Using UML 2.0 Collaborations for Compositional Service Specification

Richard Torbjørn Sanders[1], Humberto Nicolás Castejón[2], Frank Alexander Kraemer[2], and Rolv Bræk[2]

[1] SINTEF ICT, N-7465 Trondheim, Norway
richard.sanders@sintef.no
[2] NTNU, Department of Telematics, N-7491 Trondheim, Norway
{humberto.castejon, kraemer, rolv.braek}@item.ntnu.no

Abstract. Collaborations and collaboration uses are features new to UML 2.0. They possess many properties that support rapid and compositional service engineering. The notion of collaboration corresponds well with the notion of a service, and it seems promising to use them for service specification. We present an approach where collaborations are used to specify services, and show how collaborations enable high level feature composition by means of collaboration uses. We also show how service goals can be combined with behavior descriptions of collaborations to form what we call semantic interfaces. Semantic interfaces can be used to ensure compatibility when binding roles to classes and when composing systems from components. Various ways to compose collaboration behaviors are outlined and illustrated with telephony services.

1 Introduction

Service development or service engineering is currently receiving considerable attention and starting to become a discipline in its own right. Driven by the belief that future revenues will have to come from new services, a tremendous effort is being invested in new platforms, methods and tools to enable rapid development and incremental deployment of convergent services, i.e. integrated communication, multimedia and information services delivered transparently over a range of access and transport networks. The Service Oriented Architecture (SOA) and Service Oriented Computing (SOC), building on web services, are exponents of this trend in the business domain. A general challenge for service engineering, be it business or ICT applications, is to enable services and service components to be rapidly developed, and to be deployed and composed dynamically without undesirable service interactions. This is a challenging problem largely due to fundamental properties of services, i.e.:

- A service is a *partial functionality.* It can be combined with other services to provide the full functionality offered to a user.
- A service execution normally involves several *collaborating* components (i.e. a service is not simply an interface to an object).
- Components can participate in several services, simultaneously or alternately.

– Services are partially dependent on each other, on shared resources and on user preferences.

In order to support model driven service engineering, corresponding modeling concepts are needed. This is where UML 2.0 collaborations come in, since they possess many properties that make them attractive for this purpose.

First of all the concept of UML collaboration corresponds closely with the concept of a service as explained above. We actually define a *service* as a collaboration between *service roles* played by objects that deliver functionality to the end users. Note that this definition is quite general and covers both client-server and peer-to-peer services as described in [1].

Secondly, UML collaboration uses provide a means to structure complex collaborations and give an overview not provided by other notations, while at the same time being precise. Collaborations have much the same simplicity and appeal as use cases, and can be used for the much same purposes, but provide additional benefits for service engineering, as will be presented in the following. Service specification using collaborations and collaboration uses fits well with the preferred view of marketers and end-users, while at the same time supporting the difficult engineering tasks of service and system designers.

Thirdly, a collaboration role can be bound to several different classifiers by means of collaboration uses. This provides the desired flexibility to bind service roles to components, the only UML requirement being that the classifier is *compatible* with the type of the role(s) bound to it. A precise definition of compatibility is left as a semantic variation in UML 2.0, but it is clear that this should entail the observable behavior on interfaces of a component.

This leads to a fourth motivation for collaborations – they lend themselves nicely to the definition of so-called *semantic interfaces* [2]. As we shall see, a two-party collaboration can define a pair of complementary semantic interfaces. Compared to traditional syntactical interfaces known from web services, CORBA, Java and UML, semantic interfaces also define the visible interface behavior and the goals of the collaboration. This extends the notion of compatibility beyond static signature matching to include safety and liveness properties. It also provides an efficient means to perform such compatibility checks at design time and even at runtime.

Finally, it may be argued that the crosscutting view of collaborations is valuable in its own right [3]. It enables us to focus on the joint behavior of objects rather than on each object individually and, not the least, to focus on the purposes and goals of the joint behavior in terms of desirable global states, called service goals in [4]. A service goal can be expressed in OCL, and is a property that identifies essential progress, thus characterizing a desired or successful outcome of a service invocation. It can be argued that service goals are closer to capturing and expressing the user needs than specifying how they are achieved in terms of detailed interactions. Moreover, goal expressions define liveness properties that must be satisfied by compatible components.

Fig. 1 provides a principal overview of service engineering using collaborations.

Our service engineering approach is both collaboration-oriented and compositional. It is collaboration-oriented because we model services as collaborations between roles played by distributed components, and it is compositional because we build services

