduce a coordination message (called "flow message" in [1]) to be sent by r_1 after the execution of a_1, and to be received by r_2 before the execution of a_2.
4. Each role has a reception pool where received messages are stored until their consumption is requested by the local behavior. We distinguish the following cases:
   (a) A single input queue which receives the messages from all other roles.
   (b) For each other role, messages are transmitted in FIFO order and stored in the pool in separate FIFO queues.
   (c) A simple pool of messages which can be requested for consumption in any order (for instance [20]).

If we apply these principles to the global requirements model of Fig. 1(a) and (b), we obtain the distributed design model of Fig. 1(e), which is shown in the form of a MSC. The messages in this design are introduced according to point (3) above. For the message sending and receiving actions, we use the notation "$s_x : m_1$" and "$r_y : m_2$", respectively, where x and y are the roles to which the message is sent, and the role from where the message was received, and m_1 and m_2 represent the types of the message involved.

It is important to note, that the messages m_2 and m_3 may lead to a race condition at reception by role z, that is, m_3 may arrive before m_2, although it is expected to arrive afterwards. A reception pool of type (b) or (c) (see above) is introduced in order to deal with such race conditions. If such a pool is used by role z, then it may consume these two messages in the order it expects, that is, first m_2 then m_3. (If m_3 arrives before m_2, it will be stored in the pool; z will wait for m_2; and then it will request the consumption of m_3). We note that a reception pool type (a) will lead to a deadlock if m_3 arrives before m_2. Therefore, this type of pool should be avoided.

One says that a global requirements model is **realizable** if a conforming distributed design model can be found. We call **basic implementation** the design model obtained by the basic approach above without any additional coordination message (using point (3)). We say that the requirements model is **directly realizable** if the basic implementation with reception pool conforms to the global requirements. We note that in the case that the global requirements are given in the form of the simple MSC without alternatives, the specification is directly realizable since it contains already all messages required for enforcing the order of the distributed actions (for instance, if we take Fig. 1(e) as the global requirements model).

3.2 Review of Work on Realizability

Realizability of global specifications has been extensively studied by many authors. Different formalisms have been used for defining the global specification, while for the definition of the local behavior of each role normally state machine models were used. The conditions for the realizability of High-level MSC (HMSC for short) have been proposed in [21], for Message Sequence Graph (MSG for short) in [22], and for Compositional MSCs in [20]. Some authors have discussed the pathologies in HMSCs that prevent their realization like non-local choice [2,10].

Global specifications in the form of a set of MSC are considered in [23]. This is related to the problem of implied scenarios. This work is extended in [22] by studying the realizability of MSC-graphs under FIFO communication. In both papers, the specification is realizable if there exist concurrent automata which implement the set of MSCs. Two types of realizability are considered: weak realizability (where the distributed design may deadlock) and safe realizability where no additional deadlock is introduced. In both cases, the behavior for each role is modelled by a finite state machine and communication is through FIFO queues.

In [20], they formally study under which conditions the global specification (compositional MSC) is directly realizable, they prove that the absence of non-deterministic, race and non-local choice lead to sound choice which is directly realizable. Different composition operators (i.e., weak and strong sequence, alternative and parallel) between sub-collaborations are studied in [7,8], and how they affect realizability.

In many cases the specifications are not directly realizable, however, they are realizable by including additional coordination messages or parameter in the implementation. Additional data is added to the messages to achieve safe realizability for MSC specifications [24]. In [25], they consider the realizability of local-choice HMSCs and proof that the implementation strongly conforms to the specification using messages parameters. The authors of [7,8] report when strong and weak sequence need coordination messages to achieve realizability, and [1] introduces the *cim* message for the realization of alternatives. In [1,14], race conditions in weak while loops are studied and an additional message parameter is introduced for obtaining realizability.

3.3 Alternatives with Different Terminating Roles

For enforcing a weak sequence between two collaborations C_1 $;_w$ C_2, no coordination messages are required in the distributed design model since the ordering defined by weak sequence is a local order only. This is different for the strict sequence C_1 $;_s$ C_2 which defines globally that all actions of C_2 must be after all actions of C_1. This can be ensured by introducing coordination messages from all roles performing a final action of C_1 (called terminating roles of C_1) to all roles performing an initial action of C_2 (called initiating roles of C_2) [1,12].

