View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by CiteSeerX

A Collaboration-based Approach to Service Specification and Detection of Implied Scenarios

Humberto Nicolás Castejón and Rolv Bræk
Department of Telematics
Norwegian University of Science and Technology
N-7491 - Trondheim, Norway
{humberto.castejon, rolv.braek}@item.ntnu.no

ABSTRACT

Methods for service specification should be simple and intuitive. At the same time they should be precise and allow early validations to be performed, in order to detect inconsistencies as early as possible in the service development cycle. In this paper we present a service specification approach based on UML 2.0 collaborations. It aims to be a constructive approach, rather than a corrective one, as it is intended to promote understanding and help reducing the number of specification errors. We also address the detection of implied scenarios from collaboration-based service specifications, and propose an approach that limits the state explosion problem. This is possible since the detection analysis is modular and it is performed at a high-level of abstraction.

Categories and Subject Descriptors: D.2.1 [**Software Engineering**]: Requirements/Specifications - Methodologies; D.2.4 [**Software Engineering**]: Software/Program Verification - Validation

General Terms: Design, Verification

Keywords: Service specification, UML 2.0 collaborations, roles, goals, goal sequences, implied scenarios.

1. INTRODUCTION

One of the challenges of service engineering lies in service specifications and the mapping from specifications to design components. It is desirable to specify services in ways that are as simple and intuitive as possible. At the same time specifications should be as precise and complete as possible, without unduly binding the design and implementation issues. It is also important that service specifications can be analysed and refined towards components that can be dynamically discovered and composed.

Many authors have identified the cross-cutting nature of services (e.g. [8, 4]) and the need to specify and analyse services using interaction diagrams (e.g. UML sequence diagrams or MSCs). We have found UML 2.0 collaborations[1] [7] useful to structure and analyse service specifications. UML 2.0 collaborations are intended to describe partial functionalities involving interactions among participating roles played by objects. Therefore, they fit well with an understanding of services as collaborations between service roles played by objects that deliver functionality to the service users. A service can be specified as a collaboration defining a structure of collaborating roles with associated behaviour. Collaborations can, in turn, be used in the definition of larger collaborations, by means of collaboration uses. This feature enables a compositional and incremental specification of services. Moreover, collaboration uses provide the desired flexibility to bind service roles to different classifiers, and provide a means to structure complex collaborations and give an overview of the service, while at the same time being precise.

Most approaches to service specification using interaction diagrams are based on scenarios describing interactions among a set of components/agents involved in a service. While this provides the desired cross-cutting overview, the result is frequently too detailed to be easily understood. An alternative approach is to organise interaction diagrams according to service goals [9] and interfaces, so that they actually describe interface behaviour. This will often result in a larger set of smaller diagrams and a reduced cross-cutting overview, which must be compensated by alternative means to precisely relate (i.e. compose) the interface behaviours involved in the service. This might be considered as a drawback. However, we believe the benefits of this approach outweigh its disadvantages. In the first place, although decomposition of the service does not eliminate the complexity of its specification, it undoubtedly helps to better understand the different service sub-problems [3]. Moreover, treating explicitly the composition of interface behaviours contributes to the comprehension and explicit documentation of their dependencies, which in turns helps with the understanding and discovery of potential conflicts. Finally, smaller diagrams describing service features on interfaces are more reusable and support an easier evolution of the service. We will see how collaborations, in conjunction with *goal sequences* [9, 11], provide an opportunity to structure services into features and to specify their dependencies, and at the same time provide the desired cross-cutting overview.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
SCESM'06, May 27, 2006, Shanghai, China.
Copyright 2006 ACM 1-59593-085-X/06/0005 ...$5.00.

[1]Collaborations in UML 2.0 are considerably different from their predecessors in UML 1.x, which are now called communication diagrams in UML 2.0.

A second challenge of service engineering lies on early validation. Analysis of service descriptions in search of inconsistencies and conflicts should be performed as earlier as possible, in order to reduce the cost in time and effort needed to eliminate the problems. In this paper we show how collaboration-based specifications can be easily analysed to find implied scenarios [1], which are unexpected scenarios that have not been captured in the specification, but that will be present in any service implementation.

1.1 Contributions and outline

The contributions of this paper are mainly two. We present a constructive approach to the specification of services based on UML 2.0 collaborations, and show how specifications created in this way open interesting opportunities for the detection of implied scenarios.

