Using Collaborations in the Development of Distributed Services[1]

Humberto Nicolás Castejón, Gregor von Bochmann, *Fellow, IEEE,* and Rolv Bræk

***Abstract*-- This paper is concerned with the compositional specification of services using UML 2 collaborations, activity and interaction diagrams. It addresses the problem of realizability: given a global specification, can we construct a set of communicating system components whose joint behavior is precisely the specified global behavior? We approach the problem by looking at how the sequencing of sub-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. Possible solutions to the realizability problems 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 sub-collaborations, provided that we know the components that initiate and terminate the different sub-collaborations.**

Index Terms: 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

FOR several decades now it has been common practice to specify and design reactive systems in terms of loosely coupled components modeled as communicating state machines [11], [16], using languages such as SDL [31] and more recently UML [49]. 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 and suitable for formal analysis as well as automatic generation of executable code.

However, there is a fundamental problem. In many cases, the behavior of services provided by a system is not performed by a single component, but by several collaborating components. This

Manuscript received February 6, 2008.

H. N. Castejón is with the Department of Telematics, Norwegian University of Science and Technology, Norway (e-mail: humberto.castejon@item.ntnu.no).

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.

has been recognized by several authors, such as [17], [39] and [19], and is sometimes referred to as the “crosscutting” nature of services [25], [40]. Often each component takes part in several different services, so in general, the behavior of services is composed from partial component behaviors, while component behaviors are composed from partial service behaviors. By structuring according to components, the behavior of each individual component can be defined precisely and completely, while the behavior of a service is fragmented. In order to model the global behavior of a service more explicitly one needs an orthogonal view where the collaborative behavior is in focus.

Interaction sequences such as MSC [32], and UML Sequence diagrams [49] are commonly used to describe collaborative behavior, and have proven to be very valuable. They are however, not without limitations. They are expressed in terms of message exchanges, which at an early stage of development may be too detailed. 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. In addition, there are problems related to the realizability of interaction scenarios, i.e. finding a set of local component behaviors whose joint execution leads precisely to the global behavior specified in the scenarios. Some authors have studied the realizability problem in the context of implied scenarios [5], [57], i.e. unspecified scenarios that will be generated by any set of components implementing the specified scenarios. Other authors have studied pathologies in interaction sequences, e.g. non-local choices [10], that prevent their realization.

We have asked ourselves if there are better ways to model services. Is it possible to model service behavior more completely? Can it be done in a more structured way without revealing more interaction detail than necessary? Is it possible to support composition and to detect and

remove realization problems? And is it possible to derive detailed implementations automatically from service models?

We have found that a promising step forward is to adopt a collaboration-oriented approach, where the main structuring units are collaborations. This is made practically possible by the new UML 2 collaboration concept [49]. The underlying ideas, however, date back to before the UML era [50], [51]. Collaborations model the concept of a service very nicely. They define a structure of partial object behaviors, called roles, and enable a precise definition of the service behavior using interaction diagrams, activity diagrams and state machines as explained in [55], [21] and [37]. They also provide a way to compose services by means of collaboration uses and to bind roles to components. In this way, UML 2 collaborations directly support (crosscutting) service modeling and service composition. As we shall see in the following, this opens many interesting opportunities. Fig. 1 illustrates the main models involved in the collaboration oriented approach being discussed in the following:

- Service models are used to formally specify and document services. Collaborations provide a structural framework for these models that can embody both the role behaviors and the interactions between the roles needed to fulfill the service.
- Design models are used to formally specify and document system structure and components providing the services. They are expressed in terms of communicating state machines, typically using UML2 active objects or SDL agents (processes). Each of these will realize one or more collaboration roles.
- Implementations are executable code automatically generated from the design models.

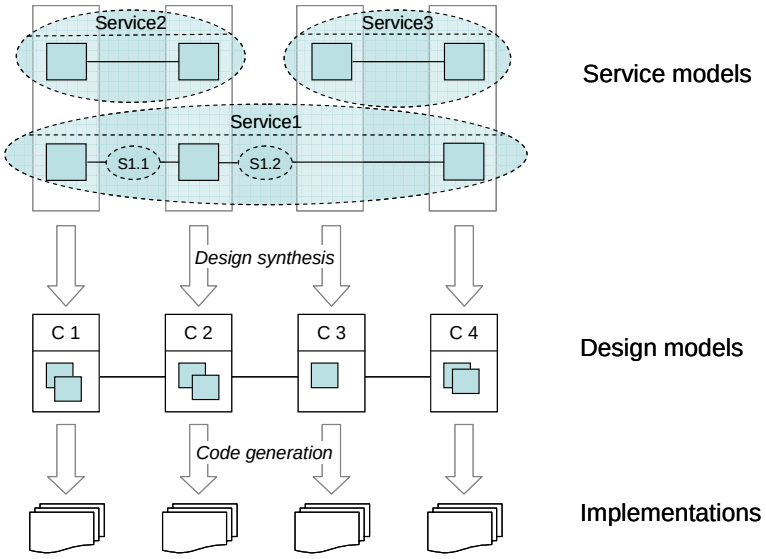

Fig. 1. Collaboration oriented development

This paper is concerned with the crucial first steps of expressing service models using UML 2 collaborations and deriving well-formed design models expressed as communicating state machines. The ensuing steps from design component models to implementations and dynamic deployment on service platforms can be solved in different ways, see for instance [53] and [18], and are not discussed further here.

An important property of collaborations is that it is possible and convenient to compose/decompose collaborations structurally into sub-collaborations, by means of collaboration uses. These refer to separately defined collaborations and provide a mechanism for abstraction and collaboration reuse. In order to define the behavior of collaborations, we have found it useful to distinguish between the behavior of *composite collaborations* and *elementary collaborations* (collaborations that are not further decomposed into sub-collaborations). The elementary collaborations that result from the decomposition process are often quite simple, reusable and possible to define completely in terms of interaction sequences. Binary collaborations can in many cases be associated with interfaces and their sub-collaborations with

phases or features of the interface behavior. Assuming the behavior of elementary collaborations are completely defined using interaction diagrams or other notations, the question then is how to define the overall behavior of composite collaborations in terms of sub-collaboration behaviors. In the SOA domain this kind of behavior is called “choreography” [24], a term we will use in the following. Several notations may be used to define the choreography of sub-collaborations (i.e. their global execution ordering). We have found UML2 Activity diagrams a good candidate, as they provide many of the composition operators needed for the purpose. This will be elaborated in Section 2.

Interestingly, the operations needed to define a choreography also enable us to identify and classify the underlying reasons leading to realization problems. Many of these can be found by analyzing choreographies at the level of its sub-collaborations without needing to go into interaction details. When this is not possible, potential problem spots can be pinpointed so that detailed interaction analysis can be focused on those. In Section 3 we present our results in this area.

In Section 4 we discuss the feasibility of automatically synthesizing components from collaborations, i.e. to automate the step from service models to design models. We foresee a process where choreographies defined using activity diagrams are used directly. This points towards a highly automated process form collaboration oriented service models to executable services.

2 USING COLLABORATIONS TO MODEL SERVICES

2.1 A case study: TeleConsultation

We consider as an example a telemedicine consultation service. 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. Since the patient may stay at home and the hospital is a considerable distance away from the patient's home, the patient has been equipped with the necessary testing equipment at home. The patient will call the hospital on a regular basis to have remote tests done and consult with a doctor. A consultation may proceed as follows:

1. The patient calls the telemedicine reception desk to ask for a consultation session with one of the doctors. The receptionist will register the information needed, and then see if any of the appropriate doctors is available.
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 and may use the supplied information to look up the patient journal.
4. A voice connection is established between the patient and the doctor allowing the consultation to take place.
5. During the consultation the doctor may decide to perform some remote tests using the equipment located at the patient's site. The doctor evaluates the results and advises the patient about the 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.

Each of these points may be considered an activity. An UML activity diagram describing the order of execution of these activities is given in Fig. 2. The fact that the patient and the doctor behave concurrently and may take initiatives independently of each other is reflected by the use of two initial nodes. The result is a diagram with two concurrent parts that are joined for the assignment and consultation activities.

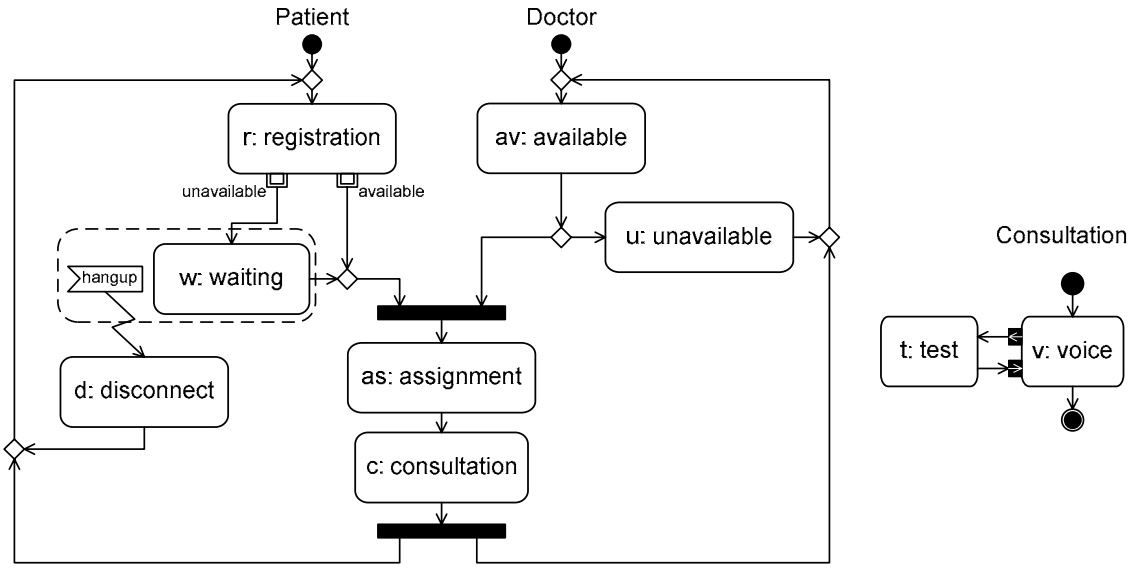

Fig. 2 Activity diagrams describing a TeleConsultation.