In the case of a collaboration with alternatives C_1 and C_2, followed in strict sequence by another collaboration C_3, the situation is in general more complex, as shown by the example of Fig. 3. This case is not mentioned in [13], and it is excluded from the discussion in [1,12], If the alternatives have different sets of terminating roles, the initiating roles of C_3 have to wait for two alternative sets of coordinating messages. In the example of Fig. 3, the choice between the two alternatives is made by role y (local choice), and role w has to consume, before the action e_6 in collaboration C_3, either two coordination messages from roles x and z, or another message from role z (we assume that all coordination messages can be distinguished by their type).

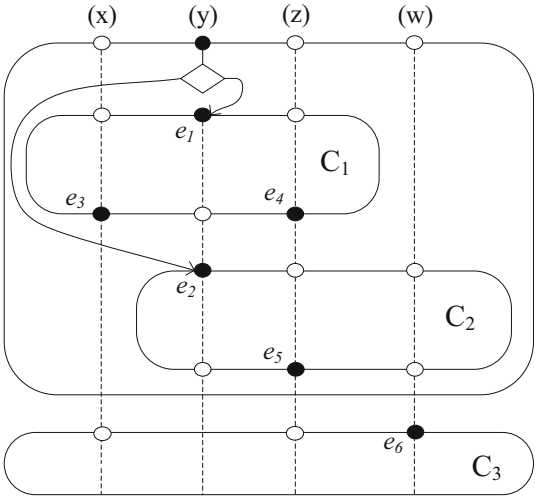

Fig. 3. Alternatives with different terminating roles

3.4 Interface Provided by the Reception Pool

The reception pool should provide an interface to the local behavior at the given role which allows to specify which messages are candidate for consumption. For avoiding the race condition in the behavior of Fig. 1(e), the local behavior at role z would request the consumption of message m_2, and then of message m_3. In

the case of the local behavior of role z in the alternative of Fig. 3, the first action in both alternatives would be the reception of a message. The local behavior would request the consumption of one of these messages and would be informed which message was received. We note that we assume that the messages can all be distinguished by their type (and/or by the sending role). We call such an interface a **basic pool interface**. It can be constructed using the basic Internet socket interface for communication with a single partner. Such an interface is for instance provided by the BPEL programming environment, which is often used for the distributed implementation of Web Service Applications.

We note, however, that this basic interface is not natural for handling strict order between two collaborations. In this case, an initiating role of the second collaboration would start with requesting the consumption of a set of messages, namely all messages to be received from the terminating roles of the first collaboration. The situation becomes even more complex for strict sequence after alternatives with different sets of terminating roles, as discussed above. In this case, the initiating role after the alternative would naturally request two or more alternative sets of messages to be consumed. In the example of Fig. 3, the behavior of role w for sub-collaboration C_3 would start with requesting either the set of messages $\{r_x : m_1,\ r_z : m_2\}$ or $\{r_z : m_3\}$. Such an interface is unfortunately not provided for BPEL programming. Also the interface function which allows for requesting the consumption of a certain message type with a parameter that has a given integer value (which is useful for handling weak while loops, as discussed in the next section), is not available with BPEL.

3.5 A Role Does Not Participate in all Alternatives

In the case of choice between several alternatives, as shown in Fig. 4, where one of the alternatives does not participate in all roles, i.e., alternative A doesn't participate in z, [1] suggested the introduction of a choice indication message *cim* to indicate the choice to those roles that do not participate in the alternative. Without such a coordination message the role z in Fig. 4 would not know when to start the initial action of collaboration C_3 when this alternative is chosen. We note that problem was not mentioned in [20].

Here we would also like to point out that this *cim* message is not required if the subsequent collaboration follows in strict sequence, and in the case of weak sequence only if the role in question has an initiating role in the subsequent collaboration. If in Fig. 4 the m_6 message would go in the opposite direction (and role z would not be initiating), role z could simply request the pool for the consumption of m_5 or m_6.

