Describing component collaboration using goal sequences

Cyril Carrez[1], Jacqueline Floch[2], Richard Sanders[2]

[1] NTNU,
Department of Telematics,
7431 Trondheim, Norway
carrez@item.ntnu.no

[2] SINTEF ICT,
7465 Trondheim, Norway
{jacqueline.floch, richard.sanders}@sintef.no

Abstract. Services are normally not performed by a single component, but result from the collaboration of several distributed components. Their precise specification and validation require complex models, where the *intention* of the service is easily lost in the detail. This paper exploits the concept of *service goals* that was earlier introduced to simplify service modeling. It describes the semantics of service goals, how to specify and how to use them. We show that so-called *goal sequences* can provide a designer-friendly, high-level description of the intention of the service, while maintaining simplicity, reusability and flexibility when composing from elementary services. By way of examples, we illustrate the difference between goal sequences and behavior descriptions. Finally we discuss issues related to the validation of goal sequences and their use at design time and runtime, for example in connection with service discovery.

Keywords: Goal sequences, collaborative components, high-level service specification.

1 Introduction

Ensuring interoperability in distributed systems has been a software engineering topic for decades. Recently the ICT community has rallied around the principles of a service oriented architecture (SOA) in order to address this challenge, see e.g. [1]. Within contemporary SOA, the composition approach called *choreography* is concerned with collaborative business processes involving multiple autonomous services, where different participants can assume different roles with different relationships. However, so far only informal specifications of service choreography have been suggested [2]. At the same time, semantic web services seek to characterize what a service can provide by offering means of expressing interfaces using Web Services Description Language (WSDL) [3]. Although WSDL aims at providing a formal definition of the interface to a service, it is restricted to a static description of operations and associated messages.

We have previously suggested the concept of a *service goal* to characterize the possible achievements of a service, and have shown how service goals can simplify service modeling in UML2 [4]. This article refines the semantics of service goals, which is one result of the EU IST project SIMS[1]. We have also suggested *goal sequences* as a means of expressing the intensions of a composite service [5], i.e. the intention of a choreography. In this article we argue for the merits of goal sequences by means of simple examples, and contribute with advances on how to model them. However, while goal sequences provide a designer-friendly overview, they do not specify everything. In this article we discuss in particular the difference between goal sequences and behavior.

SOA is increasingly gaining acceptance, influencing the way people understand and define services. However, there is a fundamental limitation of SOA as it is currently understood. In SOA, services are provided by a service provider to a service consumer. A service provider is normally a "passive object" in the sense that it never takes any initiatives towards a service user. *Collaborative services* on the other hand entail collaborations between several autonomous entities that may behave in a proactive manner and may take initiatives towards each other. This is typical for telecom services, but also for a large class of services such as attentive services, context aware services, notification services and ambient intelligence. In this paper we consider collaborative services, where multiple components interact to perform a composite service. This generalization allows for a wider class of services.

The structure of this position paper is as follows: in section 2 we present service goals and their semantics, showing how composite services are modeled from elementary services using UML2 collaborations, and how goals characterize so-called semantic interfaces. Section 3 presents goal sequences as an intuitive way of modeling the intention of composite services, similar to choreographies. In section 4 we discuss issues related to validation and composition at design time and at runtime. We also discuss related work, and finally conclude by drawing some perspectives.

2 Semantics of service goals

As proposed by Sanders et al. [4], services are modeled by UML2 collaborations [6]. We distinguish between *composite* and *elementary* collaborations, as shown in Fig. 1. Elementary collaborations specify partial service behaviors. They define a collaboration between exactly two parts, called *semantic interfaces*, as well as the service goals of the collaboration. Semantic interfaces specify interface behavior, while service goals (or *goals* for short) specify the desired outcome of that behavior; both are discussed in this section. Composite collaborations, on the other hand, specify the service roles implemented by components[2] that take part in the service. Composite collaborations are in fact composed of UML2 *collaboration uses*, where

[1] Semantic Interfaces for Mobile Services; see http://www.ist-sims.org

[2] We distinguish between the specification of a service, and its implementation. With that distinction in mind, we speak of service roles when specifying the service (at design time), while we speak of components when we execute the service implementation (at runtime). Service roles are depicted by an octagon in the composite collaboration.

each collaboration use is typed by an elementary collaboration. A service role can be bound to a number of semantic interfaces, which thus type its ports. For example, Fig. 1a specifies a service where a *Traveler* interacts with a *Hotel* and a *Plane* in order to plan a travel. The interactions are typed by the elementary collaborations *ReserveHotel* and *ReservePlane*.