2.2 Collaboration structure

An important aspect in requirements specification and service modeling, as well as in workflow modeling, is the identification of the actors involved in the different activities.

Traditional UML use cases distinguish the *system* and actors that are part of the system environment. For high-level workflow modeling and domain analysis, one usually identifies various actors that participate without necessarily identifying a system. For the TeleConsultation example, we can identify the following actors: the patient, the doctor and the receptionist. It should be noted here that it is useful to make a distinction between real actors and the roles that actors play. In specifications and designs it does not make much sense to identify particular actors. One rather identifies roles that may be composed and bound to real actors in different

ways. As illustrated in Fig. 2, roles (patient and doctor) may have different responsibilities and follow partly independent workflows with some joint activities.

A UML collaboration diagram is well suited to model role structures and identify sub-collaborations among the roles. This is illustrated for the TeleConsultation example in Fig. 3, where 4 roles and 7 sub-collaborations are identified. These sub-collaborations are modeled as UML collaboration uses, and their roles are bound to the roles of TeleConsultation[2] When decomposing collaborations into sub-collaborations one tends to identify sub-collaborations that involve just two roles. Such *binary* collaborations can be used to define interfaces and interface behavior. This has some advantages: (1) the behaviors are relatively small, (2) they can be completely defined, and (3) they are units of reuse. In this example, all sub-collaborations except *consultation* are elementary and associated with an interface between two roles.

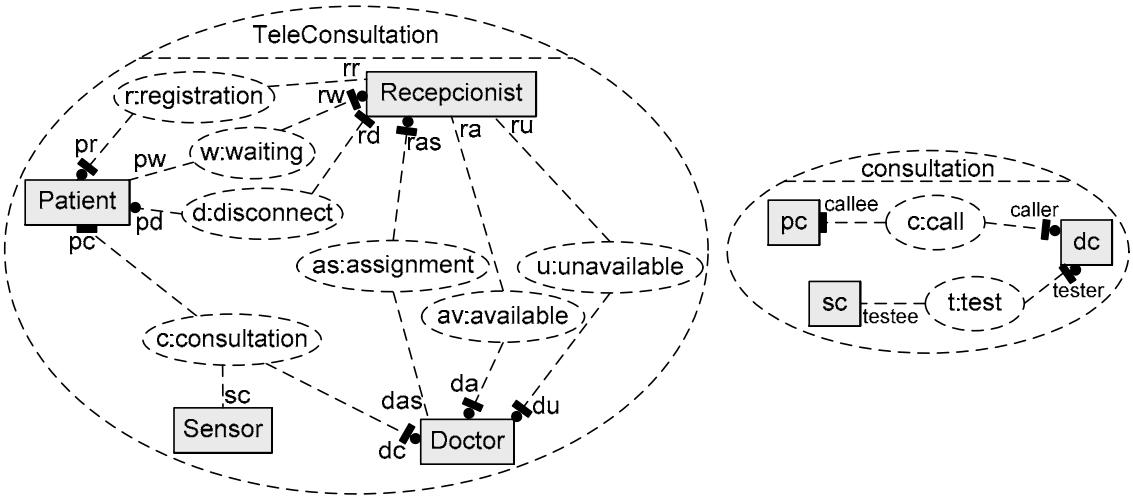

Fig. 3 Roles and sub-collaborations in the hospital visit.

As explained in [54] and [56] binary collaborations can be used to define semantic interfaces that are used to type components in order to enable efficient discovery and compatibility checking at design time and runtime. Used in this way, collaborations are useful during the entire

[2] The “dots” and “bars” in this diagram are not standard UML. They are used to indicate the initiating and terminating roles of a collaboration use, as will be explained in Section 2.4

life cycle, not only in early service modeling.

In service and workflow modeling one often distinguishes between actors and resources. Both may be associated with a given action and required for its correct execution. The main difference between these two entities is normally that actors may take initiatives, while resources are rather passive. For the modeling of collaborations, we consider both actors and resources as roles that participate in the execution of an action. In the TeleConsultation the sensors are rather passive resources. The doctors may be seen as shared resources from the point of view of patients, and the receptionist as a resource allocator. Contrary to sensors, the patients and the doctors can take independent initiatives, and this exemplifies that resources may also be active.

2.3 Collaboration behavior: Choreography

The activity diagram in Fig. 2 defines the global behavior for the TeleConsultation collaboration. The activities have been chosen so that each activity corresponds to a sub-collaboration in Fig. 3, and in this way it defines their choreography. Note how this diagram defines collaboration ordering in a visual way without going into the details of interactions.

UML Activity diagrams appear to be at a suitable level of abstraction for defining the choreography of sub-collaborations within a given composite collaboration. They can express sequential, alternative and concurrent behavior including the possibility of looping, as well as interruption and activity invocation (e.g. *voice* invoking *test* in the *consultation* collaboration), as illustrated in Fig. 2. The units of execution are atomic actions or activities. An activity can be further refined and described by a separate activity diagram that will be invoked when the activity becomes activated.

There is, however, one important feature of collaborations that is different from what is normally assumed of an activity. A collaboration involves several roles (participants). In activity

diagrams, each activity normally involves only a single role. It must therefore be foreseen that an activity or action, representing a sub-collaboration within an activity diagram, involves more than one role. This information can be provided by annotating the diagram as shown in Fig. 4. UML is very open-ended concerning the precise notation for activity diagrams and allows this kind of notational extension. The important point here is the kind of information we would like to include in the models, not exactly how it is done. The same information might be supplied in different ways, as will be discussed in Section 2.5.

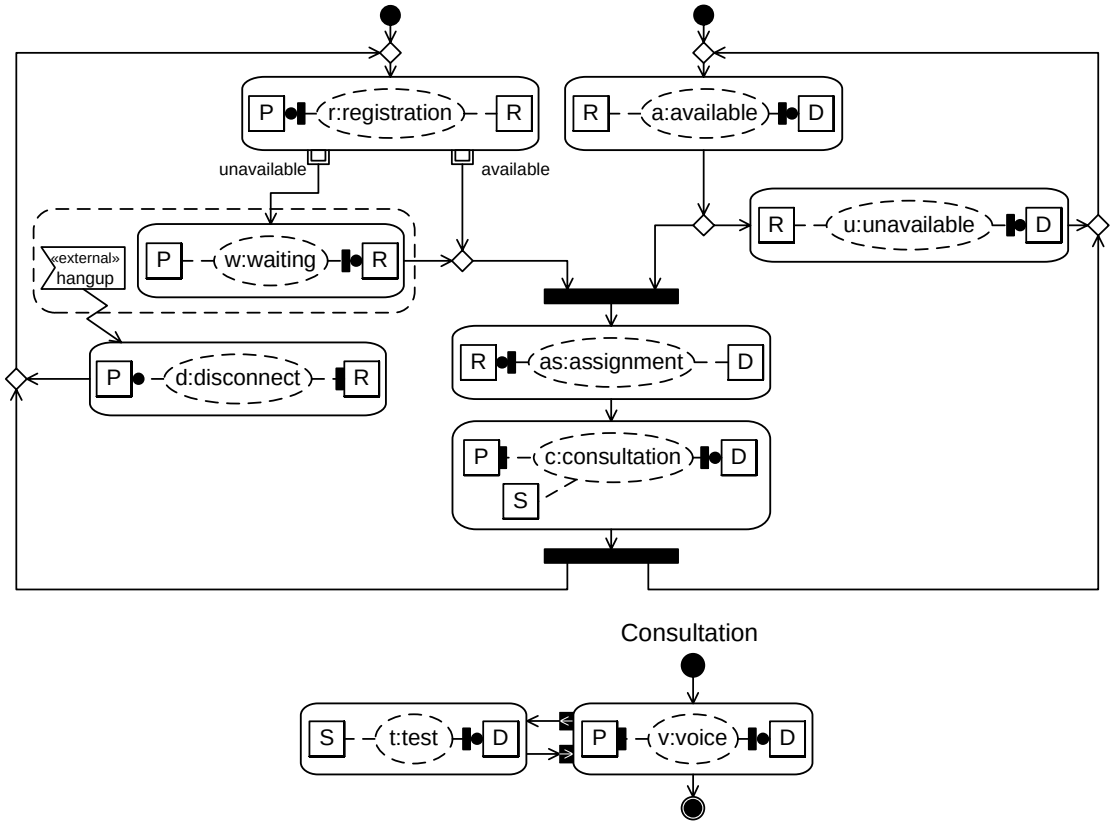

Fig. 4 Choreography for the TeleConsultation collaboration.