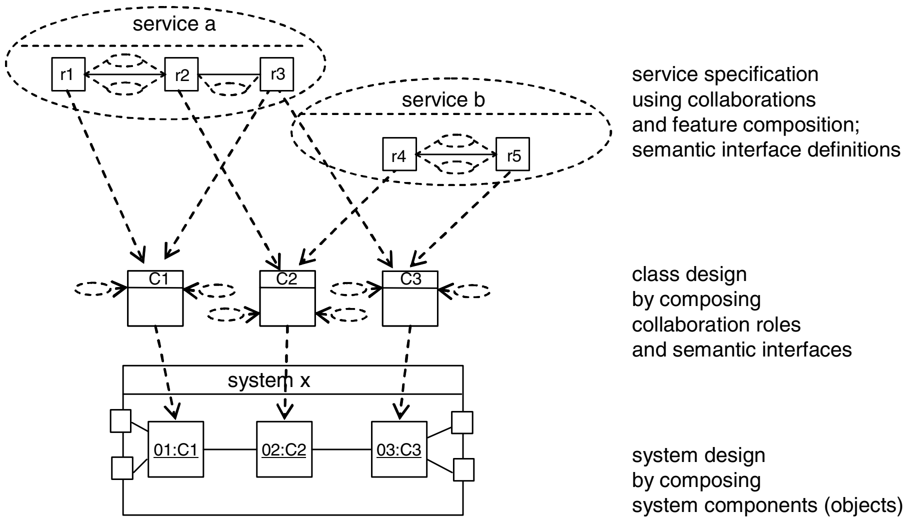


Fig. 1. Service engineering overview

from other smaller services. We treat collaborations and collaboration roles as units of reuse.

We consider the following composition cases:

1. Composition of two-party services and semantic interfaces from two-party collaborations.
2. Composition of multi-party services from two-party or n-party collaborations.
3. Class design by composing service roles and semantic interfaces.

Class design is out of the scope of this paper. Here we focus on the use of collaborations for service specification. It is our belief that class design can become a more mechanical process supported by tools if it takes collaborations and semantic interfaces as input. Our experience so far indicates that this is the case [5, 6]. However, further work is still needed to confirm this with certainty.

1.1 Structure of the Paper

In section 2 we present how service structures can be described in UML, and how service behavior can be described. We introduce the concept of service goals, and discuss how they can be defined in service structures and in the behavioral descriptions. We introduce what lies in a semantic interface, and discuss compatibility between roles and classifiers.

In section 3 we discuss the composition of two-party collaborations used for defining semantic interfaces, as well as composing multi-party services from subordinate collaborations, and indicate directions toward class design. Finally we conclude.

2 Collaborations, Goals and Semantic Interfaces

2.1 Collaboration Structure

When used for service specification, the structure of a collaboration identifies the service roles that collaborate to provide the service, as well as their multiplicity and interconnections. Fig. 2 depicts a collaboration called `UserCall` specifying the structure of a classical telephone call service. This collaboration diagram tells us that exactly two roles, `A` and `B`, of type `Caller` and `Callee` respectively, are needed to provide a `UserCall` service, and that a communication path between instances playing those roles must exist.

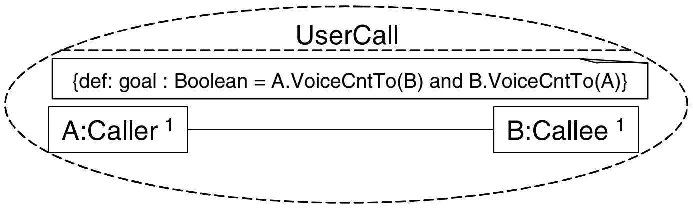


Fig. 2. The `UserCall` service specified as a collaboration with a goal expression

Specifying a service as a collaboration enables roles to be identified and described without introducing undue bindings to implementation details. Thus a service can be specified and understood as a behavioral component of its own, independent of systems components that implement them.

As we shall see, the behavior of collaborations can be described at several levels of detail. Furthermore, collaborations can themselves be used as components in collaboration compositions, thus becoming units of reuse.

2.2 Collaboration Goals

The diagram in Fig. 2 also shows a goal that should be reached by the `UserCall` collaboration. It is represented by an OCL predicate over properties of the two participating roles. In this case it is a simple logical addition of the role goals of `A` and `B`, to show that `A` has a voice connection to `B` and `B` has a voice connection to `A`:

```
VoiceCnt(A,B) = A.VoiceCntTo(B) and B.VoiceCntTo(A)
```

Goal expressions like this can be made very high level, protocol independent and close to the essential purpose of a service as seen from a user point of view. They are actually formal requirements expressions. In this respect they are not new; the novelty lies in the natural binding to the different service specification diagrams, such as collaborations and sequence diagrams. Furthermore, a goal expression represents a liveness property that should hold in actual collaboration uses and therefore constitutes part of the required compatibility of role binding. This illustrates one asset of UML collaborations: they are natural places to express crosscutting properties of services.

2.3 Collaboration Behavior

Since UML collaborations inherit from both structured classifiers and behaviored classifiers, they have a large range of expression forms at their disposal. In addition to expressing structural relationships, it is possible to express all forms of behavioral aspects of collaborations, such as interactions, activities and state machines. The UML standard [7] and reference book [8] focus mainly on the structural features of collaborations, and provide few guidelines on how the behavior of a collaboration is described, nor do they explain how collaboration behavior is related to the behavior of its constituent parts, i.e. the roles and role classifiers.

In the following we suggest how the behavior of a collaboration can be described for the purpose of service specification. We first specify the main states a collaboration goes through with a state diagram. This helps to abstract away details and focus on the goal of the collaboration. Thereafter detailed interactions for the collaboration are provided in the form of sequence diagrams.

Collaboration States. The states (or phases) of a collaboration may be described in a state diagram (or activity diagram), as illustrated in Fig. 3.