4 Weak While Loop

We consider in this section a requirements model including a weak while loop as shown in Fig. 2(b). We assume that the decision of repeating collaboration C_1 or finishing with C_2 is a local choice. In the example of Fig. 5, this is done

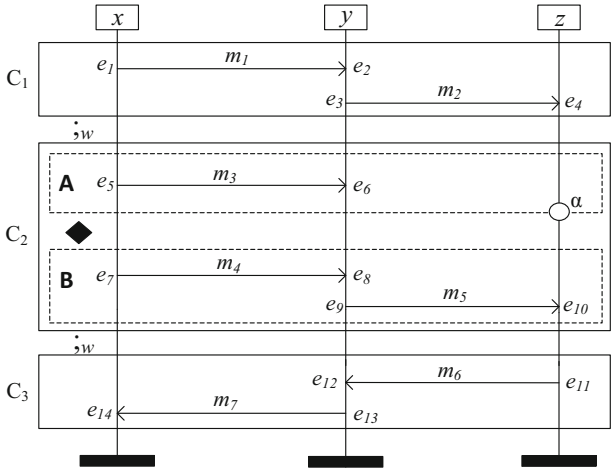

Fig. 4. An example where role z does not participate in all alternatives (α is part of the design model where *cim* must be received)

by role x. We call this role the **initiator** of the loop. The other roles are called **dependent roles**. It is important to note that the first event of a dependent role in C_1 or in C_2 is always the reception of a message (since otherwise the role would be initiating, and the choice would not be local). We also assume that all collaboration-pomsets of C_1 and C_2 involve the same set of roles.

It has been pointed out in the literature that a race may occur for a dependent role between the reception of the first messages of C_1 and C_2. For instance in Fig. 5, if the transmission of m_2 during the last repetition of C_1 is delayed for some reason, the role z may receive m_5 before the last message m_2. We call such a race a **termination race** of the loop.

Another problem that may occur is the following: If a given type of message of C_1 is not transmitted over a FIFO channel from the sending role to the receiving role, then it may happen that the message instance of the n^{th} repetition of C_1 is overtaken by the instance of the next repetition. If the message has no parameters, then there is no problem, but otherwise the parameter values would arrive out of order. We call this problem **message overtaking**. For instance, we note that message type m_8 in Fig. 5 may have message overtaking in the case that C_1 is repeated twice and the transmission of the first message m_8 is very slow.

Proposition 1. *A weak while loop with local choice is directly realizable if it does not contain any termination race nor message overtaking.*

Proof. The absence of termination race means: For any dependent role r that receives the first messages m_1 and m_2 in C_1 and C_2, respectively, it can never happen that the message m_2 is received by r while m_1 still in transit. Also, the absence of message overtaking means: For any dependent role r and any message

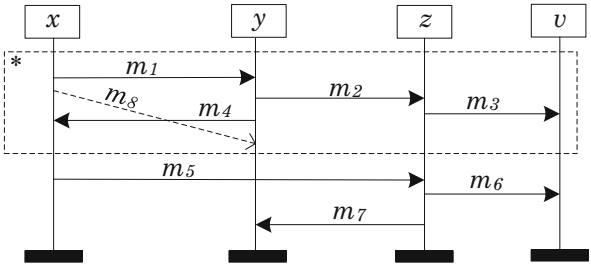

Fig. 5. An example of a weak while loop where role x is the loop initiator

type m received within C_1, it can never happen that a message of type m is overtaken by another instance of that type belonging to the next repetition of C_1. There is no need for any coordination messages since the messages are consumed in the right order. Therefore the weak while loop is directly realizable. □

In the following we discuss under what conditions there are no such problems and how a given role can be implemented in the distributed design model in the case that there are problems.

4.1 Checking the Requirements Model for Problems

We analyse in this subsection the requirements model and give some propositions that ensure the absence of termination race and message overtaking.