2.4 The nature of collaborations

A collaboration describes (joint) actions among a set of roles carried out in order to achieve a goal. The roles describe the behavior and properties that components should exhibit in order to participate in the collaboration. The order in which the actions are performed is enforced locally

by the participants and globally by the exchange of messages between the participants. This is in general a partial order, as first explained by Lamport [41]. A collaboration may include behaviour alternatives leading to different outcomes, as for instance the registration collaboration in the TeleConsultation, see Fig. 4.

For each collaboration, we distinguish the *initiating actions* and the *terminating actions*. The initiating actions are those actions for which there is no earlier action in the collaboration according to the (partial) execution order defined for the collaboration. Similarly, the terminating actions are those for which there is no later action in the collaboration. The roles involved in the execution of an initiating (resp. terminating) action of a collaboration are called *initiating* (resp. *terminating*) *roles*. In Fig. 3 and Fig. 4 the initiating roles have been identified by a dot and the terminating roles by a bar. Which roles initiate and/or terminate the execution of a sub-collaboration is very important for the coordination of the temporal order of execution of different sub-collaborations, as discussed in more detail in Section 3.

How the roles of sub-collaborations are bound to roles of enclosing collaborations and eventually to components of a distributed system design is important for the realizability of a specified ordering. Collaboration structures such as Fig. 3 specify precisely the binding of **sub-roles** (i.e. roles of sub-collaborations) to the **composite-roles** of the collaboration structure. The diagram in Fig. 3, for instance, specifies that the sub-role *pr* from *registration* is bound to the composite-role *Patient* of TeleConsultation. We will say that a composite-role is the initiating (resp. terminating) role of a sub-collaboration if it is bound to the initiating (resp. terminating) role of that sub-collaboration.

In the case of a sub-collaboration that has several terminating roles, the terminating actions performed by these different roles will normally not be synchronized. This is in contradiction to

the semantics of UML activity diagrams which states that all the outputs of an activity will become available at the same time (when the activity terminates)[3].

We note that a collaboration with more than one initiating role may be not so easy to realize because of the required coordination between the initiating roles for initiating the collaboration. As a general design guideline, it is therefore desirable to avoid collaborations with multiple initiating roles as far as possible. Related issues are further discussed in Section 3. Here are some examples:

a. Single initiating role: the patient (initiating role) registers with the receptionist.
b. Several alternative initiating roles: each side of a consultation call may take the initiative to perform the call termination sub-collaboration.
c. No initiating role identified: it is not specified whether the doctor or the sensor (triggered by the patient) initiates a test.
d. Several terminating roles: in the voice collaboration the last action may be performed by the patient or the doctor, or both.

At a high-level of abstraction, we may characterize a collaboration by a pre-condition and a post-condition. A collaboration can only be initiated if its pre-condition is satisfied; we say that the collaboration is *enabled*. The pre-condition describes the required system state for the collaboration to start. As a design guideline, we note that it is desirable that the enabling condition only depends on the state of the initiating roles. The post-condition represents a condition that will be true when the collaboration terminates; if the collaboration admits several alternative outcomes, the post-condition will be the logical OR between these alternatives. One

[3] In UML, outputs of an action are identified by so-called pins, outputs of an activity are identified by so-called parameters of the activity. There are two exceptions to the rule that all outputs occur when the activity terminates: (a) Several alternate sets of outputs may be identified (only one of these sets of outputs will occur), and (b) so-called stream inputs and outputs may occur anytime during the execution of an activity.

may also specify “goals” in terms of states or events where the purpose of a collaboration is achieved, see [54], when different from the post-condition. In the TeleConsultation example pre- and post-conditions have not been illustrated. However, we can imagine that there would be an *available* predicate reflecting the availability of the doctor. This predicate should for example be true for *assignment* to be enabled, and would become false when this collaboration finishes.

Sometimes it is also useful to identify a *triggering event* for a collaboration. This is an event that leads to the execution of the collaboration if its precondition is true when the triggering event occurs. In the case of sequential execution of two sub-collaborations, the triggering event of the second collaboration would normally be the termination of a terminating action of the first collaboration. In other cases, the triggering event may be the reception of an external input or a time-out that is not part of the collaboration being modeled. Such *external triggering events* may cause a role to initiate a collaboration seemingly spontaneously and on its own initiative. External triggering events are normally not specified explicitly, only the *initiatives*, i.e. the seemingly spontaneous actions they cause. It is important to identify such initiatives in service modeling for two main reasons: (1) most services and service features are initiated by external initiatives; (2) they give rise to concurrency and potential conflicts. Initiatives 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 patient and the doctor take independent initiatives leading to the parallel paths in Fig. 2 and Fig. 4. The two paths may be considered as different views on the service; the patient view and the doctor view. These are brought together and coordinated during the assignment and consultation collaborations. 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.

2.5 Notation

We make in the following some comments about possible notations for representing the choreography of collaborations as described above. The objective is not so much to find a notation as to identify concepts that allow specifying and analyzing the high level flow without prematurely binding the detailed interactions, and that also allow gradual detailing towards interactions and localized behavior.

In Section 2.3 we have already discussed the suitability of UML activity diagrams for defining the choreography of sub-collaborations. They allow a compositional specification approach and can express sequential, alternative and concurrent behavior, as well as interruption and activity invocation.

An important aspect of a choreography is to show which roles participate in which collaborations and whether they are initiating or terminating roles. A possible graphical notation for representing multiple roles involved in a single activity was proposed in [15] where resources and roles are represented as separate entities and their involvement in activities by a special type of arrow (see Fig. 5 (a)). A variant of this is to let activities and roles overlap as illustrated in Fig. 5 (b). This style can visualize the ordering of sub-roles within a role, and can also be used to localize control flows to particular roles when this is desirable. In the variant shown in Fig. 5 (c), this is taken one step further by representing the global choreography as one enclosing activity with partitions corresponding to the roles, and sub-collaborations modeled as activities that cross-cuts the partitions. All flow lines are here local to roles and specify precisely how role behaviors are composed. This approach has been used in several case studies, and to demonstrate

that design components defined as state machines can be derived automatically [38]. The UML specification suggests representing the partitions textually as illustrated in Fig. 5 (d). The notation used in Fig. 4 is a visual enhancement of this showing the roles graphically as well as the active collaboration use. It is an extension of the notation originally proposed in [20] and [21]. It has the advantage that each activity has a clear boundary, which helps to organize larger diagrams. We will use the simplified form shown in Fig. 5 (e), later on for illustration purposes.

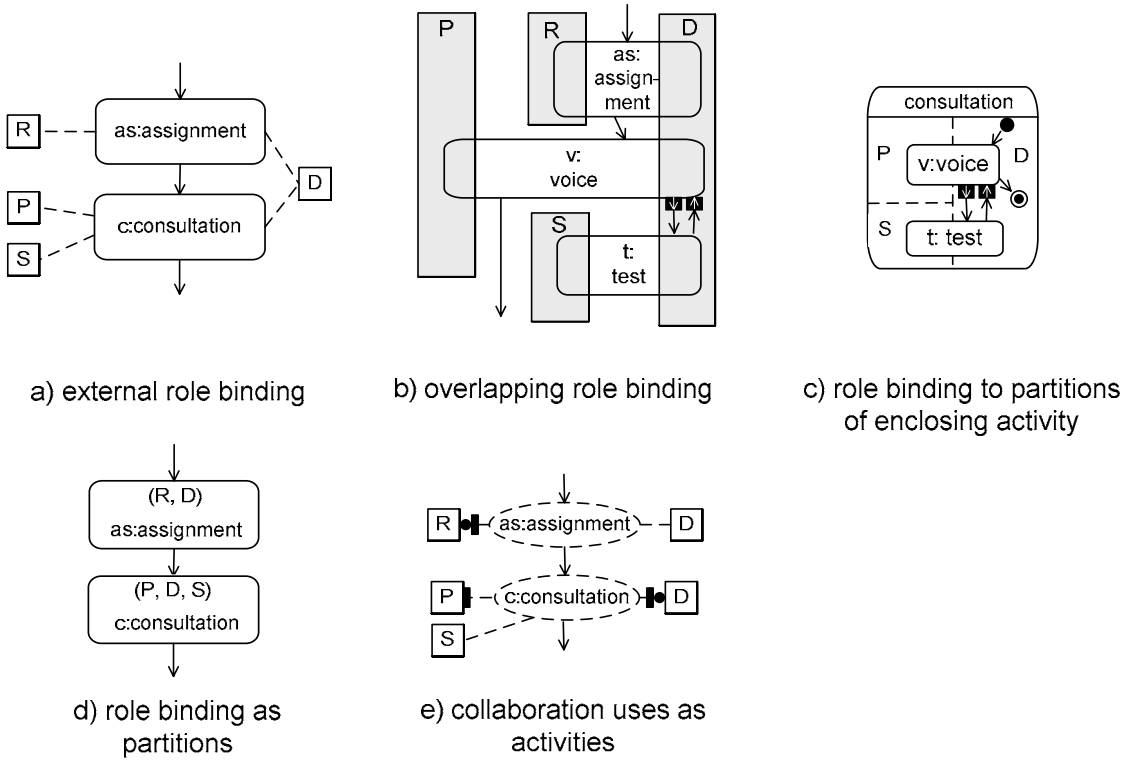

Fig. 5 Alternative notations for role binding.