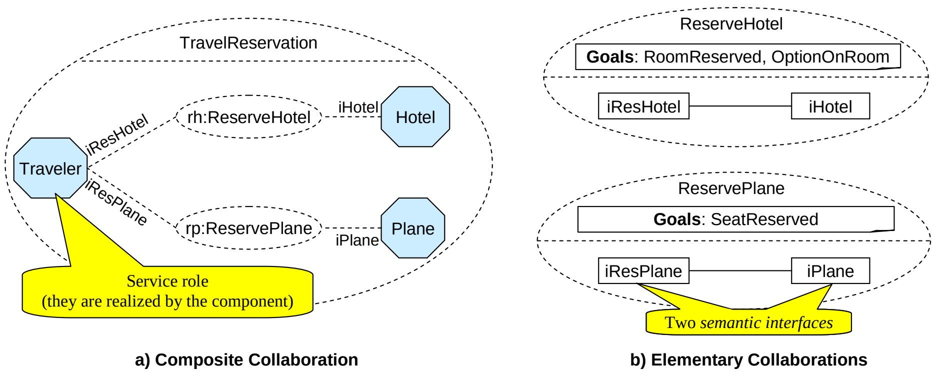

Fig. 1. Travel service modeled using collaborations and collaboration uses

2.1 Service goals and elementary collaborations

The elementary collaborations of Fig. 1b identify the goals reachable by each of them. Service goals do not define the behavior of an application, but rather the desired outcome of a behavior: they describe its *intention*. For example, concerning *ReserveHotel*, two goals can be achieved: *RoomReserved* or *OptionOnRoom*. Both are desirable outcomes of this micro-service. However, this does not mean that those goals must or will be achieved during an interaction between the *Traveler* and *Hotel*: possibly the hotel has no rooms left, meaning neither goal can be achieved.

Service goals were first proposed by Sanders [5, 7]. While Sanders described service goals using OCL expressions, we describe the goals using ontologies [8]. Ontologies allow us to describe the semantics of the goals (for instance “establish a multimedia call”), allowing flexible reasoning on goals and user-friendly descriptions.

We also differ from Sanders in the number of goals an elementary collaboration can achieve. Specifically we do not consider partial or sub-goals to describe a partial achievement in the collaboration. Several goals can be specified, but only one can be achieved during the execution of the elementary collaboration at runtime. This restriction was motivated by the desire to have a simple and intuitive specification when service goals are used during composition of a service (see section 3).

2.2 Service goals and semantic interfaces

While a goal characterizes the desired outcome of a behavior, the behavior itself is described by a semantic interface [9]. A semantic interface describes the visible behavior of a service role at a connection endpoint. Goals are attached to that behavior, allowing one to specify how a semantic interface can achieve a goal in a collaboration. Semantic interfaces type the ports of the service role, and are used to validate the composite service: when two service roles interact through ports, compatibility checks can be applied on complementary semantic interfaces to ensure a consistent interaction [10, 11].

Semantic interfaces are specified using UML state machines, with message passing semantics. Triggers and effects specify respectively a reception or a sending of a signal, thus specifying how to interact with the semantic interface[3]. We use a stereotype <<goal>> state to specify that the interaction has achieved a particular goal at this point. This way, goals represent "progress" in the behavior, and thus are a characterization of liveness. For example, Fig. 2 shows the state machine of the semantic interface *iHotel* of the elementary collaboration *ReserveHotel* presented in Fig. 1. One can ask for available rooms at specific dates, and either reserve the room and thus achieve the goal *RoomReserved*[4], or take an option on that room and achieve the goal *OptionOnRoom*. A <<goal>> state has exactly one outgoing transition, stereotyped <<transitionGoal>>: this transition is instantaneous. Goal states are represented by a dashed state symbol in Fig. 2.