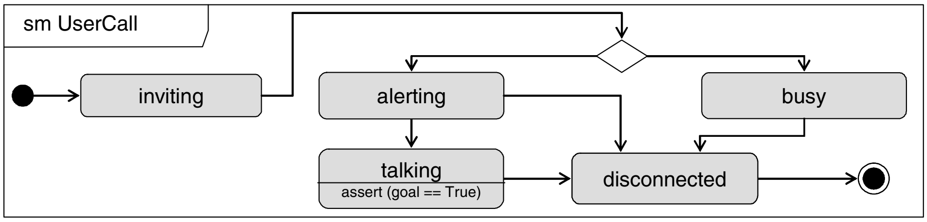


Fig. 3. State machine diagram for collaboration `UserCall`

This state diagram describes well known situations in the progress of a basic telephone call. The transitions between the states are represented by arcs, but we have chosen not to define exactly what causes them. For instance the transition from `alerting` to `disconnected` can be due to the `caller` hanging up, the callee not answering before a timeout, or the network malfunctioning. Leaving such details undefined can be desirable in a high level service specification.

But what do states of a collaboration mean? Given that a collaboration is not instantiated as an object, no entity is ever in a collaboration state. Rather, a collaboration state is a conceptual state expressing certain situations or conditions on the combined states of the roles `A` and `B` during the collaboration, see Fig. 4. It may be considered as a liveness property of the collaboration.

The possibility to focus on the joint behavior and goals rather than the individual role behavior is an important asset of collaborations. The role behaviors must somehow be aligned with each other; we indicate a way of doing so in Fig. 4. One must ensure that the role behaviors are dual, i.e. they are fully compatible with respect to safety properties, and that they can reach the joint collaboration states and goals and thereby satisfy liveness properties. A two-party collaboration satisfying these properties defines a pair of semantic interfaces [2].

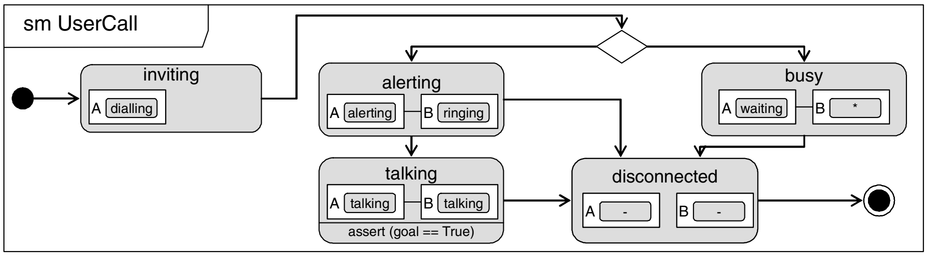


Fig. 4. State machine diagram for `UserCall` with role states and service goal expression (UML enhancement illustrating role states in collaboration states)

By describing state machines for both the collaboration and the role classifiers, a certain amount of redundancy is added, and the question of compatibility between them arises. This can either be considered as a problem to be avoided, or as a feature that can be put to use. In our view validating consistency between the role behavior and the collaboration behavior is an opportunity that should not be missed.

Interactions. Interaction diagrams are often partial descriptions that are not meant to describe complete behavior, unlike state machine diagrams. For the purpose of service specification interactions for a collaboration should at least focus on the successful cases, i.e. those that lead to the achievement of service goals.

In Fig. 5 we have described interactions that lead to the achievement of the service goal of a collaboration called `Invite`. The goal of this collaboration is to bring the collaborating instances to the `talking` state. The goal is indicated by an adornment in the continuation label `talking`.

2.4 Semantic Interfaces and Compatibility

In principle, components can participate in any service as long as they can play their part of the service. Therefore, the specification of a service should not bind the service roles to specific classifiers [9]. In [10] we used association classes to specify services, but they fail to meet the requirements for flexible role binding. This is because with associations the binding is determined by the classifiers at the association ends. Collaborations do not have this limitation. With the help of collaboration uses, collaborations roles can be bound to any classifiers that are compatible with the role types. This is shown in Fig. 6, where the same classifier, `UserAgent`, is bound to two different roles, `A` and `B`. This is possible as long as the `UserAgent` class is compatible with both collaboration roles. Our interpretation of compatibility is that the `UserAgent` must have visible interface behavior that is goal equivalent with the behavior of both roles, implying that the roles of the collaboration can be achieved.