Note that the variants in Fig. 5 (a) to Fig. 5 (d) are formed simply by varying the representation and localization of roles relative to activity boundaries. The variants in Fig. 5 (b) and Fig. 5 (c) allow localizing control flows to roles. Semantically the only difference between these variants is the localization of control flow. Note that the variant in Fig. 5 (b) resembles a sequence diagram, where messages are replaced by sub-collaborations. The variants in Fig. 4 and Fig. 5 (e) have an explicit reference to collaboration uses, while in the other variants this is maintained by naming

conventions.

High-Level Message Sequence Charts (HMSC) [32] or UML Interaction Overview Diagrams (IOD) are other notations that also have a suitable level of abstraction for choreography, but they are tied to interactions and do not readily allow the same flexibility to combine notations. Moreover, they lack certain operators (e.g. preemption) and semantics needed to fully define collaborative behavior as also pointed out in [39].

A textual notation might also be used to define the temporal ordering of actions and sub-collaborations within a given collaboration. In previous work [13], [34] a notation based on process algebras was used. If one uses Use Case Maps [6] to model collaborations, the concept of sub-collaboration could be modeled as a "stub", and a participant as a "component"; however, also here it is assumed that each action (called "responsibility") is associated with at most one "component".

To specify the behavior of elementary collaborations there are several options. Activity diagrams may be used to define the behavior in a way that allow component behaviors to be derived automatically [38]. If one chooses to define elementary collaborations using interaction diagrams, these may be referenced from the activities of a choreography [20]. One may also refer to collaborations representing semantic interfaces with goals and behavior defined by two role state machines [54]. Alternative approaches of combining sequence diagrams with other diagrams have been proposed by several authors, e.g. [52] and [58].

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.

3 ORDERING OPERATORS FOR CHOREOGRAPHY

In this section we discuss the sequencing operators that we consider important for describing the execution order of collaborations. These are the standard concepts of sequential execution, alternatives, concurrency, and interruption which are supported by most notations for workflow modeling, including UML Activity Diagrams and Use Case Maps. We discuss in the following some particular semantic features for these concepts which are not provided by the standard semantics of Activity Diagrams. We also introduce the concept of activity invocation, a variant of a remote procedure call. We note that some of these features are also discussed in Wohed's analysis of the control-flow perspective of UML Activity Diagrams [59].

3.1 Realization problems

We consider in this paper that the model of a distributed service is defined by a composition of several sub-collaborations, as discussed in Section 2. The service model defines a global order about different actions that will be performed by different service roles. At the system design level, as shown in Fig. 1, the roles of the collaborations are assigned to certain system components and their local behavior may be defined by state machine models. One may attempt to obtain the specification for the behavior of a given system component by projecting the global service behavior specification onto that component, that is, ignoring all actions at the other components in the service model and deriving the order of the local actions at the given component from the ordering of these actions in the service model. We say that a design model is **directly realized** from a given service model, if the behavior of each system component of the

design is obtained by projecting the service model onto the given system component. If the behavior of the directly realized design model is equivalent to the overall system behavior defined by the service model, we say that the service model is **directly realizable**.

Unfortunately, in some cases the directly realized design will generate interaction scenarios not foreseen by the service model. 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 behaviour of a distributed system is described from a global perspective, while it is realized by independent components with only local knowledge.

We will discuss in this section under which circumstances a choreography is directly realizable. We will discuss for each composition operator 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 service model. 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 if the problems are *actual*, i.e. actually exist.

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:*

1. *No reordering*: Each component has a single FIFO buffer in which all received messages are stored until they are processed. Messages are consumed in a FIFO order.
2. *Reorder 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.
3. *Full reorder*: A component may reorder received messages freely.

In the following we assume that each sub-collaboration of a choreography is directly realizable and discuss for each ordering operator different conditions for the direct realizability of the choreography.

3.2 Sequence

According to the semantics of UML activity diagrams, each activity is completely finished before the next one starts. In the most general case, this requires global coordination between all system components participating in the two activities. In many cases, such *strong sequencing* is what we want for collaborations too, but sometimes strong sequencing is too restrictive and may be replaced by *weak sequencing* where the sequential order is only enforced locally on each component without global coordination.

It is therefore necessary to allow weak sequencing as well as strong sequencing and to provide some notation for distinguishing between them. We can annotate sequential composition with a constraint of the form "{weak}" and "{strong}" for this purpose. By default we assume weak

sequencing and only annotate edges in case of strong sequencing. Note that weak sequencing is the semantics defined for sequential execution in High-Level Message Sequence Charts and UML interaction overview diagrams.

Strong and weak sequential execution of sub-activities impose ordering constraints on the collaborations and may lead to various realization problems which are further discussed below.

1.2.3 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 action(s) of C_1 and the initiating action(s) of C_2, so that the latter can only happen after the former are finished. The situation is particularly simple in the case of a localized sequence composition as defined below.

Definition 1 (localized sequence composition). *A sequence composition* $C_1 \circ C_2$ *is a* localized sequence composition *if all terminating actions of* C_1 *and all initiating actions of* C_2 *are located at the same composite role.*

In the case of a localized sequence composition, there is no semantic difference between strong and weak sequencing. In this case, the initiator of C_2 can know when C_1 is completely finished. We have the following proposition.

Proposition 1. *A localized sequence composition of two directly realizable collaborations is directly realizable.*

Note that this property can be checked at the choreography level, i.e. by considering the initiating and terminating roles, without considering detailed interactions. In Fig. 4, for example,

the condition is not satisfied anywhere so there is no localized sequence composition in the diagram.

If the condition is not satisfied and strong sequencing is required, coordination messages must be added from C_1's terminating composite-roles to C_2's initiating-composite roles. This could be done automatically by a synthesis algorithm [13].

2.2.3 Weak Sequence

Weak sequencing of two sub-collaborations C_1 and C_2, written $C_1 \circ_w C_2$, basically requires each composite role in C_2 to be completely finished with previous collaborations 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 MSC and UML Interaction Diagrams.

Weak sequencing introduces concurrency, since the actions of the composed collaborations may partially overlap. Although such concurrency may be desirable for performance or timing reasons (i.e. a role may initiate a new collaboration if the actions in that collaboration are independent of the actions that have yet to be executed in the first collaboration), it comes at a price, since it may lead to specifications that are counter-intuitive and/or not directly realizable, as illustrated in Fig. 6.

According to the basic weak sequence semantics, role *B* in Fig. 6 (a) may initiate collaboration C_3 as soon as it has finished with C_1. As a result, collaborations C_2 and C_3 may be executed in any order in the realized system. This is counter-intuitive to the specification, which we assume reflects the designer's intention (i.e. that C_3 should be executed after C_2, with some allowed overlapping). If the designer's intention was that the collaborations should be concurrently executed, this should be explicitly specified by means of parallel composition. Note that in this case there are no problems with realizability since the roles of C_2 are completely disjoint from the

roles of C_3 and may execute independently. A tool should nonetheless issue a warning that the collaborations may not behave as intended and suggest replacing the sequence with a parallel composition. Note that the composition $C_2 \circ_w C_3$ has two initiating roles, i.e. *A* and *B*, that may be executed concurrently. As a guideline such initial concurrency should be avoided in order to ensure a minimal amount of causality between initiatives.

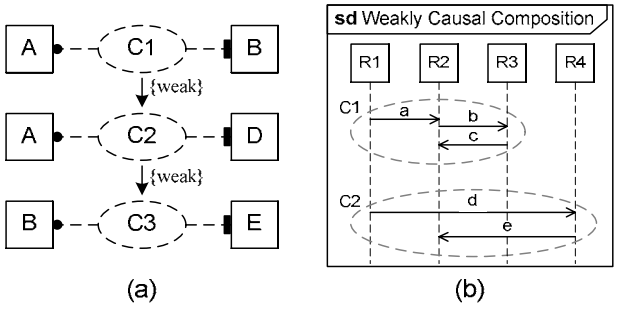

Fig. 6 Problematic weak sequential compositions

Definition 2 (weak-causality). *A weak sequential composition of two collaborations,* $C_1 \circ_w C_2$, *is* weakly-causal *if* C_2 *has a single initiating composite role and this role participates in* C_1.

The weak-causality property ensures that the initial actions of C_1 and C_2 are ordered sequentially. This can be checked at the collaboration level. We note that weak-causality is enforced in the so-called local-HMSCs of [26].

Consider the weak sequential composition of C_1 and C_2 in Fig. 6 (b). This composition is weakly causal, but not directly realizable. Component *R1* may initiate collaboration C_2 just after sending message *a* in C_1. Therefore, the actions in C_2 may overlap with the actions in C_1 that follow the sending of message *a.* For example, message *e* may be received at *R2* before message *c*, or even before message *a*. This message reception order has not been explicitly specified, and therefore it is an implied scenario. Note that such problems may only occur when a composite role (here *R2*) participates in both C_1 and C_2 and plays a non-initiating sub-role in C_2.

