On the Realizability of Collaborative Services[1]

HUMBERTO NICOLÁS CASTEJÓN, GREGOR V. BOCHMANN, AND ROLV BRÆK

Abstract This paper considers compositional specifications of services using UML 2 collaborations, activity and interaction diagrams, and addresses the realizability problem for such specifications: given a global specification, can we construct a set of communicating system components whose joint behavior is precisely the specified global behavior? We approach the problem by looking at how the sequencing of collaborations and local actions may be described using UML activity diagrams. We identify the realizability problems for each of the sequencing operators, such as strong and weak sequence, choice of alternatives, loops, and concurrency. The nature of these realizability problems and possible solutions are discussed. This brings a new look at already known problems: we show that given some conditions, certain problems can already be detected at an abstract level, without looking at the detailed interactions of the collaborations, provided that we know the components that initiate and terminate the different collaborations.

Keywords *service composition, compositional specification of collaborations, realizability of distributed implementations, distributed system design, design guidelines, deriving component behavior from global specifications, workflow for collaborations, UML activity diagrams, service oriented architecture*

1 Introduction

When developing distributed reactive systems there is a need to model behavior both from a global system perspective and a local or component-wise perspective. In many cases the behavior of services provided by a system is not performed by a single component, but by several collaborating components. This has been recognized by several authors, such as [15], [16] and [39], and is sometimes referred to as the "crosscutting" nature of services [25][40]. By "service" we here understand *an identified functionality aiming to establish some desired goals/effects among collaborating entities*. The global perspective is needed to understand, specify and analyze the collaborative behavior of services provided by a system, while the local perspective is needed to precisely design the system and its components.

In the domain of distributed reactive systems, it has been common practice to model the local perspective in terms of loosely coupled components modeled as communicating state

H. N. Castejón is with Telenor Corporate Development, Norway (e-mail: humberto.castejon@telenor.com). Part of this work was done while the author was affiliated with the Department of Telematics, Norwegian University of Science and Technology, Norway.

G. v. Bochmann is with the School of Information Technology and Engineering (SITE), University of Ottawa, Canada (e-mail: bochmann@site.uottawa.ca).

R. Bræk is with the Department of Telematics, Norwegian University of Science and Technology, Norway (e-mail: rolv.braek@item.ntnu.no).

[1] A preliminary version of this paper appeared in the proceedings of the 14th Asia-Pacific Soft. Eng. Conf. (APSEC'07), IEEE Computer Society Press, pp. 73-80, 2007. This work was partially supported by a grant from the Natural Sciences and Engineering Research Council of Canada.

machines [10][14], using languages such as SDL [32] and more recently UML state diagrams [50]. This has helped to substantially improve quality and modularity, mainly by providing means to define complex, reactive behavior precisely in a way that is understandable to humans, as well as suitable for formal analysis and automatic generation of executable code. The drawback of such an approach is that, while each individual component is defined precisely and completely, the behavior of the services the components participate in gets fragmented.

In the global perspective one seeks to define service behavior as precisely and completely as possible. Interaction sequences, such as MSCs [33] and UML sequence diagrams [50], are commonly used to describe global, collaborative behavior, and have proven to be very valuable. In the domain of service oriented business systems, one also separates between global and local behavior in a similar way. Here the global behavior is often referred to as *choreography* [24], while the local behavior is called *orchestration*.

There are some well-known difficulties associated with global behavior models and their relationship to local behaviors:

- **Incompleteness.** Due to the large number of interaction scenarios that are possible in realistic systems, it is normally too cumbersome to define them all, and therefore only typical/important scenarios are specified. The functionality required for the missing scenarios is directly specified during component design without a global view of the involved interactions.
- **Realizability.** A set of local component behaviors must be designed that realizes the specified global behavior. Deriving the component behaviors from the global model leads in many cases to a set of components whose local behavior is correct, but whose joint global behavior is not always the specified one. The realizability problem has been studied from different perspectives. For example, the authors of [5] and [56] address it from the point of view of implied scenarios (i.e. unspecified scenarios that will be generated by any set of components implementing the specified global scenarios), while other authors have studied pathologies in interaction sequences that prevent their realization (e.g. non-local choices [9]).
- **Compositionality.** Being able to define collaborative behavior in a compositional way with clear mapping to compositional component designs.

We have found that the new collaboration concept introduced in UML 2.0 [51] is useful for describing the global perspective of distributed services and approaching the abovementioned

difficulties. UML collaborations are based on ideas that date back to before the UML era [53][54] and model the concept of a service very nicely. They describe structures of collaborating elements, called roles, each performing a specific function, which collectively achieve a goal or accomplish some desired functionality [50].

An interesting feature of collaborations is that they allow a compositional specification of services. Within a collaboration defining a service, the collaborations defining other services may be referenced and used in the form of so-called *collaboration-uses*. Each collaboration-use specifies how the roles of the referenced collaboration are bound to the roles of the containing collaboration. In the following, we will refer to a collaboration that references other collaborations (via collaboration-uses) as a *composite collaboration*, and to the referenced collaborations as *sub-collaborations*. We will refer to collaborations that do not reference other collaborations as *elementary collaborations*[2].

When decomposing collaborations structurally, it often turns out that the resulting elementary collaborations are simple enough that their behavior may be completely defined using, for example, a UML sequence diagram. The following question remains then: How can we define the global behavior or choreography of a composite collaboration in terms of the behaviors of its sub-collaborations (i.e. the global execution ordering of the sub-collaborations)?

Several notations may be used to define a choreography. We have found UML 2 activity diagrams to be a good candidate, as their semantics in terms of token flow rules enable most of the activity orderings needed for the purpose, the concept of activity partitions may be used to represent the collaboration roles, and hierarchical modeling is supported. We note, however, that we use a slightly modified semantics for activity diagrams in order to accommodate weak sequencing, as explained in Section 2.3.

This paper is concerned with the realization problems that may occur when a global collaboration behavior, defined as a choreography of sub-collaborations, is mapped to the local behaviors of distributed components interacting asynchronously using message passing. Interestingly, the activity ordering (token flows) needed to define the choreography also enables us to identify realizability problems and to classify their underlying reasons. Many of these problems can be found by analyzing choreographies at the level of their definition in terms of sub-collaborations without needing to look at the detailed behavior of those sub-

[2] We note that the sub-collaborations of a composite collaboration may be elementary collaborations, as well as composite collaborations.

collaborations. When this is not possible, potential problem spots can be pinpointed so that detailed interaction analysis can focus on those.

In Section 2 we introduce the basic concepts for compositional collaboration modeling and choreography, and present a case study. In Section 3 we present our results concerning realizability. We review related work in Section 4 and conclude with a discussion of the presented work and its possible applications in Section 5.

2 Using Collaborations to Model Services

In this section we introduce a telemedicine service and show how it can be modeled using UML 2 collaborations and a choreography defined in the form of an activity diagram. We discuss thereafter some issues concerning the triggering of collaborations. We conclude the section with a discussion on the use of activity diagrams to describe collaboration choreographies.

2.1 A case study: TeleConsultation

We consider as an example a telemedicine consultation service, in the following called TeleConsultation. A patient is being treated over an extended period of time for an illness that requires frequent tests and consultations with a doctor at the hospital to set the right doses of medicine. The patient has been equipped with the necessary testing equipment at home and a terminal with the necessary software. The patient will call the hospital on a regular basis to consult with a doctor and have remote tests done. A consultation may proceed as follows:

1. The patient uses the terminal to access a virtual reception desk at the hospital and to request a consultation session with a doctor assigned to this kind of consultation.
2. If no doctor is available, the patient will be put on hold, possibly listening to music, until a doctor is available. If the patient does not want to wait he/she may hang up (and call back later).
3. When a doctor becomes available while the patient is still waiting, the doctor is assigned to the patient.
4. A voice connection is established between the patient terminal and the doctor terminal allowing the consultation to take place.
5. During the consultation the doctor may perform remote tests using the equipment located at the patient's site and a central data logging facility located at the hospital.

The doctor evaluates the results and advises the patient about further treatment. Either the doctor or the patient may end the consultation call.

6. After the consultation call is ended, the doctor may spend some time updating the patient journal and doing other necessary work before signaling that he/she is available for a new call. The doctor may signal that he/she is unavailable when leaving office for a longer period, or going off-duty.

In the following sub-sections we discuss how the structure and behavior of the TeleConsultation service can be modeled with the help of UML 2 collaborations, activity diagrams and sequence diagrams.

2.1.1 Service structure: UML 2 Collaborations

UML 2 collaborations are well-suited to represent the structural aspects of services. They are both structured classifiers and behaviored classifiers. As structured classifiers they define a structure of roles, connectors and collaboration-uses. The latter represent occurrences of collaborations in a given context and can be used to represent the sub-collaborations that take place among the roles of a service. Figure 1 shows a collaboration describing the structure of the *TeleConsultation* service, which consists of four roles (*PatTrm*, *DocTrm*, *VRecDsk*, *DataLogger*) and seven collaboration-uses (*rd*:*RequestDoc*, *w:Waiting*, *wd:Withdraw*, *as:Assign*, *av:Available*, *u:Unavailable*, *c:Consultation*) denoting the occurrence of sub-collaborations among the roles. We note that we have chosen to keep the patient and the doctor outside the collaboration, and let them be represented by their terminals (*PatTrm*, *DocTrm*). This means that interactions across the user interfaces are not represented, only what goes on between the two terminals, the virtual reception desk (*VRecDsk*) and the data logger (*DataLogger*).

The collaboration-uses in the *TeleConsultation* collaboration (Figure 1) refer to collaborations that are defined separately and then reused in the *TeleConsultation* service (e.g. the *Consultation* collaboration is defined in Figure 4(a) and reused via the *c:Consultation* collaboration-use). Each collaboration-use defines how the roles of the referenced sub-collaboration are bound to the roles of the enclosing collaboration (e.g. the diagram in Figure 1 specifies that the role *pt* of *Consultation* is bound to the role *PatTrm* of *TeleConsultation*, so that *PatTrm* will behave as *pt* when involved in the *Consultation* sub-collaboration).