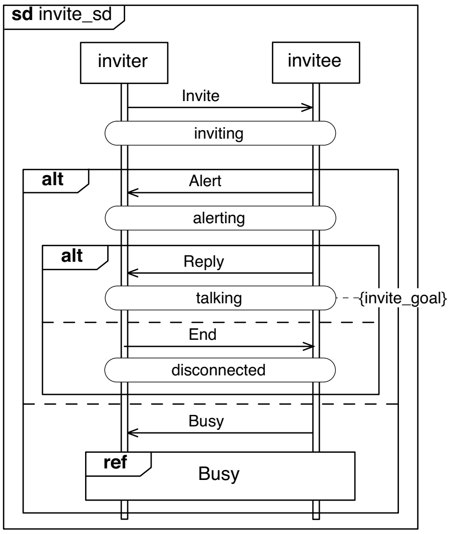


Fig. 5. Sequence diagram for collaboration `Invite`

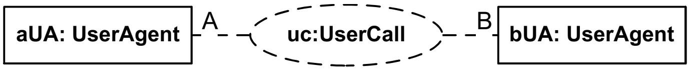


Fig. 6. Binding roles to component classes in a collaboration use

This can be put to use by defining a pair of *semantic interfaces* in a two-way collaboration like `UserCall`, as proposed in [2]. The semantic interfaces include goal expressions and role behaviors for the two collaboration roles. Such role behavior can be seen as a kind of protocol state machine specifying only the input/output behavior visible on the interface. It can be derived from a general state machine by making a projection of its behavior on the interface in question. In the case of the `UserAgent` in Fig. 6, compatibility can be checked in two steps. First we verify that the collaboration goals of `UserCall` are reachable given the roles `A` and `B`. Then we check that the projected behaviors of `UserAgent` on each side of the connection defined by `UserCall` are goal equivalent to the respective behaviors of `A` and `B`. This enables a compositional and scalable validation approach where the most computation intensive work (making projections and comparing behaviors) can be done at design time. When dynamically binding roles to system components at runtime, validation need not be repeated.

The UML standard [7] says that "a collaboration is often defined in terms of roles typed by interfaces". Unfortunately an interface typing a role can only describe either a *provided* interface, or a *required* interface, but not a combination. This is a limitation. We want role classifiers to describe both the required and the provided interface behavior in a single modeling unit. Typing a role by two interfaces, a required and a provided one, is not legal in the current version of UML, nor would this result in a unified interface description. Similarly, a protocol state machine attached to an interface only constrains the sequence of operation calls to a component, and can not be used to describe a two-way interface.

The limitations of interfaces may be overcome, however, if UML allowed describing interface behavior in terms of state machines that model the (projected) input/output behavior of a component on the interface, such as the Port State Machines (PoSM) proposed by Mencl [11]. This is indeed close to the port state machines of ROOM [12], and should be included in UML. Goal compatibility between a component and a port state machine could then be defined in terms of behavior projection.

Given that the behavior of a collaboration role is described in a state machine diagram enriched with service goals, it is relatively straightforward to validate safety and liveness compatibility between a classifier and a semantic interface to which it is bound [6, 13, 10], thus ascertaining goal equivalence between objects and roles.

3 Composition from Collaborations

3.1 Composition of Two-Party Services and Semantic Interfaces from Two-Party Collaborations

With collaboration uses we can express how services can be composed from elementary service features, as illustrated in Fig. 7.

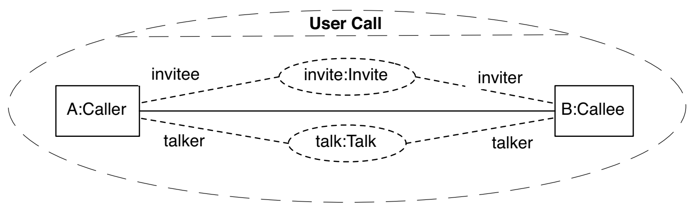


Fig. 7. `UserCall` composed of elementary features (subordinate collaboration uses)

In Fig. 7 the `UserCall` collaboration is decomposed into smaller features, `invite` and `talk`, represented as collaboration uses. These are related to the distinct states of the `UserCall` service (see Fig. 3) and to the sequence diagram for `Invite` (see Fig. 5). To simplify the example, we have grouped the states for `UserCall` so that the goal of the `invite` collaboration is to bring the `UserCall` collaboration to the state `talking`, upon which the `talk` collaboration use takes over. However, it is not clear from Fig. 7 what relationship there is between `invite` and `talk`, that is, if their interactions are interleaved or if they represent a sequence.

It is of central importance to service engineering to make the sequence of goals and the relationships between collaborations explicit. This may be done in several ways. One possibility is showing dependencies between the subordinate collaboration uses and/or their roles in the collaboration diagram itself. Another possibility is to utilize pre- and post-conditions. A third possibility is to use interaction overview diagrams or activity diagrams to express goal sequences, as suggested in Fig. 8a below.

Interaction overview diagrams are a form of activity diagram, and thus the token passing semantics of the latter apply. To express goal relationships, the following interpretation of the tokens is employed: a token being passed represents that a goal is achieved, while an input token implies that a subsequent collaboration use (i.e. a service) is enabled. This can be exploited by mechanisms supporting the dynamic discovery of service opportunities [2, 4]. Note that what happens if the goal is not achieved is not described – the focus is on the achievement of goals. However, if the goal is not achieved in a referenced collaboration, the goal sequence is interrupted.

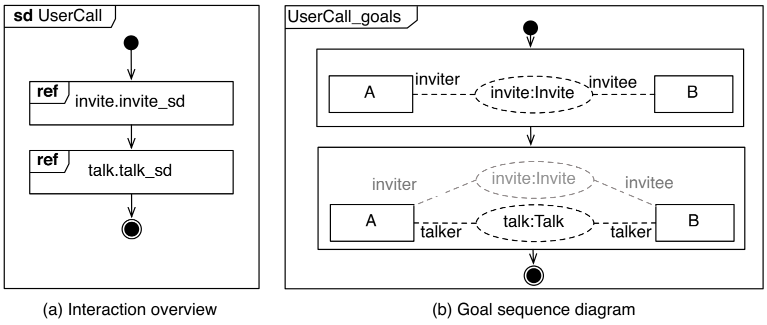


Fig. 8. Overview of the subordinate collaboration uses of `UserCall`

With this interpretation, Fig. 8a specifies that after `invite` has achieved its service goal, the subordinate collaboration use `talk` is enabled. Note that this relationship applies in the context of their use, i.e. in the collaboration `UserCall`. It is not stated in the specification of the subordinate collaborations `Invite` and `Talk`, which are thus free to be used in other collaboration contexts.