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

This property can be easily checked at the choreography level and represents a situation where weak sequencing is unproblematic. In the opposite case, where a non-initiating role in C_2 also participates in C_1, there is a *potential race* condition.

In the literature about MSCs, the possibility that messages may be received in a different order than the one specified 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 receiving event or a sending event, and both events are located at 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 pair of roles, but do not prevent races when more than two roles are involved. This is the case for the situation in Fig. 6 (b). Such races may in general be resolved by means of input buffering that can reorder between sources (unless choices are involved, as we will see in Section 3.2.3).

We note that race conditions may not only appear between *directly* composed collaborations, but also between *indirectly* composed ones. This is because a role in a collaboration that is composed in weak sequence can remain active during several succeeding collaboration steps. For example, in the TeleConsultation service a race condition exists between *registration* and *consultation* at composite role *P* (see Fig. 4). In this case it is the weak sequencing between

registration and *assignment* that makes such race possible, since the sub-role played by *P* in *registration* may still be active (i.e. not finished) while *assignment* is executed and when *consultation* is initiated. We therefore say that there is **indirect weak sequencing** between *registration* and *consultation*. This "propagation" of weak sequencing makes it more difficult to avoid races.

A 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 composition)[4]**.** A composition $C_1 \circ_w C_2$ *is* send-causal *if the composite role initiating* C_2 *is either the terminating role of* C_1 *or the role that performs the last sending event of* C_1.

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

1) *it is a single message transmission, or*
2) *it is formed by, possibly repeated, send-causal compositions* $C = C_1 \circ_w C_2$ *where* C_1 *and* C_2 *are send causal.*

It has been shown in [22] that when send-causality is enforced, races may only occur between two or more consecutive receiving events (i.e. not between a sending event and a receiving event).

Proposition 3. *In a send-causal collaboration, race conditions may only exist between two or more consecutive receiving events.*

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

If $C = C_1 \circ_w C_2$ is send-causal a *potential race condition* exists on a composite role R in C if the sub-role that R plays in C_1 ends with a message reception (i.e. is a terminating role) and the sub-role R plays in C_2 starts with another message reception (i.e. is a non-initiating role). 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 components. For example, in the TeleConsultation service the collaborations *available* and *assignment* are composed in weak sequence (see Fig. 4). Role *D* plays a terminating sub-role in *available*, while it plays a non-initiating sub-role in *assignment*. Therefore, a potential race condition exists at *D* between the receptions of the last message in *available* and the first message in *assignment*. This race is only *actual* 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 Fig. 4.

Proposition 4. *A send-causal weak sequential composition of a sequence of directly-realizable collaborations* $C = C_1 \circ_w C_2 \ldots \circ_w C_n$ *(n>1) is directly realizable*

- *over a communication service with* in-order delivery *if the following condition is satisfied: if a composite role plays a terminating role in a collaboration* C_i *(*$1 \leq i < n$*) followed by a non-initiating role in another collaboration* C_j *(*$i < j \leq n$*), then the last message it receives in* C_i *and the first one it receives in* C_j *are sent by the same peer-composite role; or*
- *over a communication service with* out-of-order delivery *only if no composite role plays a terminating sub-role followed by a non-initiating sub-role.*

For binary sub-collaborations we can easily identify which composite role sends the first and last messages, if we know which composite roles are the initiating and terminating roles. Using

Proposition 4, we can determine whether a collaboration is directly realizable and identify actual races at the choreography level without considering the detailed interactions. In the case of n-ary collaborations, we can perform the same early analysis, but only potential races can be discovered.

One interesting aspect of the specification with collaborations is that we can get information about potential races from the diagram describing the structural composition of collaborations, see Fig. 3. In such diagrams we can see whether a component participates in several collaborations, and whether it plays at least one terminating and one non-initiating role in them. If that is the case, a potential race exists. This information could then be used to direct the analysis of the behavioral specification (i.e. the choreography). For example, from the collaboration diagram for *TeleConsultation* (see Fig. 3 (a)) we can see that *Patient* participates both in *registration* and *consultation*, playing a terminating role in the first one and a non-initiating role in the second one. From this information we can conclude that a potential race exists at *Patient* between those two collaborations. We could then check whether a path from *registration* to *consultation* exists in the choreography. If that is the case, the race is actual. Now, if we consider the collaboration diagram for *consultation* it is easy to see that there will not be races between *voice* and *test* at *Doctor*, since the latter does not play a non-initiating role in any of them.

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 Resolution of Race Conditions

Race conditions can be resolved in several ways. Some authors [44], [23] 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 Fig. 4) the race condition between *registration* and *consultation* at composite role *P* may be eliminated by introducing strong sequencing between *registration* and *assignment*.

Other authors (e.g. [36], [46]) 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 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. [36] 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. 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 *P* between *registration* 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 collaboration, two or more alternative sub-collaborations may be performed. Alternative composition is specified by means of choice operators, and describes alternatives between different execution paths. In a choice one or more *choosing* composite roles decide the alternative of the choice to be executed, based on the (implicit or explicit) conditions associated with the alternatives. Choosing roles are initiating roles. The other *non-choosing* composite roles involved in the choice follow the decision made by the choosing roles (i.e. execute the alternative chosen by the latter). Non-choosing 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 sequential composition with one inlet and a set of alternative outlets. The propositions and guidelines for sequential composition, given in Section 3.2, apply to every path through the choice. However, we will see how the choice-propagation process affects the resolution of races.

We assume collaborations to be weakly-causally composed, and therefore consider that the set of choosing roles is the union of the initiating roles of all collaborations immediately following the decision node.

1.3.3 Decision-making Process

We may distinguish the following situations:

1. The enabling conditions of the alternatives are mutually exclusive; only one of the sub-collaborations can be initiated.
2. The enabling conditions of several alternatives could be true; if the initiating composite roles of these sub-collaborations are different and there is no coordination between these roles, 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, see Fig. 7. We distinguish the following two sub-cases:
   a. The different sub-collaborations have different goals; only one of them should succeed. We call this situation *competing initiatives*.
   b. The different sub-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 doctor and the patient initiates the terminating collaboration of a voice call at the same time, see Fig. 7. We call this situation *mixed initiatives with common goals*.

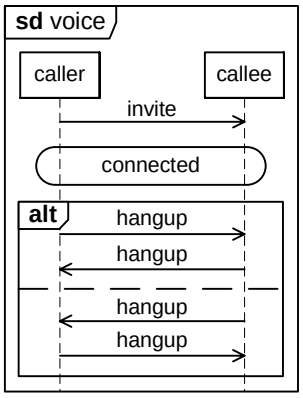

Fig. 7 A sequence diagram for the voice collaboration illustrating mixed initiatives with common goals.

Local Choice. Deciding the alternative to be executed becomes simple if there is only one choosing role, and the enabling conditions and triggering events for the alternatives 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 are directly realizable, since the decision is made by a single role based only on its local knowledge.

Non-local choice. The decision-making process gets complicated when there is more than one choosing role, as in Fig. 8 (a), where there are two choosing roles, namely A and B . From a global perspective, our intention is that once the choice node is reached, either role A initiates collaboration discA with B, or role B initiates collaboration discB with A. We are assuming then that there is an implicitly synchronization between A and B, which allows them to agree on the alternative to be executed. However, in a directly realized system, components A and B will not be able to synchronize and may decide to initiate both collaborations simultaneously resulting in a mixed initiative.

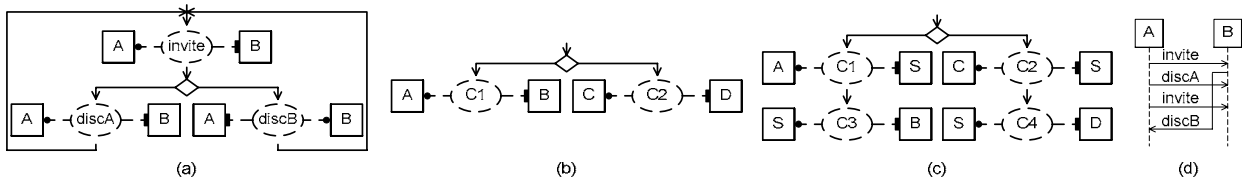

Fig. 8 (a) Example of a non-local choice; (b) Non-local choice where a mixed initiative conflict cannot be detected; (c) Non-local choice where a mixed initiative conflict can be detected.