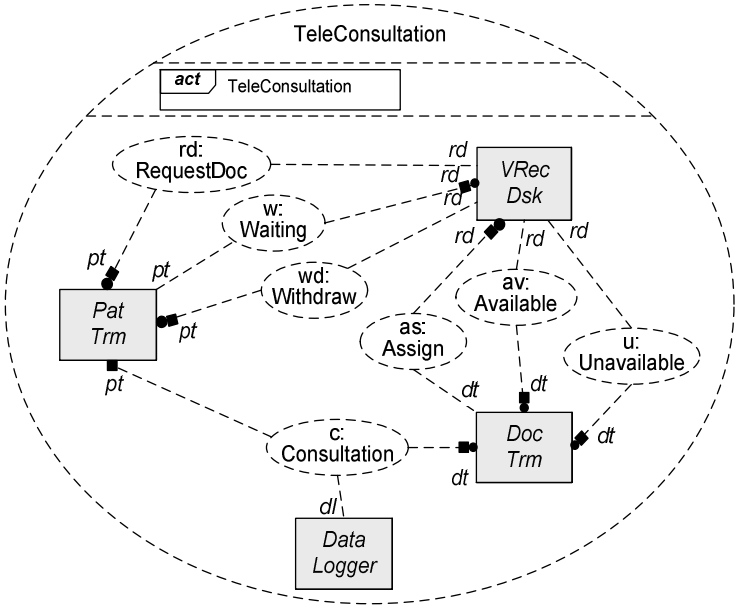

Figure 1. The *TeleConsultation* service as a UML 2 collaboration

We note that the "dots" and "squares" in the diagram of Figure 1 are not standard UML. They have been used to indicate the initiating and terminating roles, respectively, of each sub-collaboration[3]. A role is an **initiating role** of a collaboration *C* if it takes the initiative to start the collaboration (i.e. the first local action it performs within *C* is not preceded by any other local action within *C* (by any other role)). The terminating roles are defined similarly. In the following, we will say that a role of a composite collaboration is the initiating (resp. terminating) role of a sub-collaboration if it is bound to the initiating (resp. terminating) role of that sub-collaboration.

2.1.2 Service behavior: Choreography

Given that elementary collaborations are defined using sequence diagrams, UML interaction overview diagrams are a-priori good candidates to describe collaboration choreographies. However, since interaction overview diagrams are rather loosely defined as a restricted form of activity diagrams (although with different semantics), we prefer using the more general activity diagrams for choreography description. Activity diagrams provide more general forms of execution orders, such as interruptions and non-properly nested joins and forks.

[3] In order to allow UML compliant tools to use this notation, a UML "stereotype" can be introduced that refines "ConnectableElement" with two Boolean attributes that, for each role involved in a collaboration, indicate whether the role is an initiating or terminating role of the collaboration.

They also allow representing roles as partitions, which is necessary to enable us to analyze realizability at the level of a choreography (see Section 3).

The activity diagram in Figure 2 defines the global behavior for the *TeleConsultation* collaboration using the following conventions:

- The activity nodes are *CallBehaviorActions* that invoke the behavior associated with a collaboration type (i.e. an activity diagram or sequence diagram) made available via a collaboration-use (in Figure 1). In addition, we assume that the behavior specification of each collaboration has the same name as the collaboration itself (e.g. the behavior of the *Consultation* collaboration is defined by an activity diagram with the same name – see Figure 4(b)). In this way the diagram defines a choreography of, possibly nested, collaboration behaviors. In Figure 2, the *c.Consultation* activity node is a CallBehaviorAction invoking the *Consultation* behavior (see Figure 4(b)) defined in the scope of the *Consultation* collaboration (see Figure 4(a)) and made available in *TeleConsultation* via the *c:Consultation* collaboration-use.
- The participating roles are indicated by partitions of the activities (separated using dashed lines)[4].
- The initiating and terminating roles of the invoked collaborations are indicated using dots and squares, respectively. As in the case of collaborations, this notation is not part of UML activity diagrams, but may be provided by additional profiling.

Note how the diagram in Figure 2 defines the order of execution of the *TeleConsultation*'s sub-collaborations in a visual way without going into the details of those sub-collaborations and their message exchanges. Still, some realizability problems can already be detected from the information that the diagram provides, as further discussed in Section 3. Also note that while our use of UML activity diagrams is syntactically correct, we allow weak sequencing semantics as further discussed in Section 2.3.

Finally, we mention that it is often useful to introduce variables that are used to define guards for alternate choices or sub-activity invocations. They typically represent databases or state variables and are important for the description of the overall system behavior. At the early stages of development, these variables may be considered global variables (as in Use Case Maps [6]). At the later stages, they must be allocated to particular system components or replaced by other means of keeping the pertinent information.

[4] Note that UML leaves the notation of partitions open.

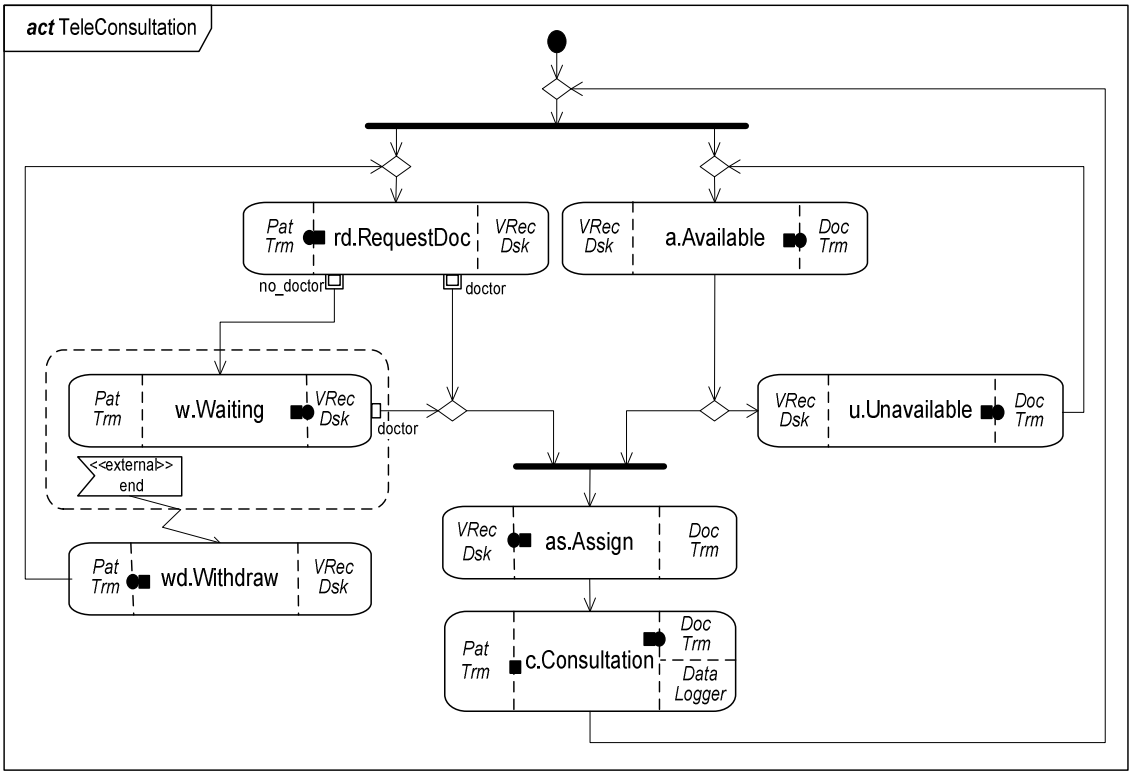

Figure 2. Choreography for the *TeleConsultation* collaboration

2.1.3 Adding tests to the TeleConsultation

During a consultation, the doctor may initiate and carry out some tests. Testing involves three roles: a test unit (*tu*) at the patient site; a central data-logger (*dl*) at the hospital, and the doctor terminal (*dt*). The *Test* collaboration and its choreography are shown in Figure 3.

We have now defined two separate collaborations *TeleConsultation* and *Test* without any formal binding among them. This illustrates how collaborations can be used to define services separately. An independently specified service collaboration may then be used in the definition of another service collaboration by means of a collaboration-use. In the case of this example, the *Test* shall be invoked as part of the *Consultation*. The definition of the latter collaboration has therefore been refined to include *Test* as a sub-collaboration by means of the *t:Test* collaboration-use (see Figure 4(a)). The choreography of *Consultation* has been accordingly modified to invoke the behavior of *Test* in the appropriate order, as shown in Figure 4 (b)[5].

[5] Another way of modeling the invocation of a sub-collaborations is by using streaming pins, as explained in [17]

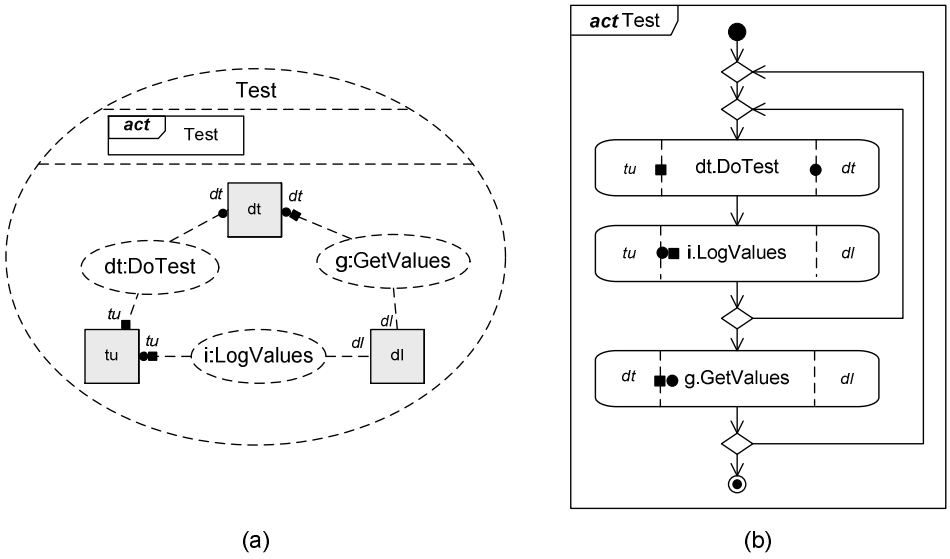

Figure 3. The *Test* collaboration with choreography

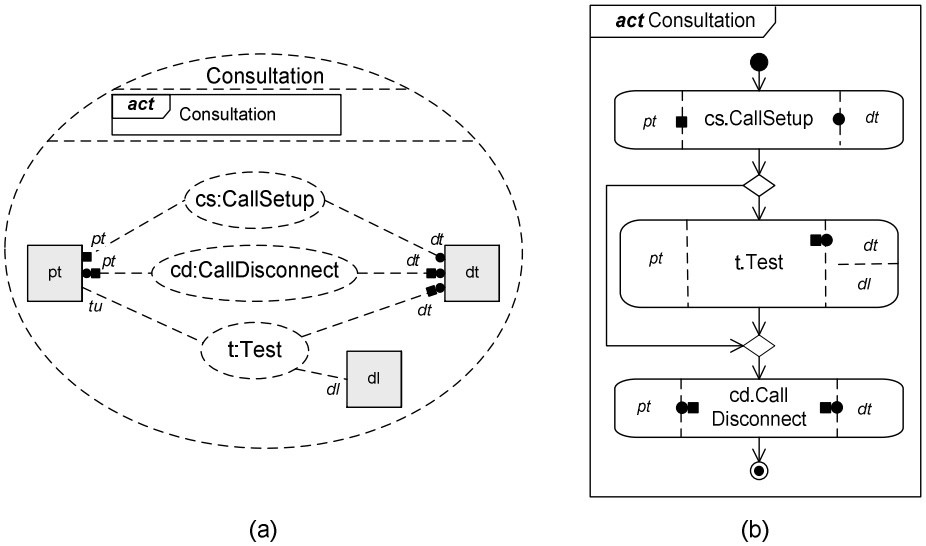

Figure 4. Consultation with *Test*

2.2 Collaboration triggering

For each collaboration, we may identify a *triggering event* that leads to the execution of the collaboration. Triggering events are always external to the collaboration that they trigger. With composite collaborations, the triggering event of a sub-collaboration may be an event generated in another sub-collaboration. For example, in the case of sequential execution of two sub-collaborations, the triggering event of the second sub-collaboration is usually the completion of a terminating action of the first sub-collaboration. In other cases, however, the triggering event of a sub-collaboration may be external to the enclosing composite