A minor diagrammatic enhancement to UML, which is to include an illustration of the situation with respect to the involved collaborations (see Fig. 8b), seems attractive. This is what we have called a *goal sequence diagram* [10]. The second rectangle in Fig. 8b illustrates how the roles of `Invite` and `Talk` are bound in the context of `UserCall`. They are statically bound in the `UserCall` collaboration of Fig. 7, and simply referred to in Fig. 8b. Goal sequence diagrams do not change the semantics of UML, and what is illustrated in Fig. 8b corresponds to what is expressed in Fig. 8a. Goal sequence diagrams illustrate the evolution of the collaboration structure. For instance, two shades of coloring are employed for the referenced collaboration uses: black color (e.g. for `talk`) illustrates that the collaboration use is active, while grey color (e.g. for `invite`) is for preceding collaboration uses that do not have to exist any longer. For the simple example in Fig. 8 the added value of the goal sequence diagram is not striking; Fig. 10 is perhaps a more convincing case.

Illustrating situations has been also suggested by Diethelm & al. [14]; they use communication diagrams to illustrate use cases and to illustrate do-actions in states.

Two-party collaborations can be composed to form semantic interfaces, which define role behavior and goals of a pair of complementary roles. Limiting such collaborations to a pair of roles is chosen to simplify the validation approach, which is based on validation of object behavior projections and goals over a binary association, as mentioned previously. It also simplifies composition, as components can be composed of composite states that correspond to the semantic interfaces [15].

This restriction does not hinder multi-party services to be defined; they can be composed from two-party collaborations with semantic interfaces, as well as from subordinate multi-party collaborations, as shown below. However, this complicates the validation and composition process, as several interfaces have to be validated or composed, and the relationships between the interfaces must be known. Goal sequence diagrams seem to be promising when it comes to composition, as illustrated in the next section.

3.2 Composition of Multi-party Services

An example that illustrates the potential of composing collaborations from subordinate collaborations is found in Fig. 9, where the `UserCall` service with the call transfer feature is described.

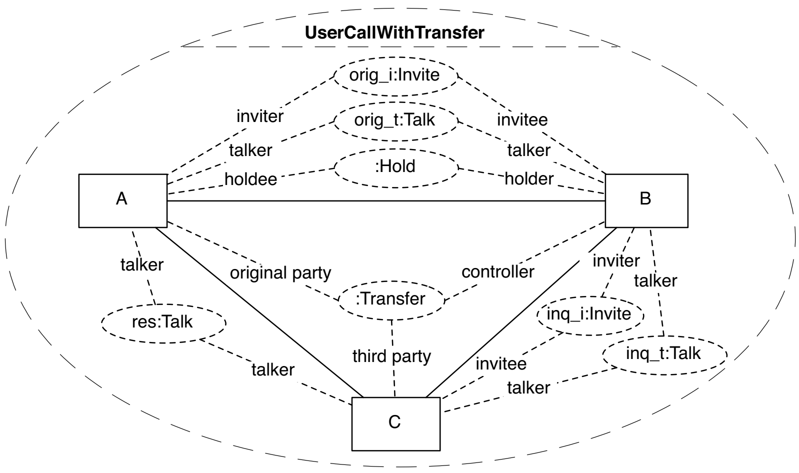


Fig. 9. The collaboration `UserCallWithTransfer`

Fig. 9 demonstrates how subordinate collaborations such as `Invite` and `Talk` may be reused in new settings, due to the flexible role binding of collaboration uses. Such reuse is a very attractive aspect of collaborations, and can help to give an intuitive understanding of a complex situation, as illustrated here. Call transfer is a classical challenge for service designers to understand and describe succinctly. From Fig. 9 it is apparent that several call invitations are involved. However, the precise ordering of the subordinate collaboration uses can not be understood from Fig. 9 alone. A goal sequence diagram for the `UserCallWithTransfer` service, as suggested in Fig. 10a, is one possibility of describing this.

Fig. 10a describes the ordering of collaboration uses required for the overall service goal of the transfer feature to be achieved. The goal sequence diagram combined with the collaboration diagram of the service (see Fig. 9) provides a compact and fairly intuitive description of a complex service. It has been common practice among telecom service engineers to make informal sketches to the same effect as an aid in service design. UML collaborations provide an opportunity to formalize and better support this practice. The goal sequence demonstrates how UML promotes reuse of units of behavior in the form of collaboration uses, and documents the evolution of the static structure depicted in the collaboration diagram. One particularly interesting aspect of the goal sequence diagram in Fig. 10a is that it shows situations in which a role, e.g. `B`, is simultaneously playing two or more sub-roles, e.g. `holder` and `inviter` in the fourth step of the sequence. Note that the simplicity of collaboration structures may be deceiving. Call transfer may look simple in Fig. 9, but when fully elaborated the underlying sequences and role behaviors can be quite complex.