Choices involving more than one choosing role are usually called **non-local choices** [10]. They are normally considered as pathologies that can lead to misunderstanding and unspecified behaviours, and algorithms have been proposed to detect them in the context of HMSCs (e.g. [10], [30]). Despite the extensive attention they have received, there is no consensus on how they should be treated. We believe this might be motivated by a lack of understanding of their nature. Some authors (e.g. [10]) consider them as the result of an underspecification and suggest their elimination. This is done by introducing explicit coordination, as a refinement step towards the design. Other authors look at non-local choices as an obstacle for realizability and propose a restricted version of HMSCs, called *local HMSCs* [29], [26], that forbid non-local choices. Finally, there are authors [28], [45] 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 chorography level as a choice where the alternatives 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 alternatives 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. In the TeleConsultation service, for example, two collaborations are

enabled after the execution of *available: assignment* and *unavailable* (see Fig. 4). The triggering events for these come form the end-users (i.e. the actual doctor and receptionist) that operate independently and are not coordinated. It makes little sense to coordinate components *D* and *R* 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 an initiating and a non-initiating sub-role, or between two non-initiating sub-roles played in collaborations with different peers. Note that if two alternatives with different choosing roles have no common roles, a mixed initiative conflict can not be detected (see, for example, Fig. 8 (b)). If the intention is that they shall be mutually-exclusive, an arbiter role should be introduced. Such arbiter role would act as an intermediary between the choosing roles and the non-choosing roles, and could detect a mixed initiative conflict (e.g. the choice in Fig. 8 (c) results from adding an arbiter role S to the choice in Fig. 8 (b)).

Situations of initiative choices were discussed by Gouda et al. [28] and Mooij et al. [45]. 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. [45] 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 [45].

2.3.3 Choice-propagation Process

The decision made by the choosing component must be properly propagated to the non-choosing components, in order for them to execute the right alternative. In each alternative, the behavior of a non-choosing component begins with the reception of a sequence of messages, which we call the *triggering trace*. Thereafter, the component may send and receive other messages. It is the triggering traces that enable a non-choosing component to determine the alternative chosen by the choosing component. In some cases, however, a non-choosing component may not be able to determine the decision made by the choosing component. As an example, we consider the local choice in Fig. 9 (a). For the component R3, the triggering traces for both alternatives are the same (i.e. the reception of message *x*). Therefore, upon reception of *x*, R3 cannot determine whether R1 decided to execute collaboration C1 or C2. That is, R1's decision is ambiguously propagated to R3. We say a choice has an **initial ambiguous propagation** if there is a non-choosing component for which the triggering traces *specified* in two alternatives have a common prefix. Note that according to this definition, triggering traces such as (?x, ?y) and (?x, ?z) cause initial ambiguous propagation since in any direct realization, the choice cannot be made immediately after *?x*. An easy solution in this case would be to delay the choice (i.e. extract *?x* from the choice). Choices with ambiguous propagation are not directly realizable. They are similar to the non-deterministic choices defined in [46]. Unfortunately, ambiguous propagation cannot be detected at the choreography level as it depends on the detailed

interactions of the sub-collaborations. In order to avoid ambiguous propagation, [13] suggested the introduction of a message parameter that indicates to which branch of the choreography the message belongs.

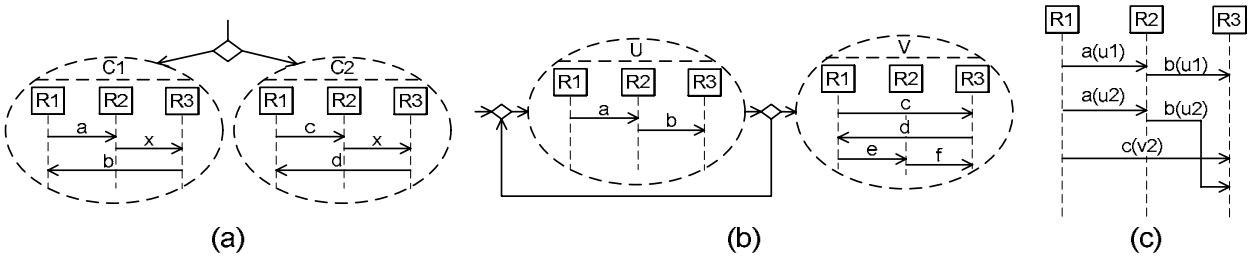

Fig. 9 Choices with (a) ambiguous propagation and (b) race propagation; (c) Behaviour implied by (b)

If any of the alternatives contain a weak sequence with a race condition, the race may make the propagation ambiguous. Consider the choice in Fig. 9 (b). In this case there is a race in the weak sequence of U and V. The choice is followed by either V or U^+ V and may result in the situation depicted in Fig. 9 (c). This example shows that in the presence of race conditions the triggering trace *observed* by a non-choosing component may differ from the specified one. Therefore, whenever race conditions may appear in any of the alternatives, we need to consider the potentially observable triggering traces in the analysis of choice propagation. For example, in Fig. 9 (b) the specified triggering traces for R3 are (?b^+,?c) and (?c). However, R3 may observe triggering traces such as (?c, ?b) or (?b, ?c, ?b). We say a choice has a **race propagation** if there is ambiguous propagation due to races. Choices with race propagation are not directly realizable. They are similar to the race choices defined in [46].

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

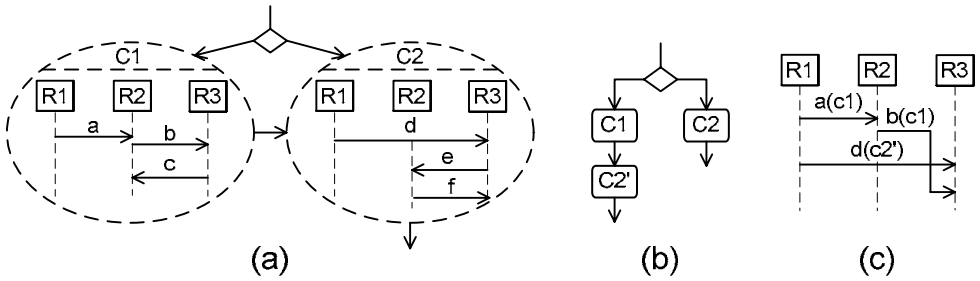

Fig. 10 (a) Choice with race propagation; (b) Unfolding of (a); (c) Behavior implied by (a)

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. However, if we try to remove the race conditions by means of message reordering for consumption (e.g. by means of separate input buffers), the race propagation problem may still persist. This is because, in general, a component would not be able to determine whether a received message should be immediately consumed as part of one alternative, or be kept for later consumption in another alternative as illustrated by the race propagation in Fig. 10 (a). To make the message reordering work, we need to mark the messages with the 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 (e.g. Fig. 10 (b) shows the unfolding of the choice in Fig. 10 (a) and the assignment of distinct collaboration ids). Fig. 10 (c) shows a possible scenario during the execution of the choice in Fig. 10 (a). If the messages are not marked, R3 cannot determine whether it should consume message *d* immediately after receiving it (i.e. in case C2 has been directly executed after the decision node), or whether it should keep it on the buffer until it receives message *b*. Marking message d with C2 (i.e. the id of the collaboration “type” it belongs to) would not help. It has to be marked with the id of the actual collaboration instance it belongs to (i.e. C2’).

When loops are involved, we need to consider the number of iterations of the loop in order to create the collaboration ids. Consider, for example, the choice in Fig. 9 (b). An unfolding of this choice would give an infinite number of alternative paths: U1→V1, U1→ U2→V2, U1→ U2→ U3→V3, and so on. In order to assign a proper id to each instance of U (or, in general, to each instance of a collaboration inside a loop) we just need to use the iteration number. The id assigned to each instance of V (or, in general, to each instance of a collaboration following the loop) depends on the total number of iterations that have been executed. The messages in the scenario of Fig. 9 (c) have been marked following these principles. Therefore, when R3 receives message *c(v2)* it may determine that there is still one message *b* on the communication medium, and wait for it without consuming *c* (for this R3 needs to keep the count of *b* messages that it has received).

In [26] the realizability of local-HMSCs is studied. The authors propose to implement the behavior of each component by means of a simple linear algorithm. This algorithm is based on the idea of marking 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 choices. Indeed, although the authors do not explicitly study the race propagation problem, their solution should in principle avoid such problems. The authors do not explain, however, the way to achieve a unique id for each HMSC node. They might consider this as something trivial, although we have shown it is not so trivial. Moreover, they propose marking all messages that are exchanged, and not only those involved in a race propagation. Components therefore have to check the data carried by *all* messages, and decide whether to consume them or not. We believe this unnecessarily increases the amount of processing needed upon message reception. We prefer to detect the cases of race propagation. Then, if we want to resolve the problematic propagation by means of message

reordering[5], we design the components so that they only mark the involved messages, and apply message reordering only to them. Alternatively, we may decide to resolve the race propagation in a probably simpler way, that is, by transforming the responsible weak sequencing into strong sequencing (see e.g. [48]).

3.4 Merge

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

3.5 Loop

Loops can be used to describe the repeated execution of a (composite) collaboration, which we call the *body* collaboration. A loop can be seen as a shortcut for strong or weak sequential composition of several executions of the same body collaboration, combined with a choice and a merge (see e.g. Fig. 9 (b)). 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 components. 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 collaboration (e.g. to be sure that one iteration is completely finished before the next one starts), the body collaboration should be initiated and terminated by the same component. When send-causal weak sequencing is specified, the component initiating the loop-body collaboration should be the one sending or receiving the last message exchanged in the collaboration.