collaboration (i.e. the reception of an external input generated by actions in the environment of the composite collaboration, or a time-out). Such *environmental triggering events* may cause a role to initiate a sub-collaboration seemingly spontaneously and on its own initiative.

Environmental triggering events need not be specified explicitly, but the *initiatives* (i.e. the seemingly spontaneous actions) that they cause need to be identified. This is important in service modeling for three reasons: (1) most services and service features are initiated by environmental initiatives; (2) they give rise to concurrency and potential conflicts; and (3) they represent links with other specified collaborations or external entities outside the specified system (e.g. users or external devices) that cause the triggering event. Initiatives of different roles normally occur independently. They start threads of sequential behavior, which execute (partly) in parallel with the behavior triggered by other initiatives. In the *TeleConsultation* service, for example, the *PatTrm* and the *DocTrm* roles behave concurrently and may take initiatives independently of each other. This is reflected by a choreography (see Figure 2) with two concurrent parts that may be considered as different views of the service; the *PatTrm* view and the *DocTrm* view. These are brought together and coordinated during the *Assign* and *Consultation* collaborations.

We note that collaborations with more than one (non-alternative) initiating role may not be so easy to realize if they are triggered by independent initiatives, and coordination between the initiating roles is needed. As a general design guideline, it is therefore desirable to avoid collaborations with multiple initiating roles as far as possible, although it cannot always be avoided. Indeed, the existence of independent initiatives and the need for their coordination is an essential property of the *TeleConsultation* service and many other services. Independent initiatives may give rise to conflicts, if they are not properly coordinated. This will be discussed further in Section 3.

2.3 Weak sequencing semantics

Sequential execution is a fundamental ordering constraint, which in the simplest case specifies that a collaboration C_2 is executed after a collaboration C_1. In UML activity diagrams, this can be modeled by drawing a flow edge from the activity node invoking C_1 to the node invoking C_2. With the basic rules for sequential control flow, this implies that all actions of C_1 must be completed before any action of C_2 may start. This is known as *strong sequencing.* Sometimes, strong sequencing is more restrictive than necessary and *weak*

sequencing is more appropriate, which means that any role involved in C_2 may start participating in that collaboration as soon as it has completed all its local actions for C_1.

Weak sequencing is the nature of distributed systems, and is the default semantics for sequential execution in notations such as High-Level Message Sequence Charts [33] and UML Interactions (including interaction overview diagrams). In a collaboration with several initiating roles, the different initiating roles may start the execution of their part of the collaboration independently of one another, and therefore at different times. Also the terminating actions of a collaboration with several terminating roles may be executed at different times by the different roles. Therefore, we consider weak sequencing the normal semantics for choreographies, while strong sequencing is a property that may emerge from sufficient coordination through messages, which can either be part of the collaborations being composed or be added upon composition to ensure the strong sequencing, if this is desired.

When using activity diagrams for describing the dynamic behaviors of choreographies, we therefore use a semantics that differs from the standard UML semantics in the following points:

1. In a collaboration with several initiating roles, the different initiating roles may start the execution of their part of the collaboration independently of one another, and therefore at different times. Similarly, the terminating actions of a collaboration may be executed at different times.
2. Control flow edges between different activities have the meaning of weak sequencing[6] (unless explicitly specified as strong sequence).

3 Analyzing the realizability of choreographies

In this section we consider choreographies describing the global behavior of composite service collaborations (i.e. the global execution ordering of their sub-collaborations), as discussed in Section 2. We then discuss the type of realizability problems that may arise when trying to obtain distributed system designs realizing these choreographies. It turns out that certain realizability difficulties can be identified by simply analyzing the execution order of the sub-collaborations, if we have the knowledge about participating, initiating and

[6] We note that weak sequencing may in principle be described in standard UML activity diagrams with the help of streaming Parameters (and pins), which allow an action execution to take inputs and provide outputs while it is executing. However, these are defined in the CompleteActivities package, and are thus not applicable to UML Interactions. Therefore, streaming Parameters cannot be used when invoking sequence diagrams via CallBehaviorActions.

terminating roles. That is, there is no need to look into the detailed behavior, in terms of message exchanges, of those sub-collaborations.

In the first subsection we define the concept of direct realization of a choreography and discuss when a choreography is said to be *directly realizable*. In the subsequent subsections, we discuss separately the analysis for the different ordering concepts that are used for defining the order of execution of sub-collaborations, including sequential execution (strong and weak sequencing), alternatives, concurrency, and interruption. These ordering concepts are supported by many modeling languages, including UML activity and sequence diagrams, Use Case Maps, BPEL and many others.

In the following, when discussing direct realizability for the choreography of a composite collaboration, some conventions will be adopted:

- The term *role* will be used to denote the roles of the composite collaboration.
- The term *collaboration* will be used to denote the sub-collaborations of the composite collaboration.
- We will say that a role of the composite collaboration participates in one of its sub-collaborations if it is bound to any of the roles of the sub-collaboration, and that is the initiating (resp. terminating) role of the sub-collaboration if it is bound to the initiating (resp. terminating) role of the sub-collaboration.

3.1 Analysis framework

3.1.1 Realization of distributed system designs

Here we consider a straightforward approach to the realization of a distributed design from a choreography specifying the global behavior of a composite collaboration, which we call **direct realization**[7]. The direct realization assumes that there is one system component for each collaboration role, whose dynamic behavior is modeled in terms of message receptions, message sendings and local actions, and the order in which these actions may occur. The dynamic behavior of each component is obtained by projecting the dynamic behavior of the choreography onto the role played by the component (i.e. removing any action executed by other roles). No extra coordination messages or attributes are added during the projection. We call the set of system components obtained by direct realization a **directly realized system**.

We assume now that the dynamic behavior of the choreography is defined by an activity

[7] Refer to [21] for a more formal approach

diagram invoking sequence diagrams, and possibly other activity diagrams in a hierarchical fashion, through so-called CallBehaviorActions (as described in Section 2). In order to simplify the projection, we suggest obtaining first a flattened choreography. For this purpose, we replace each CallBehaviorAction invoking an elementary sub-collaboration defined by a sequence diagram by an activity diagram representing the sequence of local actions defined by the sequence diagram. Each CallBehaviorAction invoking a composite sub-collaboration is also replaced by a flattened activity diagram describing the sub-collaboration´s choreography. The projection of this flattened global behavior, for a component realizing the role R, can then be obtained by copying the flattened behavior definition and replacing each send, receive or local action to be executed by a role different from R by "no operation".

We note that the resulting activity diagram, representing the behavior of a given component, may sometimes include a decision node were both alternatives start with the reception of a message. The choice between these alternatives is in this case made by the first message received. As explained in the definition of UML [50], a decision node may offer a token to both alternatives, but the token may only be consumed by an alternative when the token has been accepted by the first action of that alternative. We assume that an AcceptEventAction only accepts a token when the corresponding message has already been received; the execution of the AcceptEventAction corresponds then to the consumption of that message.

We note, however, that the dynamic behavior for component designs is usually modeled by state machines. Therefore, the activity diagram obtained by the above projection method may be translated into an equivalent UML state machine. Such a translation is in most cases quite straightforward if fork and join nodes in the activity diagram are well nested. In such a case they can be translated into fork and join pseudo-states in the state machine, and the nodes they enclose be placed in concurrent regions of a composite state (see [17]).

3.1.2 Realizability of choreographies

One may expect that the interworking of the components obtained by direct realization of a choreography may lead to a global behavior that is identical to the behavior defined by the choreography. Unfortunately, in many cases, the directly realized components may get *stuck* while interacting with each other. Their interworking may also give rise to interaction scenarios not foreseen by the specification. The problem of these *implied scenarios* was originally studied for MSC-based specifications in [4]. This problem is, however, not unique

to MSCs, but inherent to any specification language where the behavior of a distributed system is described from a global perspective, while it is realized by loosely coupled components with only local knowledge.

We consider in the following all possible execution traces. A *trace* is a sequence of message send and receive actions, and other local actions, in the order in which they are performed by the different components of the system during a particular execution. We say that a trace is **complete** if the system execution ends in a **global final state**, that is, a system state where each component is in a local state that can be accepted as final, and no message remains in transit or in an input buffer.

We say that a choreography (or collaboration) is **directly realizable** if the following two conditions are met:

(1) The set of complete traces generated by the directly realized system is equal to the set of complete traces defined by the choreography.

(2) Each trace generated by the directly realized system can always be extended to a complete trace.

The second condition implies that from any reachable system state (which was reached by a particular execution sequence) a global final state can be reached (through the extended complete trace). We say then that the system is **stuck-free** (see also [26] and [48]). This condition rules out the following design flaws:

- *Deadlock*: A component in a non-final local state waits for a message that will never be sent.
- *Unspecified reception*: A component receives a message for which there is no transition to consume it in its current state.
- *Orphan message*: This is a special kind of unspecified reception where the execution context for which this message was intended does not exist any more. This term was introduced for object-oriented systems where the destination of a message is an object; if the destination object is destroyed before the message arrives, the message becomes an orphan.