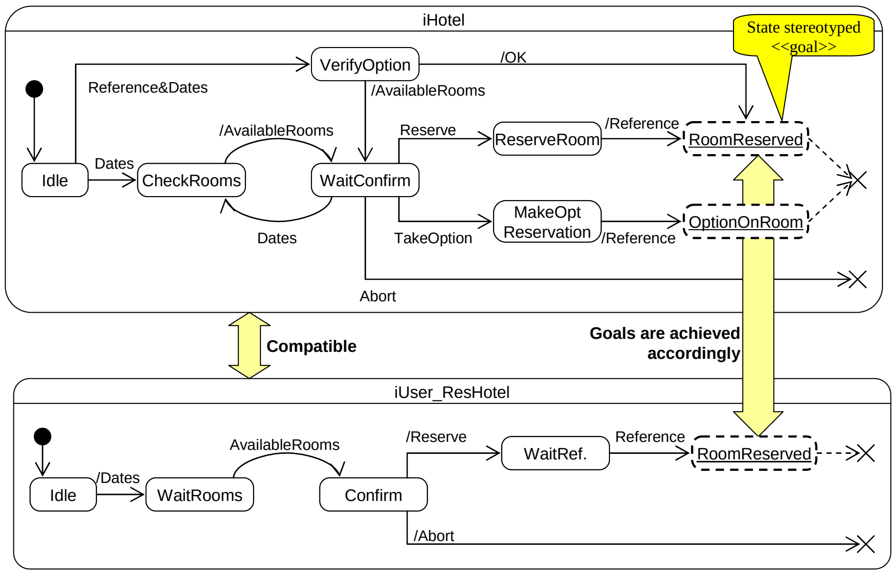

Fig. 2. Two compatible semantic interfaces, illustrating goal compatibility

We draw attention to two important issues regarding the goals and how they relate to the behavior of a semantic interface. First of all, different behaviors can lead to the same goal: for instance to achieve the goal *RoomReserved*, it is possible to ask for

[3] Parameters of signals are not taken into account.
[4] Payment of the room is performed by another elementary collaboration, as shown in section 3. For sake of simplicity, it is not included here.

available dates and reserve the room as the *iUser_ResHotel* does, or give the reference of an option on a room that was made earlier, and reserve the room if the option is still valid, shown in the upper part of the state machine of *iHotel*. Secondly, some behavior can still occur at the semantic interface after a goal has been reached, e.g. clean-up messages (for instance closing a session). Hence achieving a goal does not mean terminating a behavior.

The power of semantic interfaces lies in their use during composition. When two service roles interact, the connected semantic interfaces must be compatible, as we defined it in [11]: their interaction does not lead to unspecified message reception, deadlock, or improper termination, and their interaction is live. Concerning deadlock, we restrict ourselves to avoiding deadlocks between two semantic interfaces by ensuring that one of them will always be able to take action. By improper termination, we mean that both semantic interfaces should terminate accordingly. Finally, by live interaction, we mean they are capable of reaching a common goal. The compatibility relation is illustrated in Fig. 2, with the semantic interface *iUser_ResHotel* shown at the bottom. This semantic interface cannot make any option on a room, but is still goal compatible with *iHotel* as they can achieve the goal *RoomReserved*.

As a final point, all the entities we presented so far are elements of reuse: elementary collaborations, semantic interfaces and service roles can be reused in other composite collaborations, hence taking part in services they were not designed for in the first place. This reusability is illustrated through the examples of the article.

3 Goal sequences

So far we have shown how service goals describe the intention of partial service behaviors, and how they are related to elementary collaborations and semantic interfaces. When it comes to the service, service goals are composed in order to specify the intention of the whole service. This composition is specified by what we call *goal sequences*. Goal sequences were first introduced by Sanders [5]; in this article we propose a precise semantics allowing one to exploit them for validation purposes.

A goal sequence is a high-level specification which describes a desirable behavior, namely how goals depend on each other in terms of pre-conditions. As shown in section 4, they are used to verify that a composition of service roles is live (i.e. something useful may be achieved), or during service discovery. We distinguish between *Collaboration goal sequences* and *Role goal sequences*. The difference is that the former applies to composite collaborations and refers to goals of the elementary collaborations, while the latter applies to the service roles, and refers to the goals of its semantic interfaces. The principles presented in this section apply to both kinds of goal sequences; we will only discuss in length about collaboration goal sequences (here denoted goal sequences for short).

A goal sequence describes dependencies between the goals of the elementary collaborations that are used in a particular composite collaboration. They describe the *intention* of the composite service: that something useful can be achieved, and how it should be achieved (i.e. how the different elementary collaboration goals should be sequenced). We suggest that goal sequences are specified using UML Activity diagrams, where an activity represents a collaboration use[5] of the composite collaboration[6], and outgoing arrows represent the goals achieved by that collaboration use. Activity diagrams are very helpful for goal sequences, as several collaborations may execute in parallel. Moreover, activity diagrams are in line with the semantics of goal sequences: each activity represents a goal to be achieved. Sanders proposed interaction overview diagrams for goal sequences [4, 7]; however such diagrams currently lack tool support. We investigated using state diagrams in [11], which have more tool support, but they tend to get cluttered up when expressing parallel behavior in orthogonal states.

Fig. 3 shows the goal sequence for the *TravelReservation* presented in Fig. 1a. The two collaboration uses are represented by the two activities *rh* and *rp*. The goal sequence specifies the intention of the service, which is to reserve a room and a seat in a plane (goals *RoomReserved* and *SeatReserved*). We have deliberately chosen to drop the goal *OptionOnRoom*, as *TravelReservation* does not propose such an intention (in fact, *TravelReservation* is reusing *ReserveHotel* which may have been specified in another service). We define that it does not matter in which order the goals are achieved, as long as both of them can be achieved before the termination of the composite service. Note that this describes the intention of the service, and does not mean that each execution will actually achieve those goals: possibly the plane or the hotel is full. This example shows the primary advantage of goal sequences: it is easy to show the intention of the service. We believe that goal sequences are quite intuitive, and maintain simplicity during composition.