Loops may give rise to so-called *process divergence* [10], characterized by a component

[5] This avoids not only race propagation, but ambiguous propagation in general

sending an unbounded number of messages ahead of the receiving component. This may happen with weak sequencing if the communication between any two of the participants in the body collaboration is unidirectional. They may also give rise to so-called *orphan* messages, i.e. messages sent in one iteration and received in a later iteration. Consider the specification in Fig. 8 (a), and imagine that each collaboration consists of only one message. Then the scenario in Fig. 8 (d) is possible, where message *discB* is sent as a response to the first *invite* message, but it is received by *A* after having sent the second *invite*. Component *A* may then consume message *discB* as a response to the second *invite* message, leading to an undesired behavior. In this scenario, *discB* is a so-called *orphan* message.

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 sub-collaborations are executed independently from one another, possibly at the same time. We use forks and joins to describe concurrency, and we require they are properly nested as in UML Interaction Overview Diagrams. Concurrent sub-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 possible 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 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 require coordination. Seen from the patient, 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 [20] and [21] 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 sub-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 of the sequential flow compositions in the set of compositions defined by a fork/join the conditions for (weak/strong) sequential composition explained in Section 3.2 apply.

For 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 component 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 [13].

3.7 Interruption

We consider here the interruption of a sub-collaboration C by another sub-collaboration C^{int} that may become enabled, for instance as soon as C is initiated, or when A 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* the observation of the external event *hangup* performed by the patient results in the interruption of the *waiting* collaboration by the *disconnect* 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 sub-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-awaited 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.

3.8 Activity invocation

In many cases, a given collaboration A needs to invoke another collaboration B in order to carry out some task. In the *TeleConsultation*, the doctor invokes the *test* collaboration while in the middle of the *voice* collaboration, as illustrated in Fig. 2 and Fig. 4.

We propose to model this situation in activity diagrams using two "stream" control flow

arrows, one representing the request for service and the other representing the results returned[6]. If a collaboration A, invoking a collaboration B, suspends its own behavior while waiting for the results of B, then this collaboration invocation corresponds to the semantics of a procedure call, which is a case of strong sequencing. This is directly realizable if collaboration B is initiated and terminated by a single role that also participates in A. If this is not the case, additional coordination messages are needed to ensure strong sequencing.

Activity invocations should be checked to ensure that no invocation cycles are created (e.g. A invokes B, which in turns invokes C, which in turns invokes A). These cycles may lead to deadlocks. Mixed initiatives may also appear if the invoking collaboration does not suspend its behavior. An example can be found in the TeleConsultation service. Assume that the behavior of *voice* is given by the sequence diagram in Fig. 7, and that the *test* collaboration is invoked when *connected* is true. Then a result from *test* may be received when callee has already sent a *hangup* message.

We note here that streaming pins allow information to be exchanged between concurrent activities and provides a general mechanism to model information exchange between collaborations that are executing in parallel, as has been demonstrated by Kraemer and Herrmann [38]. This possibility is not elaborated here, but we remark that such interchange is directly realizable if localized within one role, as indicated in Fig. 5 (b) and Fig. 5 (c).

3.9 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

[6] One may use several "stream" control flows for representing different types of results that could be obtained, such as normal and exceptional cases.

the problem of realizability to the notion of implied scenarios. They consider a specification given as a set of MSCs describing asynchronous interactions, and analyze it to check if it implies any non-specified MSC. Intuitively, a realizable specification does not contain 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 [57]. The authors present a technique to detect implied scenarios from a specification describing both positive, as well as 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 [47], [9], [26]. The authors of [47] study safe realizability, with additional message contents, of regular (finite state) HMSCs with FIFO channels. This work is extended in [9], 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 [26] discusses two classes of unbounded HMSCs. They claim that so-called *local-choice* HMSCs are always safely realizable with additional message contents[7]. A subclass of *local-choice* HMSCs that are safely realizable without additional message contents was studied in [29].

Other authors have studied conditions for realizability of Compositional MSCs [46] and pathologies in HMSCs [10], [30] and UML sequence diagrams [8] that prevent their realization.

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

None of these works discusses the nature of the realization problems.

3.10 System composition

In service engineering it is desirable that services can be modeled as independently as possible and then be composed in a modular way at design time and/or runtime. So far we have discussed the composition of a service defined as a collaboration among roles.

In general, a system may provide many different services, and many occurrences of the same service may be running concurrently. A collaboration, such as the TeleConsultation, may just be one of many collaborations to be realized in a given system, and a given system may run several TeleConsultations concurrently. In order to be executed in a system, each role must be bound to a component that can execute it, either statically or dynamically. In general a role may be assigned to many different components and each component may be assigned several different roles.

Using UML, the system structure and the binding of roles to components may be defined using composite classes with inner parts, as illustrated in Fig. 11. Clearly *system composition* has similarities with, but is not the same as the collaboration composition we have discussed so far. One will normally not define the complete system behavior explicitly as the choreography of an enclosing collaboration, but rather let it follow implicitly from the system structure and the binding of roles to the system components. System composition raises a number of issues and problems related to the composition and coordination of roles that will not be discussed further here. We only remark that, to a large extent, they may be handled outside the roles by additional coordination functionality in the components.

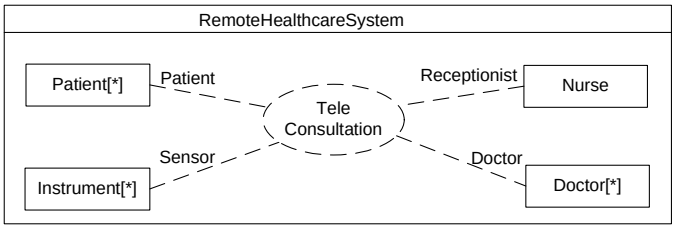

Fig. 11 Actors with role bindings

4 GOING FROM COLLABORATIONS TO COMPONENT DESIGNS

We discuss in this section the possibility of deriving the behaviors of the components in a distributed design automatically from the behavior of a collaboration which we assume is given in the form described in Section 2. In general, each component will realize the behavior of at least one role identified in the collaboration. The behavior of each component will be given in terms of local actions to be performed and messages exchanged with the other components within the system. The term "protocol" [14] denotes this behavior that must be satisfied for obtaining compatibility between the actions performed by the different system components.

During the past years, much research effort has been spent on the problem of deriving component behaviors from scenario-based specifications of the system behavior (for a survey, see [42]). The system behavior is usually defined in terms of sequence diagrams or similar notations. In this context, most of the issues discussed in Section 3 must be addressed. As explained in Sections 2 and 3, collaborations provide useful structuring and composition mechanisms to describe and analyze the requirements that are the starting point for a systematic development process, and are therefore preferable to message sequence diagrams, which are at a relatively low level of abstraction. Nevertheless, message sequence scenarios can be derived from higher-level specifications in the form of activity diagrams or Use Case Maps [7], and then one could derive component behaviors in a second step. In the following, however, we do not follow that approach, but consider instead the direct derivation of component behavior from the

specification of a collaboration.

4.1 Protocol derivation from service specification

Traditionally, an abstract view of a collaboration within a distributed system is called a service [12]. The specification of a service behavior describes actions that are executed at different "service access points" and their temporal order. A service access point corresponds to a role (or a participating component) in the definition of a collaboration. At this level of abstraction, the exchange of messages between the different roles is normally not shown. However, these messages are essential at the level of the protocol specification which defines the behavior of each component in the distributed system. A body of work exists that describes algorithms for deriving a protocol specification from a given service specification. The service specification defines the temporal ordering of elementary actions that are associated with the components of the system. A protocol derivation algorithm, therefore, derives the necessary message exchanges between the different components in order to assure that the service actions are executed in an order consistent with the service specification. The initial work in [13] assumes that the service specification consists of elementary actions where the temporal ordering is defined by sequence, alternative and concurrency operators; the inclusion of message parameters for data transfer was added in [27]. In [35], temporal orderings with regular recursion and sub-collaborations are introduced. General recursion is dealt with in [48] and [34], and the latter also deals with interruption.

The basic idea of these algorithms is to first identify for each sub-collaboration the roles (components) involved, and in particular the initiating and terminating roles (components). If two sub-collaborations should be executed in the temporal order of a strong sequence, then the protocol derivation algorithm introduces a **coordination message** from each terminating

component of the first sub-collaboration to each initiating component of the second sub-collaboration. In many cases, the sub-collaborations have only a single initiating and terminating component; in this case a single coordination message is sufficient; and when both components are the same, no coordination message is required, since the sequencing can be enforced by the single component.

We note that most of these approaches only consider strong sequencing and assume that the service specification does not include a non-local choice. However, a non-local choice can be handled by introducing a conflict-resolution protocol between the components involved, for instance in the form of a circulating token, or by introducing priorities as suggested by Gouda [28], however, no general solution exists. Most derivation algorithms also assume that each component has separate input buffers for all its partners, and that there is no message overtaking.

4.2 Protocol derivation for Petri-nets

The problem of protocol derivation from service specifications has been studied also for the case that the service specification is given as a Petri net or some extended form of Petri nets [60], [33]. This is of particular interest to us because the semantics of activity diagrams can be described naturally with Petri nets. An elementary action of a collaboration (described as an activity diagram) corresponds to a transition of the corresponding Petri net. Each transition of the Petri net is therefore associated with one of the roles of the collaboration. The Petri net tokens that pass from one transition to another represent coordination messages. Non-local choice remains a problem. We note that Petri net extensions have been considered, including pre- and post-conditions for transitions and variables that may be located at different components. The derived protocol includes mechanisms for checking non-local pre-conditions, updating of variables, and the control of access to shared resources [61].