In the following sections we will discuss under which circumstances a choreography is directly realizable. We will discuss for each ordering concept what problems of direct realizability may occur, how they may be detected, and what kind of additional mechanisms could be introduced into the directly realized design model in order to assure that the resulting behavior conforms to the choreography. These mechanisms include additional

coordination messages, and additional parameters in the messages of the directly realized design. Provided we know the initiating and terminating roles, we are in many situations able to identify problems by looking only at the sub-collaboration ordering defined by the choreography. In other cases, we are able to identify potential problems at the choreography level, but need to consider detailed interactions of the sub-collaborations to see whether the problems actually exist.

3.1.3 Message ordering

It is important to note that the question whether a choreography is directly realizable depends not only on the ordering defined by the choreography, but also on the characteristics of the underlying communication service that is used for the transmission of messages between the different system components. Important characteristics of the communication service are the type of transmission channels, and the type of input buffering at each component. We assume that there is no message loss, and distinguish between channels with out-of-order delivery (i.e. messages sent from a given source to a given destination may be received in a different order than they were sent) and channels with in-order delivery. Concerning the input buffering we distinguish between the following schemes of message reordering for consumption:

- *No reordering*: Each component has a single FIFO buffer in which all received messages are stored until they are processed. Messages are consumed in FIFO order.
- *Reordering between sources*: A component has separate FIFO buffers for messages received from different source components, and may locally determine from which source the next message should be consumed.
- *Full reordering*: A component may freely reorder received messages for consumption.

3.2 Sequential execution

3.2.1 Strong Sequence

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 roles before C_2 can be initiated. It requires a direct precedence relation between the terminating actions of C_1 and the initiating actions of C_2, so that the latter can only happen after the former are finished. The situation is particularly simple in the case of a localized sequence as defined below.

Definition 1 (localized sequence). *A sequence* $C_1 \circ C_2$ *is* localized *if all terminating actions of* C_1 *and all initiating actions of* C_2 *are performed by the same role.*

An example of a localized sequence can be found in the *TeleConsultation* service between the collaborations *DoTest* and *LogValues* (see Figure 3(b)). We note that in the case of localized sequences, there is no semantic difference between strong and weak sequencing. We have the following proposition.

Proposition 1. *A strong sequence of two directly realizable collaborations,* $C_1 \circ_s C_2$*, is directly realizable iff it is localized.*

Proof.[8] ($\Leftarrow$) In order to prove that the sequence is directly realizable, we need to prove that (1) its direct realization is stuck-free, and (2) the set of complete traces it defines is the same as the set of complete traces generated by its realization. If the sequence is localized, then the following property is true: (**P**) each role is completely finished in C_1 before it sends or receives any message in C_2. Since C_1 and C_2 are both directly realizable, a role behavior may only get stuck if either there is a race and it receives a message from C_2, while still participating in C_1, or if it receives an orphan message from C_1 while already participating in C_2. However, due to **P**, neither of these cases is possible, which proves (1). The traces specified by the choreography will always be generated by the realized system (as a result of the direct realization algorithm). Due to **P**, it is easy to see that all traces generated by the realization will also be specified by the choreography, since every role will always execute all events of C_1 (in the specified order, since it is directly realizable) before any event of C_2. This proves (2). Proving the other direction of the clause ($\Rightarrow$) is easy by contradiction assuming that the composition is not localized. (*End of proof*)

Note that the localization property of a sequence can be checked at the choreography level, that is, by considering just the initiating and terminating roles (without the detailed behavior) of the collaborations.

We note that the proposed semantics for choreography graphs considers any sequence to be *weak* by default (see next sub-section). A designer may still tag an edge in the choreography graph as "strong". An analysis tool could then check whether the sequence defined by the

[8] Formal proofs of the propositions in this paper can be found in [21]

edge is localized and therefore strong and directly realizable. If not, coordination messages could be automatically added by a synthesis algorithm [11] from the terminating roles of the preceding collaboration to the initiating roles of the succeeding collaboration.

3.2.2 Weak Sequence

Weak sequencing of two collaborations C_1 and C_2, written $C_1 \circ_w C_2$, basically requires each role in C_2 to be completely finished with C_1 before it may initiate participation in C_2. This means that the actions in the two collaborations are sequenced on a per-role basis. This corresponds to the semantics of MSCs and UML Interaction diagrams.

Weak sequencing introduces implicit concurrency between the composed collaborations, since their actions 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 counter-intuitive and/or not directly realizable. Consider, for example, the sequence of the collaborations in the *Test* choreography (see Figure 3). According to the basic weak sequence semantics, role *DocTrm* (acting as role *dt*) may initiate collaboration *GetValues* as soon as it has finished with *DoTest*. As a result, collaborations *LogValues* and *GetValues* may be executed in any order in the directly realized system. This is not only counter-intuitive to the specification, which we assume reflects the designer's intention (i.e. *GetValues* should be executed after *LogValues*, with some allowed overlapping), but may also lead to realizability problems. Note that the sequence $LogValues \circ_w GetValues$ has two initiating roles, that is, *TestUnit* and *DocTrm*, which may be executed concurrently. As a guideline, such initial concurrency should be avoided in order to ensure some causality between initiatives. This is ensured by the following property.

Definition 2 (weak-causality). *A weak sequence of two collaborations,* $C_1 \circ_w C_2$, *is* weakly-causal *if each initiating role of* C_2 *participates in* C_1.

This property can be checked at the collaboration level. We note that weak-causality is enforced in the so-called local-HMSCs of [27].

Consider now the sequence diagram in Figure 5, which relates to the choreographies of *Test* (Figure 3b) and *Consultation* (Figure 4b). It is easy to see that the sequence between *Test* and *CallDisconnect* is weakly-causal, but not directly realizable. The *PatTrm* role may initiate *CallDisconnect* just after receiving the message *ack* in *Test*. Therefore, the actions initiated by *PatTrm* in *CallDisconnect* may overlap with the actions in *Test* that follow the

reception of message *ack.* For example, message *disc* may be received at *DocTrm* before the message *report*. This message reception order has not been explicitly specified, and is therefore an implied scenario. Note that such problems may only occur when a role (here *DocTrm*) participates in two consecutive collaborations and plays a non-initiating role in the second one. This fact is summed up by the following proposition.

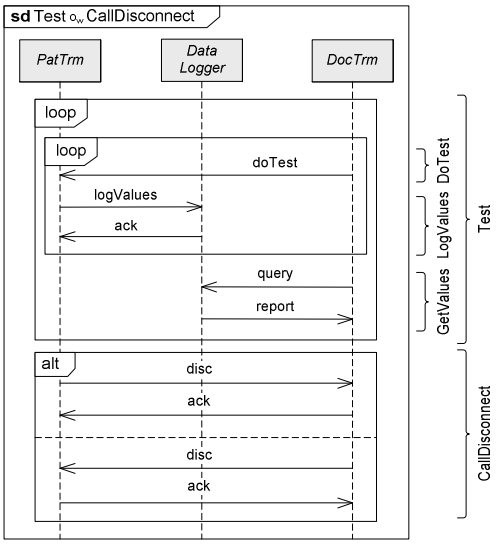

Figure 5. Sequence diagram showing the sequence of *Test* and *CallDisconnect*

Proposition 2. *A weakly-causal sequence of two directly realizable collaborations,* $C_1 \circ_w C_2$*, is directly realizable if no role participating in* C_1 *participates in* C_2 *as a non-initiating role.*

Proof. Given the condition in the proposition, let us first prove the following property (**P**): each role is completely finished in C_1 before it sends or receives any message in C_2. There are two possible cases. The one where a role does not participate in C_1 is trivial. In the other case, where a role *p* participates in both C_1 and C_2, *p* is initiating and will therefore always begin its participation in C_2 with a message sending. The direct realization algorithm ensures that such message sending will never happen until *p* has executed all actions in C_1. And since C_2 is directly realizable, any other actions performed by *p* in C_2 will also always happen after *p* is finished with C_1, which proves **P**. Considering this property, and following the same reasoning used in the proof of Proposition 1, we can easily prove that the direct realization of the sequence is stuck-free and it generates a set of complete traces equal to the ones defined by the sequential choreography. (*End of proof*)

Proposition 2 can be easily checked at the choreography level and represents a situation

where weak sequencing is unproblematic (e.g. given that *RequestDoc* and *Assign* are directly realizable, their weakly-causal sequencing in the choreography of Figure 2 is directly realizable according to Proposition 2). We note, however, that the condition required by Proposition 2 may be too restrictive in many applications. If such condition is not respected (i.e. there are roles participating in C_2 as non-initiating roles that also participate in C_1, such as in the sequence *RequestDoc* and *Waiting* in Figure 2), messages may be received in a different order than specified by the choreography. In the literature about MSCs, this situation is usually called a **race condition** [2]. In general, a race condition can occur when the specification requires a receiving event to happen after another event, and both events are performed by the same component. The reason lies in the controllability of events. While a component can control when its sending events should happen, it cannot control the timing of its receiving events. The actual occurrence of races highly depends on the underlying communication service being used. Channels with in-order delivery prevent races in the communication between a given pair of roles, but do not prevent races when more than two roles are involved.

A stronger property that helps to reduce the number of races and facilitates their detection is *send-causality*, which requires all sending events to be totally ordered.

Definition 3 (send-causal sequence). A weak sequence $C_1 \circ_w C_2$ *is* send-causal *if the role initiating* C_2 *is either the terminating role of* C_1 *or the role that performs the last sending event of* C_1.

We note that, for the sake of simplicity, Definition 3 assumes that each collaboration has only a single initiating event and a single last sending event, but the definition could be easily generalized to consider multiple ones.

Definition 4 (send-causal collaboration). *A collaboration C is send-causal if:*

1) *C is a single message transmission, or*
2) $C = C_1 \circ_w C_2$, *where* C_1 *and* C_2 *are send-causal collaborations, and the weak sequencing is send-causal.*

Note that the collaboration *C* is introduced here as a kind of bracketing for the purpose of defining sequences of collaborations and for analyzing such sequences. It is not to be understood as composition of activity diagrams in general. We assume that such composition is done already in the given collaborations and activity diagrams that we analyze.

It has been shown in [20] 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). This is captured by the following proposition.