Proposition 2 (Absence of termination race). *A dependent role r of a weak while loop has no termination race if one of the following conditions is satisfied:*

(a) The reception of the first message m by r in C_1 is before that last event in C_1 of the initiator.
(b) The first messages received by r in C_1 and C_2 are sent by the same role r', and the communication is over a FIFO channel. Note that it is assumed here that role r' has no termination race.

Proof. **For (a)**: If the first message m in C_1 to be received by r is in transit, then it must have been sent and not yet received. Since the reception is before the last event in C_1 of the initiator, the initiator must be involved in the execution of C_1 when a message m is in transit. Therefore there can never be the first message received by r in C_2 in transit at the same time. Therefore there cannot be a race.

For (b): Since we assume that the sending role r' has no termination race, the first messages in C_1 will be sent before the first message in C_2. Because they are sent over a FIFO channel, they will also be received in this order. □

Proposition 3 (Absence of message overtaking). *For the reception of a message type m by a role r during the repetitions of C_1, there is no danger of message overtaking, if one of the following conditions is satisfied:*

(a) The reception of the message is before the last event in C_1 of the initiator.
(b) The message is received over a FIFO channel from the sending role.
(c) The receiving role is the initiator of the loop.

Proof. The proof for **(a)** is similar to the previous proposition. Point **(b)** is evident. Point **(c)** follows from the fact that the initiator waits for receiving all messages related to one repetition of C_1 before it starts another repetition. □

4.2 Deriving a Distributed Design Model for a Weak While Loop

To get a conforming distributed design model in the case of a termination race, it was suggested in the literature to include in the first messages received by a role in C_1 and in C_2 a sequence parameter which indicates the number of repetitions of C_1, and accept the first message of C_2 only if it contains the right sequence number. This approach was also used in [1], however, for all roles and with sequence parameters in all messages. This, by the way, also solves the message overtaking problem.

In this section we discuss how a conforming design model can be constructed by introducing a minimum number of sequence parameters in messages.

As pointed out by Proposition 1, the basic implementation provides a conforming design model if there are no problems of termination race nor message overtaking. We propose to construct a conforming design model by starting out with the basic implementation, and then performing modifications to the behavior of those roles that have any of these problems.

For a role r that has termination race (TR), one of the following modifications can be used:

- **Modification-TR-1:**
  (a) A sequence parameter is introduced into the first message receive in C_2.
  (b) The behavior has a local variable N which is initialized to 0 before the while loop starts. Each time the first message of C_1 is received, N is incremented.
  (c) In the request to the message pool for consuming the first message of C_1 or C_2, a condition is added to the consumption of the first message of C_2, namely that the parameter value is equal to N.
  (d) This assumes that the behavior of the sending role also has a modification introducing a local variable N (which is incremented) and sending the value of N as message parameter.

This modification-TR-1 corresponds to what is proposed in [1]. It presents the difficulty that a message pool providing a suitable interface must be used. To avoid this difficulty, the following modification was proposed in [14].

– **Modification-TR-2:** Points (a), (b) and (d) as above. Instead of (c), we have the following:
  • A request is given to the message pool to consume the first message of C_1 or of C_2. If the first message in C_2 is received, the message is stored in a buffer and its parameter is stored in a second local variable M. If (M = N), the behavior of C_2 starts using the first message in the buffer. Otherwise the local behavior of C_1 is performed. Then the process goes back to the beginning of the loop and waits again for an instance of first message for C_1 or C_2,

For a role r that has message overtaking(MO) for a message type m, the following modifications can be used:

– **Modification-MO-1:** Points (b) and (d) as in Modification-TR 1. Instead of (a) and (c), we have the following:
  • A sequence parameter is introduced into the message type that has the problem of overtaking.
  • In the request for consumption to the message pool of message m, a condition is added to the consumption, namely that the parameter value is equal to N.

• **Modification-MO-2:** Ensure that the message m is received through a FIFO channel.

Modification-MO-1 has the disadvantage that an additional message parameter must be introduced and the message pool needs to support consumption requests with parameter conditions. It is often much easier to ensure FIFO delivery between the sending and receiving roles.