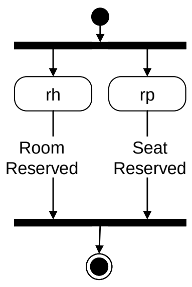

Fig. 3. Goal sequence for the composite collaboration *TravelReservation*

Fig. 3 shows one typical pattern for goal sequences, namely two goals that can be achieved in parallel. Fig. 4 shows patterns needed to specify different kinds of pre-conditions. The first one, on the upper left corner, shows the principle of goal sequences. In this pattern, two goals *g1* and *g2* are sequenced; the semantics is that the achievement of *g1* is a pre-condition for the achievement of *g2*. We say that *g1 enables g2*. As we shall see in section 4, this does not mean that *g1* enables the

[5] Recall that a collaboration use is typed by an elementary collaboration. Hence the goals of the collaboration use are the goals of the corresponding elementary collaboration. This way, an elementary collaboration can be used in many places in a composite collaboration.

[6] For role goal sequences, activities represent semantic interfaces.

collaboration *C2*, as the behavior of *C2* may start before or without *g1* being achieved. Boolean expressions AND and OR can also be specified in pre-conditions, as shown at the bottom of the figure.

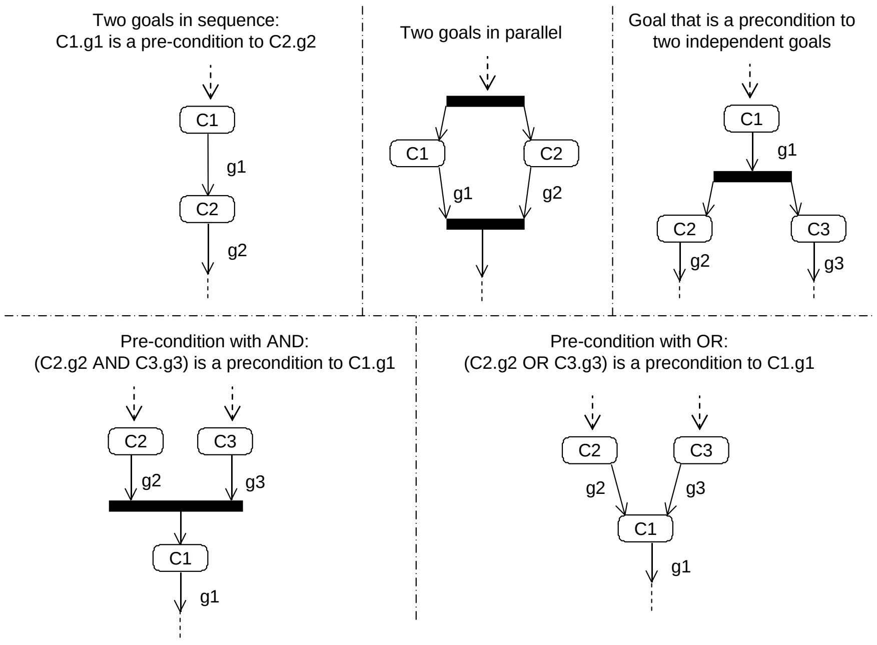

Fig. 4. Patterns for collaboration goal sequences

Goal sequences are very useful and intuitive when it comes to the design of collaborative services, i.e. when several participants can take initiative. For instance, Fig. 5 specifies a payment functionality when reserving a room to the hotel. As shown on the left of the figure, three service roles take part in the service: in addition to the *User* and the *Hotel*, there is also a *Bank*. Several collaboration uses demonstrate the composition of micro-services, most of them can be reused in different services: *Pay* and *ConfirmPayment* can be used in any service where money is involved. The goal sequence is shown in Fig. 5b: the room has first to be reserved, and then the user pays the bank, which in turn pays the hotel. Confirmation of payment and booking ends the service. Note, again, that the order of those two goals is of no importance.

Fig. 5 also shows the difference between the (collaboration) goal sequence and the role goal sequence: Fig. 5c is the role goal sequence for the *Bank*, which specifies how the service role should sequence the goals. We see the role goal sequence is in fact a subset of the collaboration goal sequence in Fig. 5b. Role goal sequences should not need to be specified by hand, but rather be derived automatically from the collaboration goal sequence.