Proposition 3. *In a send-causal collaboration, race conditions are only possible between two events performed by a role if both of them are receiving events and the role does not perform any sending event between them.*

Lemma 1. *In a send-causal collaboration using a communication service with in-order-delivery, races are only possible among messages sent by different roles.*

Given Proposition 3, it is clear that if a sequence $C_1 \circ_w C_2$ is send-causal, a *potential race condition* exists for a role *p* if the last action it executes in C_1 is a message reception (i.e. *p* is a terminating role in C_1) and the first action it plays in C_2 is another message reception (i.e. *p* is a non-initiating role in C_2). Whether the 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 roles. For example, in the *TeleConsultation* service, the collaborations *Available* and *Assign* are composed in weak sequence (see Figure 2). Role *DocTrm* plays a terminating role in *Available*, while it plays a non-initiating role in *Assign*. Therefore, a potential race condition exists at *DocTrm* between the receptions of the last message in *Available* and the first message in *Assign*. Since both messages have the same source (i.e. *VRecDsk*), this race can only occur in the case of out-of-order delivery. Note that we can identify this potential race simply by considering the initiating and terminating roles in the choreography in Figure 2.

Proposition 4. *A send-causal sequence of two directly realizable collaboration, $C_1 \circ_w C_2$, is directly realizable*

- *over a communication service with* in-order delivery *if whenever a role plays a terminating role in C_1 and a non-initiating role in C_2, then the last message it receives in C_1 and the first one it receives in C_2 are sent by the same peer role; or*
- *over a communication service with* out-of-order delivery *only if no role plays a terminating role in C_1 and a non-initiating role in C_2.*

Proof. Given the conditions in the proposition, let us first prove the following property (**P**): each role is completely finished in C_1 before it sends or receives any message in C_2. We

have to consider only two cases: (1) a role *p* plays a non-terminating role in C_1 (i.e. ends with a sending) and a non-initiating role in C_2; and (2) a role *p* plays a terminating role in C_1 and a non-initiating role in C_2. The other cases are covered by Proposition 3 (since send-causality implies weak-causality). For case (1), we note that since the sequence is send-causal and C_1 is directly realizable, C_1 must be send-causal. Then, the last sending by *p* in C_1 will always happen before the first sending in C_2 and, thus, before the first reception by *p* in C_2, and before any other action by *p* in C_2 (due to direct realizability of C_2). For case (2), the condition in the proposition ensures that there is no race between the last reception by *p* in C_1 and its first reception in C_2; and given that C_1 and C_2 are directly realizable, this guarantees that *p* is finished with C_1 before it sends or receives any message in C_2, which proves **P**. Considering this property, and following the same reasoning used in the proof of Proposition 1, we can easily prove that the direct realization of the sequence is stuck-free and it generates a set of complete traces equal to the ones defined by the choreography. (*End of proof*)

In order to check Proposition 4, we need to identify the role that sends the last message of C_1 and the first message of C_2. This is straightforward in the case of binary collaborations, if we know which roles play the initiating and terminating roles (as in the aforementioned case of the sequence of *Available* and *Assign*). Thus, for binary collaborations, Proposition 4 allows us to determine whether their sequence is directly realizable and identify actual races at the choreography level, without considering their detailed behaviors. In the case of n-ary collaborations, we can still perform the same early analysis, but only potential races may be discovered. That information could then be used to direct the detailed analysis of the behavioral specification (i.e. the choreography). For example, looking at the choreography in Figure 4b, we can see that *pt* plays a terminating role in *CallSetup* and a non-initiating role in *Test*. In the case of out-of-order delivery, Proposition 4 tells us that there is a race at *pt*. However, in the case of in-order delivery, we can only determine that there is a potential race, but without knowing whether the first message that *pt* receives in *Test* is sent by *dt* or *dl*, we cannot conclude whether the race can actually occur.

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.

3.2.3 Sequences Combining Strong and Weak Steps

In the previous sub-sections we have considered sequences of just two collaborations. In practice, choreographies describe longer sequences where some sequencing steps may be weak and other steps strong. The overall behavior described by such sequences and their realizability depends on the order in which the weak and strong sequences are combined (i.e. on whether priority is given to weak sequencing or to strong sequencing) [41]. Consider, for example, the sequence $Consultation \circ_w Available \circ_s Unavailable$ described by the *TeleConsultation* choreography (assuming the sequencing between *Available* and *Unavailable* was specified as strong) (see Figure 2). If priority is given to strong sequencing over weak sequencing, the meaning of the sequence is $Consultation \circ_w (Available \circ_s Unavailable)$. That means that only the *Available* collaboration is required to be completely finished before *Unavailable* is initiated. Since *Available* is terminated by the same role that initiates *Unavailable*, the specified strong sequencing is directly realizable. If, on the contrary, priority is given to weak sequencing over strong sequencing, the meaning of the sequence becomes $(Consultation \circ_w Available) \circ_s Unavailable$. Now, both *Consultation* and *Available* are required to be completely finished before *Unavailable* is initiated. This strong sequencing is however not directly realizable, since in those cases where *PatTrm* executes the last action of *Consultation*, that collaboration may still not be finished before *Unavailable* is initiated. In this paper we assume that priority is given to strong sequencing, that is, that given any two collaborations directly connected by a strong sequence link in a choreography, the localized property is only required for the sequence of those two collaborations.

We also note that when dealing with sequences of more than two collaborations race conditions may not only appear between *directly* sequenced collaborations, but also between collaborations that are not in direct sequence. This is because a role in a collaboration that is weakly sequenced may remain active during several succeeding collaborations. This "propagation" of weak sequencing makes it more difficult to avoid races. Consider the sequence $RequestDoc \circ_w Assign \circ_w Consultation$ defined by the choreography of the *TeleConsultation* service (see Figure 2). Looking at each of the two weak sequence steps separately does not reveal any direct realizability problem (i.e. there are no races between *RequestDoc* and *Assign* or between *Assign* and *Consultation*). However, there is a race

condition between *RequestDoc* and *Consultation* for the *PatTrm* role. In this case, it is the weak sequencing between *RequestDoc* and *Assign* that makes such race possible. Indeed, the *PatTrm* role may still be active in *RequestDoc* (i.e. not finished) when *Assign* is initiated, and since *PatTrm* does not participate in *Assign*, it may still be active when *Consultation* is initiated. We therefore say that there is **indirect weak sequencing** between *RequestDoc* and *Consultation*, which may lead to races, such as in this example. Thus, when analyzing the direct realizability of a sequence of more than two collaborations, it is not sufficient to check the direct realizability of each pair of directly sequenced collaborations, but we also need to look for races due to indirect weak sequencing. Although Propositions 1, 2 and 4 only consider sequences of two collaborations, and do not therefore take into account indirect weak sequencing, we can still use the results of those propositions in an incremental way. Since weak sequencing is associative[9], given a weak sequence $C_1 \circ_w C_2 \ldots \circ_w C_n$ (n>2) of directly realizable collaborations, we may choose any sub-sequence $C_i \circ_w C_{i+1}$ of two adjacent collaborations and use Proposition 2 or 4 to check its realizability. We may then consider this sub-sequence as a new collaboration, replace the latter in the original sequence, thus reducing its length by one, and repeat the process. The set of initiating roles of the new collaboration would be the union of the initiating roles of C_i and the initiating roles of C_{i+1} that do not participate in C_i. Similarly, the set of its terminating roles would be the union of the terminating roles of C_{i+1} and the terminating roles of C_i that do not participate in C_{i+1}.

For sequences where some sequencing steps are weak and others are strong, and given that strong sequencing has priority (as discussed above), we could use a similar approach where we first check all the strong sequencing steps using Proposition 1, and thereafter the weak sequencing steps as explained above.

3.2.4 Resolution of Race Conditions

Race conditions can be resolved in several ways. Some authors [22][45] have proposed mechanisms to automatically eliminate race conditions by means of synchronization messages. We note that when the send-causality property is satisfied, a 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. For example, in the *TeleConsultation* service (see Figure 2) the race condition

[9] The associativity of weak sequencing has been proved, for example, in [29] with respect to a trace-based semantics. A similar result could be obtained for weak sequencing in the context of this paper.

between *RequestDoc* and *Consultation* for role *PatTrm* may be eliminated by introducing strong sequencing between *RequestDoc* and *Assign*.

Other authors (e.g. [37],[47]) 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. In general, this reordering for consumption 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. [37] 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. It corresponds to the full reordering for consumption capability mentioned in Section 3.1.3. Finally, races may also be eliminated if an explicit consumption of messages in all possible orders is specified (i.e. similar to co-regions in MSCs). We note that in the presence of choices, message reordering may only be possible if the messages to be reordered are marked with the *id* of the collaboration instance that they belong to (see Section 3.3.3).

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 3.3), 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. there is no race at *PatTrm* between *RequestDoc* and *Consultation* 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.

3.3 Alternatives

We consider here the case that at some point of the execution of a choreography, a choice exists between two or more alternative collaborations. Such choices are specified by means of decision nodes, and lead to different execution paths.

In a choice, one or more *choosing* roles decide the alternative to be executed, based on the (implicit or explicit) conditions associated with the alternatives. These roles are initiating roles. The other *non-choosing* roles involved in the choice follow the decision made by the choosing roles (i.e. execute the alternative chosen by the latter). These roles are non-initiating roles. It is thus important that:

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

In the following we study how each of these aspects affect the direct realizability of a choice. We note that a choice can be seen as a sequence with one inlet and a set of alternative outlets. The propositions and guidelines for sequencing, given in the previous section, apply to each path through the choice. However, we will see how the choice-propagation process affects the resolution of races.

We assume that the collaborations in each path of the choice are weakly-causally sequenced, and therefore consider that the set of choosing roles is the union of the initiating roles of all collaborations directly connected to the decision node.

3.3.1 Decision-making Process

We may distinguish the following situations:

1. The enabling conditions of the alternative collaborations are mutually exclusive; only one of the collaborations can be initiated.
2. The enabling conditions of several alternative collaborations could be true; if the initiating roles of these collaborations are different and there is no coordination between them, several alternatives may be initiated concurrently. We call this situation *mixed initiatives*. In many cases this is due to uncoordinated external triggering events, represented by independent initiatives in the collaborations. We distinguish the following two sub-cases:
   a. The alternative collaborations have different goals; only one of them should

succeed. We call this situation *competing initiatives*.

b. The alternative collaborations have the same goal; there is no conflict between them at the semantic level, however, there is a conflict at the level of message exchanges. Example: the *PatTrm* and *DocTrm* roles simultaneously initiate the *CallDisconnect* collaboration during the *Consultation* collaboration, see Figure 4 and Figure 5. We call this situation *mixed initiatives with common goals*.

Local Choice. Deciding the alternative collaboration to be executed becomes simple if there is only one choosing role, and the enabling conditions and triggering events of all alternative collaborations are local to that role (i.e. they are expressed in terms of observable predicates, and events). Choices with this property are called **local**. It is easy to see that the decision-making process of local choices is directly realizable, since the decision is made by a single role based only on its local knowledge.