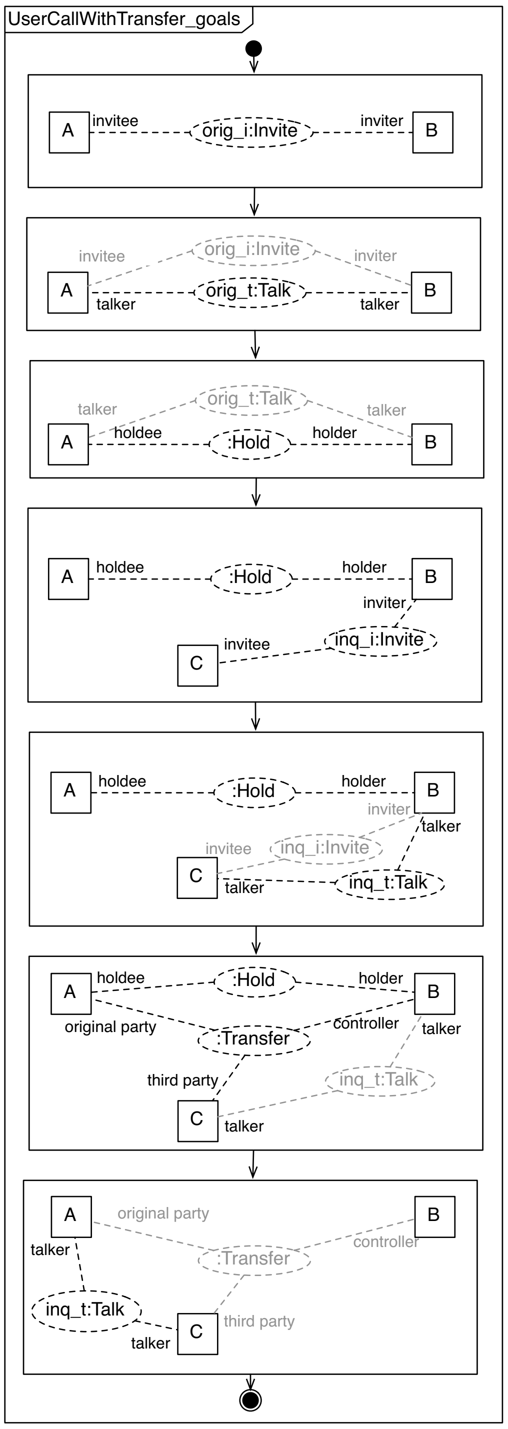
There are limits to what goal sequence diagrams are capable of expressing. For instance, it is not possible to describe goal dependencies among overlapping collaborations. This is the case, for example, of a `log-on` collaboration that requires a user authentication as part of its operation. It is desirable to model `log-on` and `authenticate` as separate collaborations to achieve reuse, and allow `log-on` to be combined with alternative authentication patterns. However, we cannot express with goal sequence diagrams that `authenticate` is enabled when `log-on` achieves a sub-goal, and that `authenticate` must achieve its goal before further progress in `log-on` is possible. An alternative notation, Use Case Maps [16], has been shown to have the necessary expressive power [5].

3.3 Towards Class Design

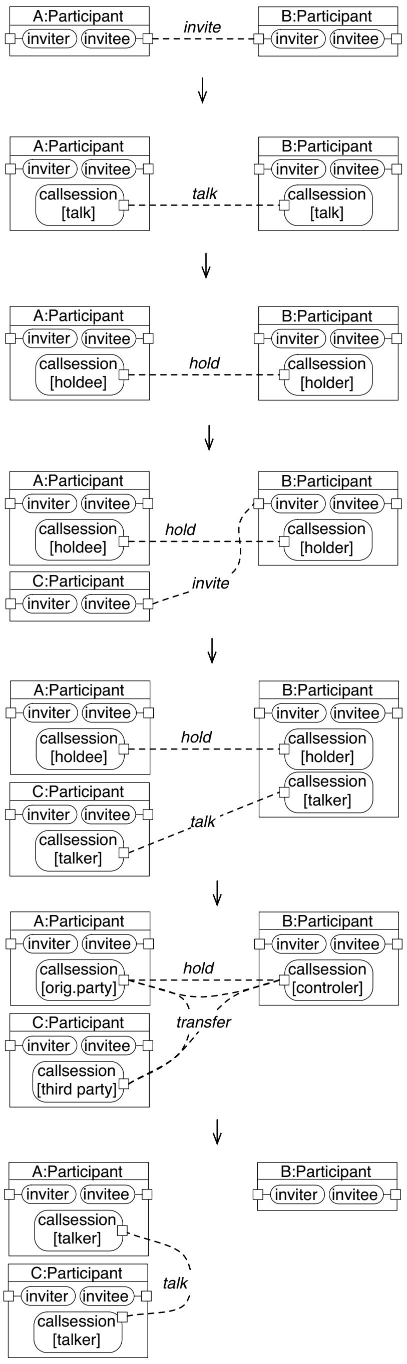
The specification of service functionality in collaborations is beneficial beyond the specification phase and can have direct influence on the design of classes and state machines. Analyzing the collaborations and the goal sequences tells us which roles a class must play over time, which requests for roles can arrive in which situations and which connections must be established to reach the goals of the implemented services. Modeling service specifications can help class design, as we now shall see.