A role goal sequence sets constraints on the behavior of a service role: the service role should sequence the goals of its semantic interfaces in the proper order. For instance, the *Bank* should be paid before it pays the booking of the room, which in turn should happen before it confirms payment to the *User*. Such constraints are one of the uses of goal sequences we discuss in the next section. However, role goal sequences do not specify precisely how to compose semantic interfaces: even though some goals should be sequenced, the service role could nonetheless interact in parallel on the associated semantic interfaces (see section 4.2).

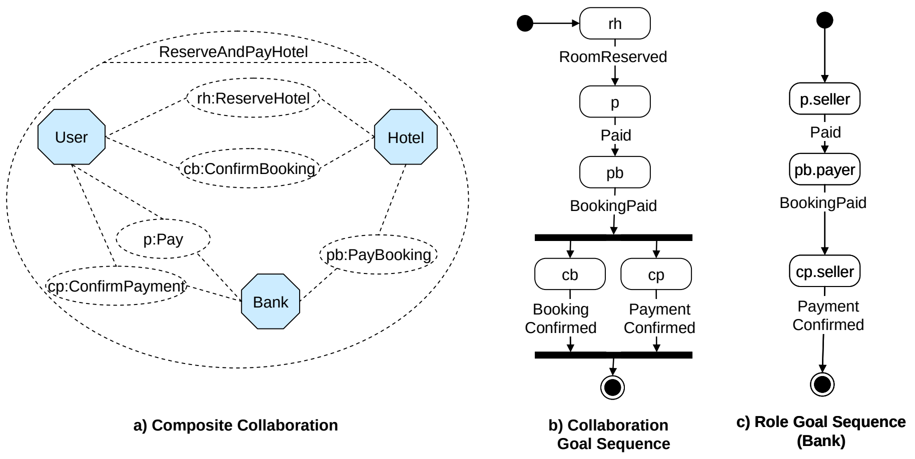

Fig. 5. Goal sequence and role goal sequence in a three-party service

4 Discussion

This section discusses several remaining open issues. We show how resolving them will enable validation of interoperability between components in a flexible manner. We first discuss the validation of goal sequences at design time, and how they can be used at runtime. We show that although they can be useful for service discovery, goal sequences are not sufficient to ensure safe composition.

4.1 Validation of goal sequences at design time and runtime

Goal sequences can be used to verify if a composition is live, meaning that the interconnected service roles are able to achieve something useful together. To ensure that the intention of the service is achievable, validation using tools can be performed; preferably this is done at design time, but if necessary it can be done at run time. The validation will ensure the correctness of a composition of service roles.

At design time, service roles can be validated against their semantic interfaces and the goal sequences. Projection and refinement mechanisms can be used in order to verify that the service role is compatible with the semantic interfaces [10, 11]. It should also be possible to check if the service role satisfies the pre-conditions on goals imposed by the role goal sequence, i.e. it sequences the goals of its semantic interfaces in the proper order.

Once service roles have been validated against service specifications (i.e. the collaborations, semantic interfaces and goal sequence), components that implement these roles can be developed[7] and deployed along with descriptors that describe their behavioral properties: semantic interfaces and role goal sequences.

At runtime, the descriptors can be used to validate a dynamic composition. Semantic interfaces can be used to check that two interconnected components are goal compatible, implying that they can achieve a goal together, for instance that a *User* and a *Hotel* can achieve the *OptionOnRoom* goal. The same principle applies to goal sequences: if components cannot sequence their goals correctly, then there is no use in starting a service session. E.g. if a particular *Hotel* requires payment before confirming a booking, then it is of no interest to a *User* that behaves according to the *ReserveAndPayHotel* service. However, several questions arise concerning such validation: given the role goal sequences of each component, is it possible to validate component collaborations on the fly in an efficient manner, i.e. so the validation can be performed by the device? Is it possible to automatically derive a collaboration goal sequence? If so, what will be the semantics of that goal sequence, i.e. the intention of the resulting service? Should it be presented to the user? If so, how?

4.2 Goal sequences and safe composition

While the previous section focused on the use of goal sequences to ensure a composition of service roles does something useful, we also need to take into account safety properties during composition. A component interacts through its semantic interfaces; some of them will be active when the component starts, while others will become active as a result of its own or external initiatives. A safe composition should make sure that if a component receives a signal on one of its semantic interfaces, it is actually ready to receive such a signal.

Unfortunately, goal sequences fall short in that area; it turns out that they only provide support for loose composition. As illustrated in Fig. 6, goal sequences do not specify how elementary collaborations are composed (i.e. in sequence or in parallel for instance). In this example, the *Boss* first asks his/her *Secretary* to plan a travel for him/her. The *Secretary* will reserve the *Hotel* and the *Plane*, and give the *Boss* a confirmation. The goal sequence shows that the elementary collaboration *PlanTravel* should not achieve its goal before the end, while in fact *PlanTravel* initiates the whole service.