Non-local choice. Choices involving more than one choosing role are usually called **non-local choices** [9]. 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. [9], [31]). Despite the extensive attention they have received, there is no consensus on how they should be treated. We believe this might be due to a lack of understanding of their nature. Some authors (e.g. [9]) consider them as the result of an under-specification 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* [27][30], that forbid non-local choices. Finally, there are authors [28][46] that consider non-local choices to be inevitable in the specification of distributed systems with autonomous processes. They propose to address them at the design level, and propose a generic implementation approach for non-local choices.

A non-local choice shows up at the choreography level as a choice where the alternative collaborations have different initiating roles. We may avoid the problem of mixed initiatives by coordinating these initiating roles (e.g. either with additional messages or with additional message contents). This would make the choice local in practice. Unfortunately, such coordination is not always feasible. If the alternative collaborations are triggered by independent external events (represented by independent initiatives), we call the choice an **initiative choice**. In these choices the occurrence of mixed initiatives is unavoidable. This is the case for a non-local choice in the *TeleConsultation* service: after the execution of

Available, there is a choice between *Assign*, which is initiated by the *VRecDsk* role, and *Unavailable*, which is initiated by *DocTrm* (see Figure 2). The events that trigger the execution of these collaborations come from the end-users (i.e. the actual patient, who triggers the *RequestDoc* collaboration, which in turn triggers *Assign*; and the actual doctor), which operate independently and are not coordinated. It makes little sense to coordinate the components playing the *PatTrm* and *DocTrm* roles 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. It is an initiative choice.

Any role involved in two or more alternatives of an initiative choice may be potentially used to detect a mixed initiative and initiate the resolution. For such roles, the mixed initiatives reveal themselves in the role behavior as choices between the reception and the sending of a message, or between the receptions of two messages from different peers. In cases where alternative collaborations with different choosing roles have no common roles, an arbiter role should be introduced. Such arbiter role would act as an intermediary between the choosing and non-choosing roles, and could detect a mixed initiative conflict.

Situations of initiative choices were discussed by Gouda et al. [28] and Mooij et al. [46]. These authors propose some resolution approaches. In the domain of communication protocols, Gouda et al. [28] 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. [46] 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 (see Section 3.5), which is not considered in either [28] or [46].

3.3.2 Choice-propagation

The decision made by the choosing role must be properly propagated to the non-choosing roles, in order for them to execute the right alternative. In each alternative path, the behavior of a non-choosing role begins with the reception of a sequence of messages, which we call

the *triggering trace* (in most cases, a single message). Thereafter, the role may send and receive other messages. It is the triggering trace that enables a non-choosing role to determine the alternative collaboration selected by the choosing role(s). In some cases, however, a non-choosing role may not be able to determine the decision made by the choosing role. As an example, we consider the local choice in Figure 6. For the role *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 *C1* or *C2*. That is, *R1*'s decision is ambiguously propagated to *R3*. We say a choice has **ambiguous propagation** if for some non-choosing role the triggering trace of one alternative is a prefix of the triggering trace of another alternative (or if they are identical). Note that if the triggering traces of two alternatives have a common prefix but differ in the suffix part (sometimes called initial ambiguous propagation), the non-choosing role may still make the right choice by delaying the decision until a complete triggering trace had been received. However, in a direct realization the role would have to make the decision after the reception of the first message.

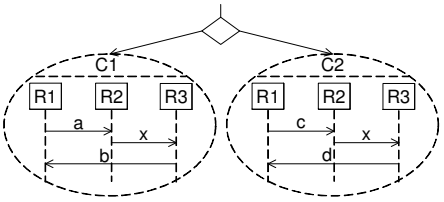

Figure 6. Choice with ambiguous propagation

Choices with ambiguous propagation are not directly realizable. They are similar to the non-deterministic choices defined in [47]. Unfortunately, ambiguous propagation cannot be detected at the choreography level as it depends on the detailed interactions of the collaborations. In order to avoid ambiguous propagation, [11] suggested the introduction of a message parameter that indicates to which branch of the choice the message belongs.

If any of the alternative paths contains a weak sequence with a race condition, the race may make the choice-propagation ambiguous. Consider, for example, the local choice between *DoTest* and *GetValues* in the choreography of the *Test* collaboration in Figure 3. Since the sequencing between *LogValues* and *GetValues* is not weakly-causal, a race exists for the *DataLogger* role between the reception of message *logValues* and the reception of message *query*, which allows for the scenario depicted in Figure 7(a). In that scenario, the choosing role of the choice (i.e. *DocTrm*) decides to execute *DoTest* and *LogValues* the first time it reaches the choice, and *GetValues* the second time it passes through the choice. The *DataLogger* role should do the same, but it receives the query message from the *GetValues*

collaboration before the *logValues* message from the *LogValues* collaboration. As a result, *DataLogger* decides to execute the *GetValues* collaboration the first and only time it reaches the choice. This example shows that in the presence of race conditions the triggering trace *observed* by a non-choosing role may differ from the specified one. Therefore, whenever race conditions may appear in any of the alternative paths, we need to consider the potentially observable triggering traces in the analysis of choice propagation. We say a choice has **race propagation** if there is ambiguous propagation due to races. Choices with race propagation, which are similar to the race choices defined in [47], are not directly realizable.

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

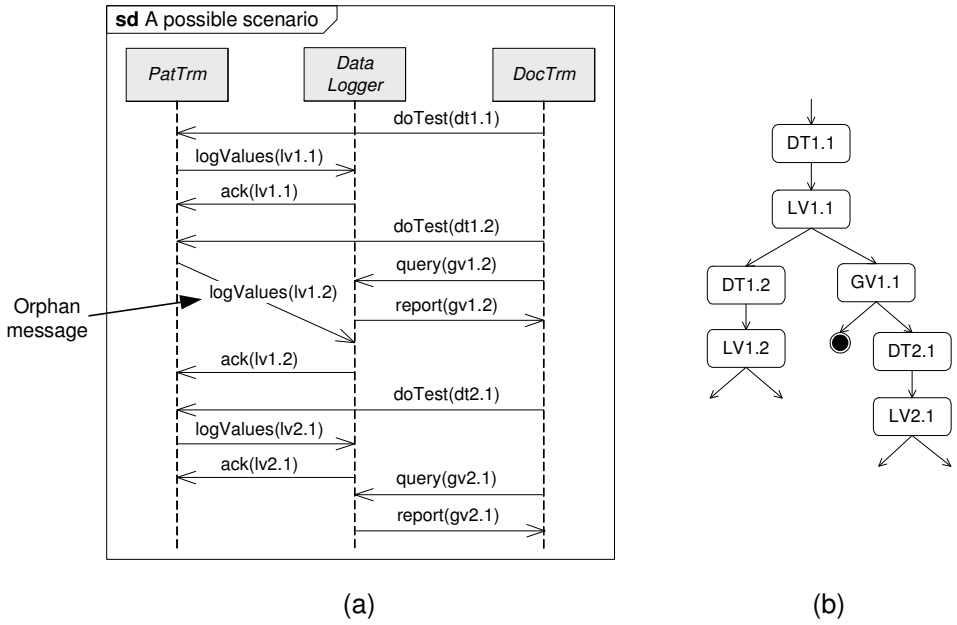

Figure 7. (a) Behavior implied by the choreography of *Test* (see also Figure 5); (b) Unfolding of the choreography of *Test* (see Figure 3)

3.3.3 Resolution of Race Propagation

To resolve the problem of race propagation we need to resolve the race(s) that lead to it. If we try to do it 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 role 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, as illustrated by the race propagation in Figure 7(a). To make the message reordering work, we need to mark the messages with the identifier (id) of the collaboration *instance* they belong to. In order to obtain such an id, we need to unfold the branches of the choice in the choreography

graph, so that they do not share any activity. Then, we need to assign a different id to each activity referring to the same collaboration. When loops are involved, we need to consider the number of iterations of the loop in order to create the collaboration ids. Figure 7(b) illustrates the unfolding of the choice between the *DoTest* and *GetValues* collaborations in Figure 3, and a possible assignment of distinct collaboration ids. In this case, a nested numbering of collaborations has been used due to the presence of nested loops. By labeling messages with these ids, the scenario in Figure 7(a) could be avoided. Upon reception of the first *query* message, with label *gv1.2*, the *DataLogger* role could determine whether it should consume it right away (if it had already received all *logValues* messages sent by *PatTrm*), or whether it should keep it in the buffer until one or more *logValues* messages were received.

In [27] the realizability of local-HMSCs is studied. The authors propose to implement the behavior of each role by means of a simple linear algorithm that marks messages with the id of the HMSC node they belong to. This is basically the same idea that we have just discussed for the resolution of race propagation. The authors do not explain, however, the need of different ids for different occurrences of the same HMSC node, neither how such unique ids may be achieved. Moreover, they propose to mark all messages, and not only those involved in a race propagation, which unnecessarily increases the complexity of the role behaviors. By explicitly detecting the cases of race propagation, we can decide upon the optimal resolution, and only mark affected messages if reordering for consumption is the chosen solution.

3.4 Merge

When two or more preceding flows merge into a single successor flow, this may be seen as a set of sequences where each preceding flow is composed with the succeeding flow. The propositions and guidelines for sequencing given in Section 3.2 apply to each path through the merge.

3.5 Loop

Loops can be used to describe the repeated execution of one or more collaborations, which we call the *body* of the loop. A loop can be seen as a shortcut for a strong or weak sequence of several executions of the same body, combined with a choice and a merge (see e.g. Figure 3). This means that the rules for strong/weak sequencing with choices and merges must be applied. We note that all executions of a loop involve the same set of roles. This fact makes the chances for races high when weak sequencing is used, even though the weak-causality

property is always satisfied. Strong sequencing should therefore be preferred in loops. When strong sequencing is specified between any two executions of the body (e.g. to be sure that one iteration is completely finished before the next one starts), the body should be initiated and terminated by the same role. When send-causal weak sequencing is specified, the role initiating the body should be the one sending or receiving the last message exchanged in the body. It should also be the one initiating the first collaboration after the loop, in order to avoid a non-local choice.

Loops may give rise to so-called *process divergence* [9], characterized by a role sending an unbounded number of messages ahead of the receiving role. This may happen with weak sequencing if the communication between any two of the participants in the body is unidirectional. They may also give rise to so-called *orphan* messages, that is, messages sent in one iteration and received in a later iteration. An example of an orphan message can be seen in Figure 7(a). The second *logValues* message is sent as part of the first iteration of the big loop in the choreography of Figure 3. However, it is consumed as part of a second iteration of the loop. The resulting behavior may be fatal, since the test results obtained by the doctor (via the *DocTrm* role) would not be correct.