Fig. 10b illustrates the coarse structure of a class `Participant` that implements all three roles `A`, `B` and `C` of `UserCallWithTransfer`. The sub-roles `invitee` and `inviter` are implemented as separate state machines, since call requests can arrive at any time. When a call request from another component is received, `invitee` creates a new instance of the state machine `callsession` to handle the request. The sub-roles `talk`, `hold` and `transfer` can be implemented by composite states inside `callsession`, as these roles are played alternately. The figure also illustrates the connections between the state machines of the components and how they evolve as the service progresses towards the achievement of its goal.

To complete class design one must consider all collaboration roles bound to the class. The `Participant` class, for example, may take part in several collabora-



(a) Goal sequence diagram



(b) Illustrated component structure

Fig. 10. Goal sequence for `UserCallWithTransfer` with related component structure

tions other than `UserCallWithTransfer`, as it is shown in Fig. 11. In that case `Participant` must be compatible with the four roles `ua`, `A`, `B` and `ub`, and class design must take this into account.

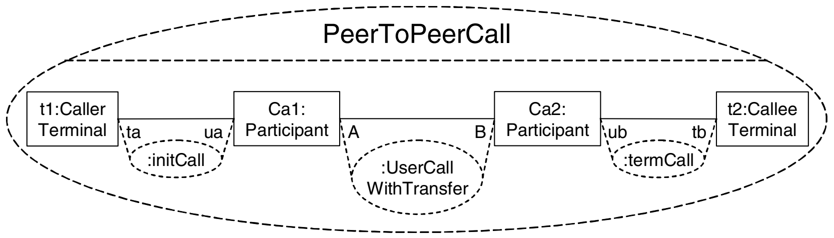


Fig. 11. Service composed of elementary services

4 Discussion

4.1 Related Work

The understanding that services involve collaboration between distributed components is not new; indeed, this was recognized since the early days of telecommunications. In terms of modeling the interaction of collaborations, various dialects of interaction diagrams existed prior to the first standardization of the ITU-T MSC language [17] in 1994. A slightly different approach was taken in the use cases of OOSE [18], where interactions were described textually. However, interactions alone do not really cover the structural aspects of the roles and the flexible binding of roles to classifiers.

Collaborative designs such as protocols have traditionally been specified by state diagrams, using combinations of informal descriptions and formal models, e.g. using SDL [19] or similar ([20, 21, 12]). But while state diagrams describe complete object behavior, the overall goals and the joint behavior tend to be blurred.

The concept of role was already introduced in the end of the 70's in the context of data modeling [22] and emerged again in the object-oriented literature. Using roles for functional modeling of collaborations was of primary concern in the OORAM methodology [23], and was one of the inputs influencing the UML work on collaborations in OMG. Within teleservice engineering it has been a long-standing convention to describe telephone services using role names like A and B. In [9] we classified different uses of the role concept, and pointed out that UML 1.x was too restrictive, since a *Classifier-Role* could bind to only one class, so they were not independent concepts that could be re-used in different classes.

Rössler & al. [3] suggested collaboration based design with a tighter integration between interaction and state diagram models, and created a specific language, CoSDL, to define collaborations [24]. CoSDL was aligned to SDL-96. Floch [6] also proposed a notation for collaboration structure diagrams, where components were designed in SDL-2000 [19].

With UML 2.0, it is now possible to model collaborations in a standardized language, increasingly supported by tools. Modeling collaborating services with UML 2.0 collaborations has earlier been suggested by Haugen and Møller-Pedersen [25]. They pointed out that there might be limitations in binding collaboration uses to classifier parts; these issues must be clarified, and binding to parts should preferably be supported. In the FUJABA approach described in [26], so-called coordination patterns are used for similar purposes as our semantic interfaces. They use a model checker to provide incremental verification based on the coordination patterns.

4.2 Further Work

A number of issues presented in this article need to be clarified and researched, and experiments in real projects must be undertaken before all problems are solved. We are currently applying these techniques on several practical service engineering cases including access control services, call control, and mobile information services. Compatibility rules between role classifiers and the objects and classes bound by collaboration uses is a semantic variation point in UML. The research on semantic interfaces [2] is a promising starting point for compatibility checking between complementary roles. Additional work on validating compatibility between roles and class designs, with tool support for composition, is being undertaken.

An experimental tool suite is currently being developed as part of the Teleservice Lab at the department of Telematics at NTNU, based on the Eclipse platform. The EU funded project *Semantic Interfaces for Mobile Services*, SIMS, to commence in 2006, will develop tool support for designing and validating collaborations, taking existing prototypes [27] as a starting point and validating the approach among industrial users.

5 Conclusion

This article has suggested ways of exploiting UML 2.0 for service engineering, and has discussed opportunities and limitations that lie in the current standard [7] in that respect. Our conclusion is that UML 2.0 collaborations seem to be a very useful expression form, as it allows one to define pieces of collaborating role behavior that can be bound to role players in a very flexible way.

Useful validation opportunities arise once criteria for role compatibility have been defined. Collaborations can be used to define semantic interfaces, which in turn can be used for compatibility checks and to support composition. We have argued for the inclusion of port state machines in UML as a more general description of semantic interface behavior than the existing protocol state machine mechanisms that have been defined in UML 2.0.

Furthermore we have suggested how minor notational enhancements can be introduced to represent collaboration situations in order to support high level feature composition; this is more of a tool issue than a language issue, but has methodological implications that are important. Finally, we have demonstrated how collaboration uses provide means to define complex multi-party services on a high level.