In addition goal sequences are not well suited for detecting deadlocks. In the *ReserveAndPayHotel* (Fig. 5a), one should make sure that the three components will not be in deadlock, i.e. each one waiting for the other in a circular manner. However, as goal sequences do not describe temporal dependencies between behaviors, it is not possible to detect deadlocks using goal sequences alone. One should not aim at simply detecting the deadlock when it happens, but rather at detecting possible deadlocks *before* starting the service, i.e. detecting *deadlock-free configurations* of components.

[7] Components also implement some functionality related to their execution environment (e.g. underlying middleware for component registration, etc.)

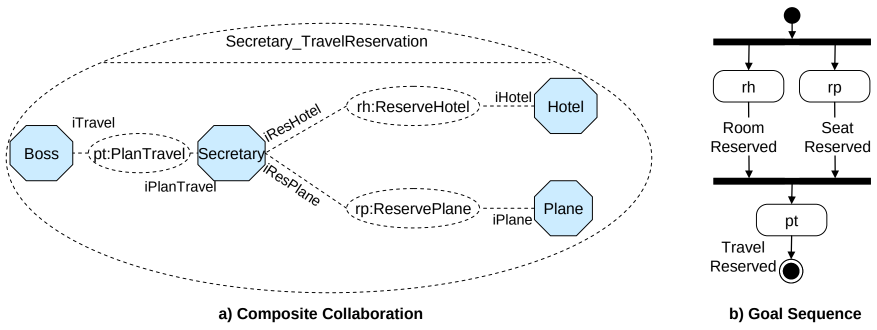

Fig. 6. Goal sequences and order of execution of elementary collaborations

4.3 Goal sequences in service discovery at runtime

Goals and Goal sequences can be exploited in service discovery at runtime. For instance a *Caller* may need to discover a *Callee* that is capable of using *Video*, and does not want to interact with a component that can only communicate via SMS.

At runtime, when a user starts a service, he/she will start some component on his/her device. This component will be involved in some service, which means it wants to discover compatible components in order to interact with them to provide some useful functionality to the user. This entails discovering components that have compatible semantic interfaces, and a compatible role goal sequence.

The discovery of compatible components can result in numerous configurations, as shown in Fig. 7. In this example, the *Traveler* wants to discover components that are compatible with its semantic interfaces and role goal sequence. Several configurations of components might be discovered, as shown on the left. Possibly some service providers have heard of this service, and developed a *TravelAgency* that performs the reservations, or the *Hotel* and the *Plane* may interact with each other to order a taxi; in all the cases, the goal sequences need to be compared and the resulting composition needs to be validated.

Without such validation, seemingly compatible but useless components might be discovered. Using goal sequences, we restrict the discovery to components that can potentially achieve the behavior intended by the user when he/she started the service. Moreover, we can take advantage of ontologies to first filter components that achieve the most appropriate goals.

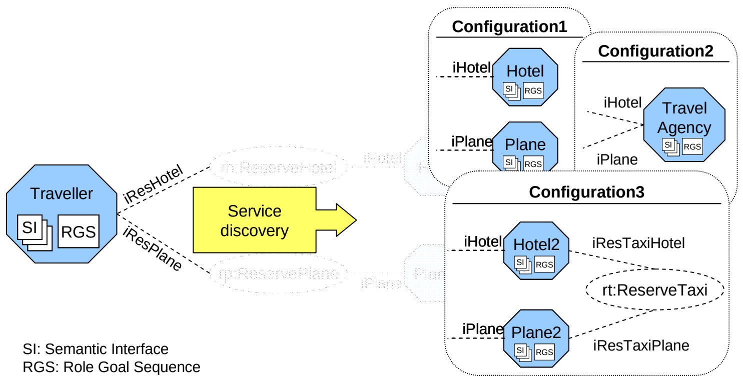

Fig. 7. Discovery of compatible components

5 Related Work

The understanding that services entail collaboration among several distributed autonomous components is not new. This was recognized since the early days of telecommunications, but is also typical for many new services such as attentive services, context aware services, notification services and ambient intelligence. In terms of modeling of collaborations, various dialects of interaction diagrams existed prior to the first standardization of the ITU-T MSC language in 1994 [12]. However, interactions alone do not really cover structural aspects nor provide flexible binding of interfaces to roles in the way now made possible using UML2 collaborations. While interaction diagrams provide a cross-cutting view of a service, they are often too detailed to be easily understood. Our approach abstracts the cross-cutting view on the service using collaborations and goal sequences, and describes the detailed behavior of interfaces using state machines.