Proposition 4 (Conforming design model). *Given a requirements model R and a distributed design model D. D conforms to R if the following condition is satisfied: D is obtained from the basic implementation of R by applying the following modifications:*

(a) For each responding role that has a termination race according to R, apply Modification-TR-1 or Modification-TR-2.
(b) For each reception by some depending role of some message with the problem of message overtaking according to R, apply Modification-MO-1 or Modification-MO-2.

Proof. We have to show that the following conditions are satisfied:

(a) Collaboration C_1 is executed by all roles the same number of times.
(b) During the N^{th} execution of C_1 by a given role r, each message m consumed by r was sent by the sending role r' during its N^{th} execution of C_1.

Condition (b) follows from point (b) of the Proposition. It is straightforward to prove that the Modification-MO-1 or Modification-MO-2 assures that there is no message overtaking in the design model. For proving Condition (a), we have to prove that if Modification-TR-1 or Modification-TR-2 are introduced for a role r, this ensures that C_2 is executed by r only after C_1 has been executed N times, where N is the number of time that the loop initiator executed C_1.

For this purpose, we group the roles into role-sets RS(i) ($i = 0, 1, 2, ...$). The initiator is in RS(0) and any dependent role that receives the first message of C_2 from a role in RS(i) is in RS(i+1). Now we do the proof by induction over i. Suppose that the roles in RS(i) execute C_1 the same number of times N as the initiator, then any role in RS(i+1) receives the first message of C_2 with the parameter N. It is easy to see that Modification-TR-1 or Modification-TR-2 ensure that the role will also executed C_1 N times before it executes C_2. When it executes C_2 and sends an initial message to another role, then this message will also include the parameter N. We conclude that all first messages of C_2 will include the same parameter value and therefore all roles will execute C_1 the same number of times. □

5 Conclusion

We consider the derivation of a distributed design model from a global requirements model which identifies the different actions to be performed by the different system components and a partial order that determines the order in which these actions may be performed. The distributed design model defines for each component the local actions to be performed and their order. We call basic implementation a design model obtained by projection of the requirements model onto each component. If this design model conforms to the requirements, we say that the requirements are directly realizable. However, in most cases additional coordination messages or parameters must be introduced to coordinate the order of actions at different components. We study special cases of alternatives followed by strict or weak sequence. We show that the choice indication message *cim*, introduced in [1] is not required in many cases.

We also study the implementation of the weak while loop, which may have the problems of termination race and message overtaking. We show under which conditions these problems are absent, and the loop is directly realizable. For the other cases, we show how a conforming design model can be obtained by introducing minimal changes to the basic implementation. Overall, this is an important improvement over what is proposed in [1].

This work is important in the context of distributed system design where the designers and developers should consider these problems and know how to solve them. This work is also important for the construction of tools that generate code for distributed applications in order to generate code without design flaws.

In the near future, we plan to use the formal partial order description to prove the conformance of the derived design model and to implement the derivation algorithm in a tool environment.