In contrast to the common practice of modeling complete service sequences involving all participating roles, our approach encourages decomposition into interface behaviors represented as two-way collaborations. The result is smaller and more reusable interface behaviors that can be validated separately, thereby addressing compositionality and scalability. The disadvantage is that behavior composition needs special attention, e.g. using goal sequences as elaborated in [5].

References

[1] Bræk, R., Floch, J.: ICT convergence: Modeling issues. In: Proc. of the 4th Int. SDL and MSC (SAM) Workshop, Ottawa, Canada, LNCS 3319, Springer (2004)

[2] Sanders, R.T., Bræk, R., von Bochmann, G., Amyot, D.: Service discovery and component reuse with semantic interfaces. In: Proc. of the 12th Int. SDL Forum, Grimstad, Norway, LNCS 3530, Springer (2005)

[3] Rößler, F., Geppert, B., Gotzhein, R.: Collaboration-based design of SDL systems. In: Proc. of the 10th Int. SDL Forum, Copenhagen, Denmark, LNCS 2078, Springer (2001)

[4] Sanders, R.T., Bræk, R.: Discovering service opportunities by evaluating service goals. In: Proc. of the 10th EUNICE and IFIP Workshop on Advances in Fixed and Mobile Networks, Tampere, Finland (2004)

[5] Castejón, H.N.: Synthesizing state-machine behaviour from UML collaborations and Use Case Maps. In: Proc. of the 12th Int. SDL Forum, Norway, LNCS 3530, Springer (2005)

[6] Floch, J.: Towards Plug-and-Play Services: Design and Validation using Roles. PhD thesis, Dep. of Telematics, Norwegain Univ. Sci. and Tech., Trondheim, Norway (2003)

[7] Object Management Group: UML 2.0 Superstructure Specification. (2004)

[8] Rumbaugh, J., Jacobson, I., Booch, G.: The Unified Modeling Language Reference Manual. 2nd edn. Addison-Wesley (2004)

[9] Bræk, R.: Using roles with types and objects for service development. In: IFIP 5th Int. Conf. on Intelligence in Networks (SMARTNET), Pathumthani, Thailand, Kluwer (1999)

[10] Sanders, R.T., Bræk, R.: Modeling peer-to-peer service goals in UML. In: Proc. of the 2nd Int. Conf. on Soft. Eng. and Formal Methods (SEFM'04), IEEE Computer Society (2004)

[11] Mencl, V.: Specifying component behavior with port state machines. Electr. Notes Theor. Comput. Sci. **101** (2004) 129–153

[12] Selic, B., Gullekson, G., Ward, P.T.: Real-Time Object-Oriented Modeling. John Wiley & Sons (1994)

[13] Floch, J., Bræk, R.: A compositional approach to service validation. In: Proc. of the 12th Int. SDL Forum, Grimstad, Norway, LNCS 3530, Springer (2005)

[14] Diethelm, I., Geiger, L., Maier, T., Zündorf, A.: Turning collaboration diagram strips into storycharts. In: Workshop on Scenarios and state machines: models, algorithms, and tools; ICSE'02, Orlando, Florida, USA. (2002)

[15] Floch, J., Bræk, R.: Using SDL for modeling behavior composition. In: Proc. of the 11th Int. SDL Forum, Stuttgart, Germany, LNCS 2708, Springer (2003)

[16] ITU-T Draft Recommendation Z.152: URN - Use Case Maps notation (UCM). (2004)

[17] ITU-T Recommendation Z.120: Message Sequence Charts (MSC). (2004)

[18] Jacobson, I., Christerson, M., Jonsson, P., Øvergaard, G.: Object-Oriented Software Engineering: A Case Driven Approach. Addison-Wesley (1992)

[19] ITU-T Recommendation Z.100: Specification and Description Language (SDL). (2002)

[20] International Organization for Standardization (ISO): Estelle: a formal description technique based on an extended state transition model. ISO9074. (1989)

[21] Harel, D.: Statecharts: A visual formalism for complex systems. Sci. Comput. Program. **8** (1987) 231–274

[22] Bachman, C.W., Daya, M.: The role concept in data models. In: Proc. of the 3rd Int. Conference on Very Large Data Bases, Tokyo, Japan, IEEE Computer Society (1977)

[23] Reenskaug, T., Wold, P., Lehne, O.A.: Working with Objects: The OOram Software Engineering Method. Prentice Hall (1996)

[24] Rößler, F., Geppert, B., Gotzhein, R.: CoSDL: An experimental language for collaboration specification. In: Proc. of the 3rd Int. SDL and MSC (SAM) Workshop, Aberystwyth, UK, LNCS 2599, Springer (2002)

[25] Haugen, Ø., Møller-Pedersen, B.: The fine arts of service modeling. Technical report, Internal report. ARTS (2003) http://www.pats.no/projects/ARTS/arts.html.

[26] Burmester, S., Giese, H., Hirsch, M., Schilling, D.: Incremental design and formal verification with UML/RT in the FUJABA real-time tool suite. In: Proc. of the Int. Workshop on Specification and Vaildation of UML models for Real Time and embedded Systems (SVERTS), associated with UML2004, Lisbon, Portugal (2004)

[27] Alsnes, R.: Role validation tool. Master's thesis, NTNU (2004)