Situations similar to loops occur if several occurrences of the same collaboration may be weakly sequenced (e.g. several consecutive sessions of a service).

3.6 Concurrency

Concurrency means that several collaborations are executed independently from one another, possibly at the same time. We use forks and joins to describe concurrency, and we require that they are properly nested as in UML Interaction Overview Diagrams. Concurrent collaborations are directly realizable as long as they are completely independent (i.e. their executions do not interfere with each other). This is clearly the case when there is no overlap among the roles. When a role participates in several concurrent collaborations it must be able to distinguish messages from the different collaborations. Otherwise, messages belonging to one collaboration may be consumed within a different collaboration.

In the *TeleConsultation* example, the virtual receptionist participates in two concurrent flows, and this indicates that the flows are partially dependent. In this case the receptionist serves to coordinate the doctor and the patient. Concurrent activities often involve shared resources for which there is competition that requires coordination. Seen from the patient's point of view, the doctors are shared resources and the coordination is performed by the

receptionist.

Indirect dependencies may also exist through passive shared resources, and shared information. In this case, appropriate coordination has to be added between the collaborations, which will normally be service-specific. In [18] and [19] we discussed the automatic detection of problems 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 collaborations, and could also be used to detect interactions between concurrent collaborations composed using forks and joins.

In a fork, a preceding flow is followed sequentially by a set of two or more succeeding flows running concurrently. The opposite takes place in a join; a set of two or more preceding flows running concurrently is followed sequentially by a single succeeding flow. For each sequence of collaborations defined by a fork/join, the conditions for (weak/strong) sequencing explained in Section 3.2 apply. For direct realizability of strong sequencing, all the collaborations immediately succeeding a fork must be initiated by the role terminating the collaboration preceding the fork. Similarly, all the collaborations immediately preceding a join must terminate at the role initiating the collaboration succeeding the join. If this is not the case, coordination messages may be added before the join/fork to ensure strong sequencing [11].

3.7 Interruption

We consider here the interruption of a collaboration C by another collaboration C^{int} that may become enabled as soon as C is initiated, or when C reaches a certain state. C^{int} requires a triggering event to be initiated, normally in the form of a request coming from an external user or another active agent. In the *TeleConsultation* service the execution of the external event *end*, performed by the patient, results in the interruption of the *Waiting* collaboration by the *Withdraw* collaboration.

As noted in [34], a semantics for cancellation with immediate termination of all activities in the interrupted process is not directly realizable in a distributed system. Instead, one has to assume that the cancellation takes some time to propagate to all participants in the interrupted collaboration, which means that certain activities of the interrupted process may still proceed for some time after the cancellation has been initiated. For example, a client may send a request to a server and, shortly after that, decide to send a cancellation message. While this message is on the way, the server would continue processing the request, and may even send

a response back to the client before it receives the cancellation message. The client would then receive an unexpected response message. Similarly, the server would receive a non-expected cancellation message.

Interruption composition is akin to mixed initiatives where the preempting collaboration has priority. Interruption implies that resolution behavior must be added. However, with interruptions the existence of mixed initiatives is clearly visible in the choreography. The detection is thus easy at the choreography level.

4 Related work on realizability

The realizability of specifications of reactive systems was first studied, in general terms, in [1]. In the context of MSC-based specifications it was first considered in [4], where the authors relate the problem of realizability to the notion of implied scenarios. The authors propose two notions of realizability, depending on whether the realization is required to be deadlock-free (*safe realizability*) or not (*weak realizability*). This work was extended in [5] to consider realizability of *bounded* HMSCs [3]. Reference [43] extends in turn [5] and provides some complexity results for a less restrictive class of HMSCs. Realizability of HMSCs with synchronous communication is considered in [56]. The authors present a technique to detect implied scenarios from a specification consisting of both positive and negative scenarios. The realizability notion considered in [5] and [43] does not allow adding data into messages or adding extra synchronization messages. This is seen as a very restrictive notion of realizability by some authors, who propose a notion of realizability where additional data can be incorporated into messages [49] [8] [27]. The authors of [49] study safe realizability, with additional message contents, of regular (finite state) HMSCs with FIFO channels. This work is extended in [8], where the authors consider non-FIFO communication, and identify a subclass of HMSCs, so-called *coherent* HMSCs, which are safely realizable with additional message contents. However, checking whether an HMSC is coherent is in general hard. Reference [27] discusses two classes of unbounded HMSCs. They claim that so-called *local-choice* HMSCs are always safely realizable with additional message contents[10]. A subclass of *local-choice* HMSCs that are safely realizable without additional message contents was studied in [30].

Other authors have studied conditions for realizability of Compositional MSCs [47] and

[10] Although their claim is true, the authors do not explain the proper format of message contents, as we discussed in Section 3.3.3

pathologies in HMSCs [9][31] and UML sequence diagrams [7] that prevent their realization. None of these works discusses the nature of the realization problems.

More recently, much work has been done on the realizability of behavior specifications written in the Choreography Description Language (CDL) developed for Web Services by the W3C consortium. This language essentially defines in which order message exchanges and local actions occur in the global context. Formalizations of this language in terms of process algebras or Petri nets have been proposed. In all this work, the question of realizability is considered in a manner very similar to what is in this paper: from the global choreography specification, local behaviors for each participating role are obtained by projection, which leads to what we called the directly realized system in Section 3.1.1; then the behavior of this distributed system is compared with the behavior of the original choreography using some appropriate conformance relation, which in most cases is trace equivalence with stuck-freeness, as discussed in Section 3.1.2.

The authors of [55] use the Chor process algebra [52] to formalize CDL behavior descriptions and use the process algebra FSP [44] for defining the role behaviors (which requires some translation from Chor to FSP). Then the LTSA toolbox, associated with FSP, is used to check the consistency of the obtained role behaviors. This paper, like most of the other papers that deal with CDL, only considers rendezvous communication between the different roles, since this is the model used by CDL. This means that the issues of weak sequencing and races, as discussed in this paper, are not dealt with, since they cannot occur in the context of rendezvous communication.

A very similar approach is described in [23], where variations of Petri nets are used for the formalization of CDL and the local role behaviors. The work in [42] considers rendezvous communication (synchronous communication) and asynchronous message passing. However, the behavior descriptions are limited to finite state machine models which do not allow for directly modeling concurrency and hierarchical descriptions, as discussed in this paper. The work in [13] also considers asynchronous communication and allows for overtaking of messages. This paper considers that each role has a local message pool, similar to [46] and [47], where received messages are stored until the local role is ready for its consumption.

5 Conclusions and Applications

In Sections 1 and 2 we introduced collaborations as a structuring framework for compositional modeling of the global behavior of distributed reactive services. We have

demonstrated how choreographies of collaborations defined using activity diagrams can be used for service specification at a higher level than interactions and at the same time help to identify and address realization problems. To our best knowledge we are able to identify all the realization problems that have been reported in the literature. Many of them can be identified at the level of the choreography without needing to consider the detailed interactions of the collaborations that are "choreographed".

A range of techniques proposed in the literature are able to determine whether there exists a realizability problem in a specification and to eventually show the consequences of such problem. They fail, however, to determine the nature or cause of that problem. We believe that having a clear understanding of the actual nature of realizability problems is essential, not only to adopt the most appropriate resolution when they are detected, but also to avoid them in the first place. This has motivated us to investigate, for each possible way of ordering collaborations in a choreography, the realizability problems that may arise and the underlying reasons leading to them.

We believe that our rules for detecting realizability problems, for cases of possibly longer sequences of collaborations, are complete in the sense that any realizability problem in such collaborations will be detected by our rules. More formal proofs of the propositions in this paper are provided in [21].

The results of this paper can be used in different contexts, such as the followings:

- *Detection of realizability problems*: This can be carried out at different levels of automation. As defined in this paper, the rules may be used as a checklist for identifying realizability problems in given choreography specifications. An automated tool for analyzing a given choreography may be implemented based on the algorithms proposed in [17].
- *Transformation into realizable choreography specifications*: Once some realizability problem has been identified, one is usually interested in modifying the specification in order to avoid the problem. Many possible ways to resolve such problems are described in this paper for the different situations that may occur. This is more difficult to automate because some of the proposed modifications imply changes to the behavior of the specifications which must be validated against the user.
- *Transformation into a set of provably correct local role behaviors*: As the final step in the design of collaborative services, one is usually interested in finding a set of

local role behaviors that have the property that their joint execution will lead to an overall system behavior that conforms to the choreography specification. As we discussed in this paper, this problem becomes trivial if the choreography specification is directly realizable. In other situations, a solution can often be found by introducing certain coordination messages, as discussed in the paper. Like for the first point above, one may envision different levels of automation: one may use the results of this paper as a set of rules about how to obtain correct role behaviors – providing solutions in many cases. An automated tool for this purpose was also developed [41]; however, it implements the algorithms described in [12] which are limited to so-called well-structured choreographies.

Several case studies have been carried out to validate the approach proposed in this paper. The most comprehensive of these studies deals with train control and train handover in the new European Rail Traffic Management System (ERTMS). The standards documents for ERTMS specify the behavior rather informally using text and fragments of sequence diagrams. We first encapsulated the sequence diagrams in collaborations and then developed several global choreographies that covered different roles and interfaces. Several cases of potential races due to weak sequencing could be identified directly in the choreography, by considering the initiating and terminating roles. None of these turned out to be problematic because the underlying communication medium conserves the message ordering. The choreographies were elaborated gradually towards a complete definition of the global behavior for the control of one train. In the end, it was necessary to use UML streaming flows and streaming pins to handle all the event orderings and inherent concurrency of the problem. Thus streaming flows seem to be a necessary addition to the descriptive formalism considered in this paper. In order to check whether all problems were detected by the application of our rules, the resulting models were transformed into a more detailed form for model checking using the ARCTIS tool [38]. No new problems were uncovered. Finally we applied the direct realization principle to derive a local behavior for each component as explained in [35] and [36].

In another case study the choreography of a city guide system was developed, see [36]. Here we studied the problem of localizing flows and control nodes (merges, choices, forks and joins) in more detail, as well as the mapping to the component behaviors. In any of these studies, we have not found any counter-example that would invalidate the propositions in this paper.

REFERENCES

[1] M. Abadi, L. Lamport, and P. Wolper, "Realizable and unrealizable specifications of reactive systems", *Proc. 16th Intl. Colloquium on Automata, Languages and Programming (ICALP'89)*, London, UK, Springer-Verlag, 1989, pp. 1–17.