References

1. von Bochmann, G.: Deriving component designs from global requirements. In: CEUR Workshop Proceedings, vol. 503, pp. 55–69 (2008)
2. Ben-Abdallah, H., Leue, S.: Syntactic detection of process divergence and non-local choice in message sequence charts. In: Brinksma, E. (ed.) TACAS 1997. LNCS, vol. 1217, pp. 259–274. Springer, Heidelberg (1997). https://doi.org/10.1007/BFb0035393
3. Mooij, A.J., Goga, N., Romijn, J.M.T.: Non-local choice and beyond: intricacies of MSC choice nodes. In: Cerioli, M. (ed.) FASE 2005. LNCS, vol. 3442, pp. 273–288. Springer, Heidelberg (2005). https://doi.org/10.1007/978-3-540-31984-9_21
4. Alur, R., Holzmann, G.J., Peled, D.: An analyzer for message sequence charts. In: Margaria, T., Steffen, B. (eds.) TACAS 1996. LNCS, vol. 1055, pp. 35–48. Springer, Heidelberg (1996). https://doi.org/10.1007/3-540-61042-1_37
5. ITU-TS, Recommendation Z.120 (02/11), Message Sequence Chart (MSC). ITU, Geneva. Technical report (2011)
6. Castejón, H.N., Bræk, R.: Formalizing collaboration goal sequences for service choreography. In: Najm, E., Pradat-Peyre, J.-F., Donzeau-Gouge, V.V. (eds.) FORTE 2006. LNCS, vol. 4229, pp. 275–291. Springer, Heidelberg (2006). https://doi.org/10.1007/11888116_21
7. Castejon, H.N., Braek, R., von Bochmann, G.: Realizability of collaboration-based service specifications. In: Proceedings - Asia-Pacific Software Engineering Conference, APSEC, pp. 73–80 (2007)
8. Castejón, H.N., von Bochmann, G., Bræk, R.: On the realizability of collaborative services. Softw. Syst. Model. **12**(3), 597–617 (2013)
9. Israr, T., von Bochmann, G.: Performance modeling of distributed collaboration services. In: ICPE 2011-Proceedings of the 2nd Joint WOSP/SIPEW International Conference on Performance Engineering, January 2011, pp. 475–480 (2011)
10. Hélouët, L.: Some pathological message sequence charts, and how to detect them. In: Reed, R., Reed, J. (eds.) SDL 2001. LNCS, vol. 2078, pp. 348–364. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-48213-X_22
11. Object Managment Group: UML 2.5.1 specification. Technical report (2017)
12. Khendek, F., von Bochmann, G., Kant, C.: New results on deriving protocol specifications from service specifications. In: Proceedings of the ACM SIGCOMM 1989, pp. 136–145 (1989)
13. Gotzhein, R., von Bochmann, G.: Deriving protocol specifications from service specifications including parameters. ACM Trans. Comput. Syst. **8**(4), 255–283 (1990)
14. Mustafa, N.M.F., von Bochmann, G.: Transforming dynamic behavior specifications from activity diagrams to BPEL. In: Proceedings of the 6th IEEE International Symposium on Service-Oriented System Engineering, SOSE 2011, pp. 305–311 (2011)
15. Pratt, V.: Modeling concurrency with partial orders. Int. J. Parallel Program. **15**(1), 33–71 (1986)
16. Lamport, L.: Time, clocks, and the ordering of events in a distributed system. Commun. ACM **21**(7), 558–565 (1978)
17. Katoen, J.P., Lambert, L.: Pomsets for message sequence charts. In: Proceeding of First Workshop SDL and MSC (SAM 1998), pp. 197–208 (1998)
18. Gischer, J.L.: The equational theory of pomsets. Theor. Comput. Sci. **61**(2–3), 199–224 (1988)

19. von Bochmann, G.: Associativity between weak and strict sequencing. In: Amyot, D., Fonseca i Casas, P., Mussbacher, G. (eds.) SAM 2014. LNCS, vol. 8769, pp. 96–109. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-11743-0_7
20. Mooij, A., Romijn, J., Wesselink, W.: Realizability criteria for compositional MSC. In: Johnson, M., Vene, V. (eds.) AMAST 2006. LNCS, vol. 4019, pp. 248–262. Springer, Heidelberg (2006). https://doi.org/10.1007/11784180_20
21. Hélouët, L., Jard, C.: Conditions for synthesis of communicating automata from HMSCs. In: Proceedings of 5th International Workshop on Formal Methods for Industrial Critical Systems, March 2000
22. Alur, R., Etessami, K., Yannakakis, M.: Realizability and verification of MSC graphs. Theor. Comput. Sci. **331**(1), 97–114 (2005)
23. Alur, R., Etessami, K., Yannakakis, M.: Inference of message sequence charts. IEEE Trans. Softw. Eng. **29**(7), 623–633 (2003)
24. Baudru, N., Morin, R.: Safe implementability of regular message sequence chart specifications. In: ACIS 4th International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD 2003) (2003)
25. Genest, B., Muscholl, A., Seidl, H., Zeitoun, M.: Infinite-state high-level MSCs: model-checking and realizability. J. Comput. Syst. Sci. **72**(4), 617–647 (2006)