4.3 Semi-automatic designs of collaborations

From the above discussion, we get the following conclusion: Given the behavior of a collaboration described in terms of sub-collaborations and elementary actions and their allowed execution order, the problem of deriving the behavior of components that will realize this global behavior through message exchanges has been solved under the assumption that there is (a) no weak sequencing, and (b) no non-local choice or mixed initiatives.

For the many cases where these assumptions are not satisfied, further work is needed for finding appropriate solutions for the component behaviors. Concerning weak sequencing, a composition of sub-collaborations satisfying Propositions 2 or 4 in Section 3 has been shown to be directly realizable. In [22] we provide proofs of this and also algorithms to check if the conditions for direct realizability of weak sequencing are satisfied or not.

We hope that the guidelines given in Section 3 will eventually lead to the semi-automated derivation of component behaviors from a service specification given in the form of sub-collaborations, elementary actions and their allowed execution order. This would lead to a semi-automatic process, where the designer has to choose domain-specific solutions to those problems for which no general solution is available, namely non-local choices and mixed initiatives.

5 CONCLUSIONS

In Section 1 we asked ourselves the following questions. Is it possible to model service behavior more completely? Can it be done in a structured way without revealing more interaction detail than necessary? Is it possible to support composition and to detect and remove realization problems? And is it possible to derive detailed implementations automatically from service models? We have shown here that a collaboration oriented approach based on UML 2 collaborations have potential to provide positive answers to several of these questions. In

particular we have demonstrated how choreographies 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 resolve realization problems. To our best knowledge we are able to identify all the realization problems that have been reported in literature, many at the level of choreography, without needing to consider detailed interactions of sub-collaborations. We have also argued that our approach can be supported by tools that automatically generate correct implementations from service specifications. Evidence of this has been provided through several demonstrations by our groups and others.

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 Soft. Eng. (ICSE'00)*, 2000.

[5] R. Alur, K. Etessami and M. Yannakakis, "Realizability and verification of MSC graphs", *Theor. Comput. Sci.*, 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] D. Amyot, D.Y. Cho, X. He and Y. He, "Generating scenarios from Use Case Map specifications", *Proc. 3rd Intl. Conf. on Quality Software (QSIC'03)*, Dallas, November 2003.

[8] 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.

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

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

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

[12] G. v. Bochmann and C. A. Sunshine, Formal methods in communication protocol design, (invited paper) IEEE Tr. COM-28, No. 4 (April 1980), pp. 624-631, reprinted in "Communication Protocol Modeling", edited by C. Sunshine, Artech House Publ., 1981.

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

[14] G. v. Bochmann, "Protocol specification for OSI", *Computer Networks and ISDN Systems*, 18, April, 1990, pp.167-184.

[15] G. v. Bochmann, "Activity nets: a UML profile for modeling work flow architectures", Technical Report, University of Ottawa, Oct. 2000.

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

[17] 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.

[18] R. Bræk and G. Melby, "Model driven service engineering", *in Model-driven Software Development*, vol. 2 of *Research and Practice in Software Engineering*, S. Beydeda, M. Bookand V. Gruhn, Eds., Springer, 2005.

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

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

[21] 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.

[22] 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.

[23] 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.

[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] B. Genest, A. Muscholl, H. Seidl and M. Zeitoun, "Infinite-state high-level MSCs: Model-checking and realizability", *J. Comput. Syst. Sci.*, 72(4), 2006, pp. 617–647.
[27] R. Gotzhein and G. v. Bochmann, "Deriving protocol specifications from service specifications including parameters", *ACM Transactions on Computer Systems*, vol.8, no.4, 1990, pp. 255-283.
[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] 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.
[30] 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.
[31] IUT-T, *Specification and Description Language (SDL)*, Recommendation Z.100, 2000.
[32] IUT-T, *Message Sequence Charts (MSC)*, Recommendation Z.120, 1998.
[33] H. Kahlouche and J. J. Girardot, "A stepwise requirement based approach for synthesizing protocol specifications in an interpreted Petri net model", *Proc. INFOCOM'96*, 1996, pp. 1165–1173.
[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] F. Khendek, G. v. Bochmann and C. Kant, "New results on deriving protocol specifications from services specifications", *Computer Communications Review*, vol.19, no.4, July, 1989, pp. 136-145.
[36] 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.
[37] F. A. Kraemer and P. Herrmann, "Service specification by composition of collaborations – An example", *Proc. 2nd Int. Workshop on Service Composition (SERCOMP'06)*, Hong Kong, IEEE Comp. Soc., 2006.
[38] F. A. Kraemer, P. Herrmanna and R. Braek, "Synthesizing components with sessions from collaboration-oriented service specifications", *Proc. 13th SDL forum*, LNCS, vol. 4745, Springer, September, 2007.
[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] L. Lamport, "Time, clocks and the ordering of events in a distributed system", *Comm. ACM*, 21, 7, July, 1978, pp. 558-565.
[42] H. Liang, J. Dingerl and Z. Diskin, "A comparative survey of scenario-based to state-based model synthesis approaches", *Proc. 5th ICSE Intl. workshop on Scenarios and State Machines: models, algorithms, and tools (SCESM'06)*, ACM Press, 2006.
[43] M. Lohrey, "Realizability of high-level message sequence charts: closing the gaps", *Theor. Comput. Sci.*, 309(1-3), 2003, pp. 529–554.
[44] B. Mitchell, "Resolving race conditions in asynchronous partial order scenarios", *IEEE Trans. Softw. Eng.*, 31(9), 2005, pp. 767–784.
[45] 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 Soft. Eng. (FASE'05)*, LNCS, 3442, Springer, 2005.
[46] 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.
[47] 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.
[48] [Nakata 98] A. Nakata, T. Higashino and K. Taniguchi, "Protocol synthesis from context-free processes using event structures", *Proc. 5th Intl. Conf. on Real-Time Computing Systems and Applications (RTCSA'98)*, Hiroshima, Japan, IEEE Comp. Soc. Press, 1998, pp.173-180.
[49] OMG, *UML 2.1.1 superstructure specification*, accessible at http://www.omg.org/cgi-bin/apps/doc?ptc/06-04-02.pdf
[50] [Reenskaug 92] 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.
[51] [Reenskaug 95] T. Reenskaug, P. Wold and O.A. Lehne, *Working with objects: The OOram software engineering method*, Prentice Hall, 1995.
[52] [Roychoudhury 03] A. Roychoudhury and P. S. Thiagarajan, "Communicating transaction processes: An MSC-based model of computation for reactive embedded systems", *Lectures on Concurrency and Petri Nets*, LNCS, vol. 3098, Springer, 2003.
[53] [Sanders 00] R. T. Sanders, "Implementing from SDL", *Telektronikk*, vol. 96, no. 4, 2000.
[54] [Sanders 05a] R. T. Sanders, R. Bræk, G. v. Bochmann and D. Amyot, "Service discovery and component reuse with semantic interfaces", *Proc. 12th Intl. SDL Forum*, Grimstad, Norway, LNCS, vol. 3530, Springer, 2005.
[55] [Sanders 05b] R. Sanders, H. N. Castejón, F. A. Kraemer and R. Bræk, "Using UML 2.0 collaborations for compositional service specification", *Proc. ACM}/IEEE 8th Intl. Conf. on Model Driven Engineering Languages and Systems (MoDELS'05)*, LNCS, vol. 3713, Springer, 2005.
[56] [Sanders 07] R. T. Sanders, "Collaborations, semantic interfaces and service goals - a new way forward for service engineering", PhD thesis, Norwegian University of Science and Technology (NTNU), 2007.
[57] [Uchitel 04] S. Uchitel, J. Kramer and J. Magee, "Incremental elaboration of scenario-based specifications and behavior models using implied scenarios", *ACM Trans. Softw. Eng. Methodol (TOSEM)*, 13(1), 2004, pp. 37–85.
[58] [Whittle 07] J. Whittle, "Precise specification of use case scenarios", *Proc. 10th Intl. Conf. on Fundamental Approaches to Software Engineering (FASE'07)*, LNCS, vol. 4422, Springer, 2007.
[59] [Wohed 05] P. Wohed, W. M. P. van der Aalst, M. Dumas, A. H. M. ter Hofstede and N. Russell, "Pattern-based analysis of the control-flow perspective of UML activity diagrams", *Proc. 24th Intl. Conf. on Conceptual Modeling (ER'05)*, LNCS, vol. 3716, Springer, 2005.
[60] [Yamaguchi 95] H. Yamaguchi, K. Okano, T. Higashino and K. Taniguchi, "Synthesis of protocol entities' specifications from service specifications in a Perti net model with registers", *Proc. 15th Intl. Conf. on Distributed Computing Systems (ICDCS'95)*, IEEE Comp. Soc. Press, 1995.
[61] [Yamaguchi 03a] H. Yamaguchi, K. El-Fakih, G. v. Bochmann and T. Higashino, "Protocol synthesis and re-synthesis with optimal allocation of resources based on extended Petri nets", *Distributed Computing*, vol. 16, no. 1, March, 2003, pp. 21-36.