In model driven development one strongly argues for developing abstract models that can be refined and transformed into implementation specific models [13]. Model driven approaches to service engineering are still in their infancy. Most of the UML-based approaches developed for service modeling focus on consumer-provider services. For example, Kramler & al. [14] propose to use UML2 collaborations for modeling web service collaboration protocols, and activity and interaction diagrams for more detailed specification. In the same way, Kraemer and Herrmann [15] specify reactive systems with UML2 collaborations for structural properties, and UML2 activities for behavioral aspects. However, the authors are more focused on design time, while we take advantage of service goals to discover useful compositions at runtime. Similarly Ermagan and Krüger [16] consider services to be collaborations between roles. They introduce a UML2 profile for the specification of service-oriented architectures. However they do not seem to exploit the capability of composition of collaborations (i.e. using UML2 collaboration uses). The definition of a UML Profile for services is an ongoing activity at the OMG. The responses submitted to the OMG RFP (request for proposal) “UML Profile and Metamodel for

Services (UPMS) RFP" [17] indicate that UML2 collaborations will gain importance in the future modeling of services. At the time of writing the submitted responses to UMPS are under discussion, and we are contributing to this work. A mechanism for expressing goals is one such contribution.

Goals have been extensively used in the engineering domain to capture, analyze, validate and document the properties a system should have [18, 19]. Similarly goals are proposed in service modeling to represent the properties desired by the user [20, 21]. While the term goal is a concept related to the user, capability is used in relation with the service and represents what the service does. In their conceptual service framework Quartel & al. [20] suggest that the definition of the user goal should provide a high-level description of the service, this to facilitate the discovery of services. They propose an abstraction level at which a service is modeled as a single interaction, that somehow matches an elementary collaboration in our work.

To the best of our knowledge, no one has used goal sequences before to represent the overall functionality of services and the dependencies between elementary collaborative behaviors. Goals associated to components and represented in the state machines are similar to progress labels introduced by Holzmann [22] and can be exploited to validate the liveness properties of interacting state machines. Related to our work and also building upon on [4], Castejón and Bræk extend the concept of goal sequences allowing a precise specification of services solely using collaborations and goal sequences (but not state machines) [23]. Their aim is to develop abstract service models that can be used for early detection of errors, such as implied scenarios. Their approach focuses on service composition at design time. Differently we consider discovery and composition at runtime and therefore need more simple service representations.

In the web service domain, intensive research work aims at the automation of service discovery and composition. Current web technologies operate at a syntactic level and therefore require human interaction. The Web Service Modeling Ontology (WSMO) is a result of that research effort [21, 24]. WSMO provides a formal language for semantically describing all relevant aspects of Web services. It defines the concepts of capability and goal that respectively relate to the Web service and the user. Capabilities include the semantic description of a variety of properties such as non-functional properties (e.g. financial or security aspects), pre- and post-conditions and interface behaviors. As a complementary concept, a goal includes the requested capability that the user expects from a service. Although detailed service descriptions are needed for precise discovery, unlike our goals WSMO does not provide any abstract description of services that would facilitate a quick initial discovery of potential, relevant services. The detailed interface behaviors, called choreographies in WSMO, are described using UML state machines in our work.

We have intentionally avoided replication between UML models and ontology artifacts. We do not define the semantics of each message using ontologies, but this could be done in the same way as for goals. Beyond discovery, WSMO also aims at facilitating service composition. It is not clear how this objective can be achieved as no support for describing temporal dependencies between composed services is provided. WSMO defines the concept of orchestration to describe how a service makes use of other services. This concept restricts to the hierarchical composition of services. WSMO does not provide support for more complex compositions such as collaborative composition. Collaborative composition is called choreography in Erl [1] and in the WS-CDL standard [2]. This use of the term choreography differs from WSMO where choreography is restricted to the definition of interface behaviors.

6 Conclusion and Perspectives

Systems modeling in high-level graphical design languages such as UML and access to advanced tools for validation, simulation and code generation has been available within certain engineering areas for quite some time, the telecoms domain being one that matured early in this respect, defining formal languages [12, 25]. It is therefore somewhat surprising that service engineering is still largely implementation-oriented without any clear separation between service logic and implementation detail. This is a paradox since service-orientation essentially means to focus on service specification and to hide the details of component design and implementation, allowing different realizations of the same service.

In this paper we argue for the benefits of characterizing partial service behaviors with goals, and of modeling them with elementary collaborations in UML2. A mechanism for expressing goals is currently being input to the upcoming UML profile and metamodel for services (UPMS).