[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] R. Alur and M. Yannakakis, "Model checking of message sequence charts", *Proc. 10th Intl. Conf. on Concurrency Theory (CONCUR'99)*, LNCS, vol. 1664, Springer, 1999, pp. 114–129.

[4] R. Alur, K. Etessami and M. Yannakakis, "Inference of Message Sequence Charts", *Proc. 22nd Intl. Conf. on Software Engineering (ICSE'00),* 2000.

[5] R. Alur, K. Etessami and M. Yannakakis, "Realizability and verification of MSC graphs", *Theoretical Computer Science*, 331(1), pp. 97–114, 2005.

[6] D. Amyot, "Introduction to the User Requirements Notation: learning by example", *Computer Networks*, vol. 42 (3), pp. 285-301, 2003.

[7] P. Baker, P. Bristow, C. Jervis, D. King, R. Thomson, B. Mitchell and S. Burton, "Detecting and resolving semantic pathologies in UML sequence diagrams", *Proc. 10th ESEC/13th ACM SIGSOFT FSE Conference*, New York, NY, USA, ACM Press, 2005, pp. 50–59.

[8] N. Baudru and R. Morin, "Safe implementability of regular Message Sequence Chart specifications", *Proc. ACIS 4th Intl. Conf. on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD'03)*, 2003, pp. 210–217

[9] H. Ben-Abdallah and S. Leue, "Syntactic detection of process divergence and non-local choice in Message Sequence Charts", *Proc. 2nd Intl. Workshop on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'97)*, 1997

[10] G. v. Bochmann, "Finite state description of communication protocols", *Computer Networks*, vol. 2, 1978, pp. 361-372.

[11] G. v. Bochmann and R. Gotzhein, "Deriving protocol specifications from service specifications", *Proc. ACM SIGCOMM Symposium*, 1986, pp. 148-156.

[12] G. v. Bochmann, "Deriving component designs from global requirements", *Proc. Intl. Workshop on Model Based Architecting and Construction of Embedded Systems (ACES)*, Toulouse, Sept. 2008.

[13] M. Bravetti and G. Zavattaro, "Contract Compliance and Choreography Conformance in the Presence of Message Queues", *Proc. of 7th Intl. Workshop on Web Services and Formal Methods*, LNCS 5387, Springer Verlag, 2009.

[14] R. Bræk, "Unified system modeling and implementation", *Proc. Intl. Switching Symposium (ISS)*, Paris, May, 1979.

[15] R. Bræk, "Using roles with types and objects for service development", *Proc. IFIP 5th Intl. Conf. on Intelligence in Networks (SMARTNET'99)*, IFIP Conference Proceedings, vol. 160, Kluwer, 1999.

[16] M. Broy, I. H. Krüger and M. Meisinger, "A formal model of services", *ACM Trans. on Software Engineering and Methodology (TOSEM)*, vol. 16, no. 1, February 2007.

[17] H. N. Castejón, "Collaborations in service engineering: Modeling, Analysis and Execution", PhD thesis, Norwegian University of Science and Technology (NTNU), 2008

[18] H. N. Castejón and R. Bræk, "A collaboration-based approach to service Specification and detection of implied scenarios", *Proc. 5th ICSE Intl. workshop on Scenarios and state machines: models, algorithms and tools (SCESM'06)*, ACM Press, 2006.

[19] H. N. Castejón and R. Bræk, "Formalizing collaboration goal sequences for service choreography", *Proc. 26th IFIP WG 6.1 Intl. Conf. on Formal Methods for Networked and Distributed Systems (FORTE'06)*, LNCS, vol. 4229, Springer-Verlag, 2006.

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

[21] H. N. Castejón, G. v. Bochmann and R. Bræk, "Direct realizability", Unpublished technical report, http://www.site.uottawa.ca/~bochmann/dsrg/PublicDocuments/Publications/Cast09.pdf, 2009.

[22] C.-A. Chen, S. Kalvala and J. Sinclair, "Race conditions in Message Sequence Charts", *Proc. 3rd Asian Symposium on Programming Languages and Systems (APLAS'05)*, LNCS, vol. 3780, Springer, 2005, pp. 195–211.

[23] G. Decker and M. Weske, "Local Enforceability in Interaction Petri Nets", *Proc. of 5th Intl. Conf. on Business Process Management*, Springer-Verlag, 2007.

[24] T. Erl, *Service oriented architecture: concepts, technology and design*, Prentice Hall, ISBN 0-13-185858-0

[25] K. Fisler and S. Krishnamurthi, "Modular verification of collaboration-based software designs", *Proc. 8th European Software Engineering Conference*, New York, ACM Press, 2001

[26] C. Fournet, T. Hoare, S. K. Rajamani, and J. Rehof, "Stuck-free Conformance", *Proc. 16th Intl. Conf. on Computer Aided Verification (CAV'04)*, LNCS, vol. 3114, Springer, 2004

[27] B. Genest, A. Muscholl, H. Seidl and M. Zeitoun, "Infinite-state high-level MSCs: Model-checking and realizability", *Journal of Computer and System Sciences.*, 72(4), 2006, pp. 617–647.

[28] M. G. Gouda and Y.-T. Yu, "Synthesis of communicating Finite State Machines with guaranteed progress", *IEEE Trans. on Communications*, vol. Com-32, no. 7, July 1984, pp. 779-788.

[29] Ø. Haugen, K.E. Husa, R.K. Runde, and K. Stølen, "Why Timed Sequence Diagrams Require Three-Event Semantics", Research report 309, University of Oslo, 2006.

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

[31] L. Hélouët, "Some pathological Message Sequence Charts, and how to detect them", *Proc. 10th Intl. SDL Forum*, LNCS, vol. 2078, Springer-Verlag, 2001, pp. 348–364.

[32] IUT-T, *Specification and Description Language (SDL)*, Recommendation Z.100, 2000.

[33] IUT-T, *Message Sequence Charts (MSC)*, Recommendation Z.120, 1998.

[34] C. Kant, T. Higashino and G. v. Bochmann, "Deriving protocol specifications from service specifications written in LOTOS", *Distributed Computing*, vol. 10, no. 1, 1996, pp.29-47.

[35] S.B.Kathayat, R. Bræk and H.N. Le, "Automatic Derivation of Components from Choreographies - A Case Study", *Proc. Of Annual Intl. Conf. on Software Engineering*, 2010.

[36] S.B.Kathayat and R. Bræk, "From Flow- Global Choreography to Component types", *Proc. of 7th Workshop on System Analysis and Modeling*, LNCS 6598, Srpinger –Verlag, 2011.

[37] F. Khendek and X. J. Zhang, "From MSC to SDL: Overview and an application to the autonomous shuttle transport system", *Proc. 2003 Dagstuhl Workshop on Scenarios: Models, Transformations and Tools*, LNCS, vol. 3466, 2005.

[38] F.A. Kraemer, V. Sâtten and P. Herrmann, "Tool support for the rapid composition, analysis and implementation of reactive services", *Journal of Systems and Software*, 82(12):2068-2080, 2009.

[39] I. Krüger, "Capturing overlapping, triggered and preemptive collaborations using MSCs", *Proc. 6th Intl. Conf. on Fundamental Approaches to Software Engineering (FASE'03)*, LNCS, vol. 2621, Springer, 2003.

[40] I. Krüger and R. Mathew, "Component synthesis from service specifications", *Proc. Intl. Dagstuhl Workshop on Scenarios: Models, Transformations and Tools*, LNCS, vol. 3466, Springer, 2003.

[41] F. Laamarti, "Derivation of component designs from global specifications", Master Thesis, SITE, University of Ottawa, 2010. Available at: http://www.site.uottawa.ca/~bochmann/dsrg/Docs/Master-theses/Laamarti%20-%20Derivation-of-Component-Designs-from-Global-Specifications.pdf.

[42] N. Lohmann and K. Wolf, "Realizability is Controllability", *Proc. of 6th Intl. Conf. on Web Services and Formal Methods*, Springer-Verlag, 2010.

[43] M. Lohrey, "Realizability of high-level message sequence charts: closing the gaps", *Theoretical Computer Science*, 309(1-3), 2003, pp. 529–554.

[44] J. Magee and J. Kramer, Concurrency: State Models & Java Programs, 2nd Edition, Wiley, 2006.

[45] B. Mitchell, "Resolving race conditions in asynchronous partial order scenarios", *IEEE Trans. on Software Engineering.*, 31(9), 2005, pp. 767–784.

[46] A. J. Mooij, N. Goga and J. Romijn, "Non-local choice and beyond: Intricacies of MSC choice nodes", *Proc. Intl. Conf. on Fundamental Approaches to Software Engineering (FASE'05)*, LNCS, 3442, Springer, 2005.

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

[48] A. Mousavi et al., "Strong safe realizability of message sequence chart specifications", *Proc. Intl. Symp. on Fundamentals of Software Engineering, Springer*, LNCS 4767, 2007

[49] M. Mukund, K. N. Kumar and M. A. Sohoni, "Synthesizing distributed finite-state systems from MSCs", *Proc. 11th Intl. Conf. on Concurrency Theory (CONCUR'00)*, LNCS, vol. 1877, Springer, 2000, pp. 521–535.

[50] Object Management Group, *UML 2.2 Superstructure Specification*, http://www.omg.org/cgi-bin/doc?formal/09-02-02.pdf, February 2009

[51] Object Management Group, *UML 2.0 Superstructure Specification*, http://www.omg.org/spec/UML/2.0/Superstructure/PDF/, July 2005

[52] Z. Qiu, X. Zhao, C. Cai and H. Yang, "Towards the Theoretical Foundation of Choreography", *Proc. of 16th Intl. Conf. on World Wide Web*, 2007

[53] T. Reenskaug, E.P. Andersen, A.J. Berre, A. Hurlen, A. Landmark, O.A. Lehne, E. Nordhagen, E. Ness-Ulseth, G. Oftedal, A.L. Skaar and P. Stenslet, "OORASS: Seamless support for the creation and maintenance of object-oriented systems", *Journal of Object-oriented Programming*, 5(6), 1992, pp. 27-41.

[54] T. Reenskaug, P. Wold and O.A. Lehne, *Working with objects: The OOram software engineering method*, Prentice Hall, 1995.

[55] N. Roohi and G. Salaün, "Realizability and Dynamic Reconfiguration of Chor Specifications", *Informatica*, 2011, in press.

[56] S. Uchitel, J. Kramer and J. Magee, "Incremental elaboration of scenario-based specifications and behavior models using implied scenarios", *ACM Trans. on Software Engineering and Methodology (TOSEM)*, 13(1), 2004, pp. 37–85.