Focusing on goals enables service engineers to design and analyze service composition at a high level; we argue for the merits of goal sequences as an intuitive description of the intention of service choreographies. We have discussed how goal sequences can benefit service discovery, while they fall short of being sufficient for comprehensive validation and automated composition. Solutions for dynamic composition and runtime validation require further work. However, there is much to be gained both at design time and for service discovery at runtime by abstracting away unnecessary implementation details.

Acknowledgements. Our work is supported by the EU IST 6th framework program.

7 References

1. Erl, T. Service-Oriented Architecture - Concepts, Technology, and Design. 6th ed. 2006. Prentice Hall. ISBN 0-13-185858-0

2. W3C. Web Services Choreography Description Language (WSCDL) Version 1.0 - W3C Candidate Recommendation - 9 November 2005. 2005.

3. W3C. Web Services Description Language (WSDL) Version 2.0 - W3C Recommendation - 26 June 2007. 2007.

4. Sanders, R., Castejón, H., Kraemer, F., Bræk, R.: Using UML 2.0 Collaborations for Compositional Service Specification. In: Proceedings of the 8th International Conference of Model Driven Engineering Languages and Systems, LNCS 3713, Springer (2005).

5. Sanders, R. and Bræk, R.: Modeling Peer-to-peer Service Goals in UML. In Proc. of the 2nd Intl. Conf. on Software Engineering and Formal Methods (SEFM04), IEEE Computer Society Press (2004)

6. Object Management Group: Unified Modeling Language: Superstructure version 2.1.1, formal/2007-02-05 (2007). Available at http://www.omg.org/cgi-bin/doc?formal/07-02-05

7. Sanders, R.: Collaborations, Semantic Interfaces and Service Goals: a way forward for Service Engineering. Doctoral theses at NTNU 2007:68. NTNU (2007)

8. SIMS: Deliverable D3.4 Techniques for Ontology-Driven Semantic Interface Artefacts, final version. (2007). Available at http://www.ist-sims.org/

9. Sanders, R., Bræk, R., Bochmann, G., Amyot. D.: Service Discovery and Component Reuse with Semantic Interfaces. In Proc. of the 12th Intl. Conf. on Model Driven Systems Design (SDL Forum 2005). Springer (2005)

10. Floch J.: Towards Plug-and-Play Services: Design and Validation using Roles. PhD Thesis 2003:47. NTNU (2003)

11. SIMS: Deliverable D2.1 Languages and Method Guidelines, first version. (2007). Available at http://www.ist-sims.org/

12. ITU-T Recommendation Z.120: Message Sequence Charts (MSC). (2004)

13. Mellor, S., Clark, A., Futagami, T.: Special Issue on Model-Driven Development. IEEE Software 20(5) (2003)

14. Kramler, G., Kapsammer, E., Kappel, G., Retschitzegger, W.: Towards Using UML 2 for Modelling Web Service Collaboration Protocols. In Proceedings of the First International Conference on Interoperability of Enterprise Software and Applications (2005).

15. Kraemer, F. A., Herrmann, P.: Service Specification by Composition of Collaborations – An Example. In Proc. of the 2nd Intl. Workshop on Service Composition (Sercomp). IEEE Computer Society (2006)

16. Ermagan, V. and I.H. Krüger, I.H.: A UML2 Profile for Service Modeling. In Proceedings of the 10th Intl. Conf. of Model Driven Engineering Languages and Systems (2007)

17. OMG. UML Profile and Metamodel for Services (UPMS) RFP - soa/06-09-09. Available from: http://www.omg.org/cgi-bin/doc?soa/2006-9-9

18. Lamsweerde, A.: Goal-oriented requirements engineering: A guided tour. In Proceedings of the 5th IEEE International Symposium on Requirements Engineering. (2001).

19. Yu, E.: Towards modelling and reasoning support for early phase requirements engineering. In Proceedings of the 3rd IEEE Intl. Symposium on Requirements Engineering. (1997).

20. Quartel, D. A. C., Stehen, M. W. A., Pokraev, S., and van Sinderen, M. J.: COSMO: A conceptual framework for service modelling and refinement. Information Systems Frontiers 9(2-3). (2007)

21. Roman, D., et al.: Web Service Modeling Ontology. Journal of Applied Ontology, vol 1. (IOS Press). (2005).

22. Holzmann, G.J. Design and Validation of Computer Protocols. 1991. Prentice Hall. ISBN 0-13-539834-7

23. Castejón, H. N. and Bræk, R. A Collaboration-based Approach to Service Specification and Detection of Implied Scenarios. ICSE's 5th Workshop on Scenarios and State Machines: Models, Algorithms and Tools (SCESM'06), 2006.

24. Web Service Modeling Ontology (WSMO). D2v1.3. WSMO Final Draft 21 October 2006.

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