The paper is structured as follows. In section 2 we briefly discuss the use of UML collaborations for service engineering. We then detail their use for the specification of services in section 3. The proposed approach for the detection of implied scenarios is presented in section 4, before we end with related work and some discussion in sections 5 and 6.

2. UNDERSTANDING COLLABORATIONS

According to the UML 2.0 standard [7], collaborations describe a structure of roles that collaborate to collectively accomplish some task. Rather than describing specific object instances, roles specify the properties that any object instance must have in order to participate in the collaboration. Collaborations can be bound to specific contexts by means of collaboration uses, which specify the binding of sub-roles to (composite) roles or classifiers.

In [11] we discussed the suitability of UML collaborations for service engineering. When used in this context, collaborations describe the structure of the service in terms of a structure of roles, and describe the interactions of these roles as part of the service. Service goals [9], expressed in terms of service predicates, can be associated to collaborations as a means to express liveness properties. Two types of service goals can be described: *event goals,* denoting desired events; and *state goals,* which are properties of global collaboration states that we wish to reach, and which entail combinations of role goals. A service collaboration may be decomposed into a number of two-way collaborations defining semantic interfaces in terms of visible interface behaviour and goals [11]. These semantic interfaces may be further decomposed into sub-collaborations corresponding to phases or features provided across the interface. This kind of decomposition lends itself to behaviour analysis and refinement towards design classes having well defined interfaces, which can be used for dynamic discovery and compatibility validation using semantic interfaces as explained in [10].

A positive side-effect of decomposing the behaviour of a service into small collaborations is that the dependencies between the resulting collaborations must be explicitly captured in order to address their composition. Note that the behaviour of an atomic collaboration (i.e. without sub-collaborations) can be directly described by means of a set of interaction diagrams, while the behaviour of a composite collaboration follows from the composition of the behaviours specified for its sub-collaborations. Therefore, only if sub-collaboration composition relationships are well described, the service behaviour will be well specified.

Collaborations can be inter-related in a number of different ways. They can be independent, so their executions interleave, they can enable each other (i.e. sequential dependency), or they can interrupt each other. Finally, collaborations can maintain goal dependencies, as we discussed in [2]. This happens when the outcome of a collaboration (i.e. its goal) depends on the outcome of other collaboration(s). A goal dependency normally implies a dependency between interfaces, such that the interactions happening on one interface i (as part of collaboration *C1*) are temporarily suspended to perform interactions on another interface j (as part of collaboration *C2*). Upon completion of the operations on interface j, collaboration *C1* on interface i is resumed.

We use *collaboration goal sequences* to describe collaboration dependencies. They show the evolution of services in terms of sub-collaborations and their goals. That is, they capture the liveness aspects of services by describing the intended ordering of sub-collaborations at runtime, and by showing at which points these sub-collaborations interact. Goal achievement is used to specify *interaction points* between collaborations, hence the name *collaboration goal sequences.* These sequences were first introduced in [9, 11]. In this paper we modify and extend their original notation and semantics in order to describe suspend/resume relationships between sub-collaborations.

3. SERVICE SPECIFICATION WITH COLLABORATIONS

In this section we describe an approach to the specification of services from early requirements, by means of UML collaborations. It consists of five steps that take the designer gradually from a more abstract specification to a more detailed one. A direct consequence of this is that the understanding of the service increases at each step, so that when some aspects of it have become clearer, it may be necessary to take a step back and modify the previous specification. The proposed approach is therefore iterative in nature.

We will explain and demonstrate the approach using a transportation case study that originally appeared in [12], and whose requirements are detailed in the next section.

3.1 Case Study: A Transportation Service

The transportation service consists of one high-speed vehicle transporting one passenger at a time between two terminals. In order to travel, a passenger must buy a ticket at one of the terminals. If the vehicle is at that terminal, the departure gate is indicated and the passenger can enter the vehicle. The terminal then initiates the departure procedure by communicating to the vehicle its destination. When this procedure is finished, the vehicle starts the journey and, when it is 100 meters away from the destination terminal, it initiates the arrival procedure by sending an alert to the terminal. Once the vehicle is at the destination, the passenger disembarks.

If the vehicle is not initially at the terminal where the passenger buys the ticket, the terminal requests the vehicle from the control center. The center then relays the request to the other terminal, which sends the vehicle using the departure procedure. Once the vehicle arrives at the requesting terminal, the departure gate is communicated to the passenger and the service continues as explained above.

3.2 The Specification Approach in Detail

Our methodology for service specification has five steps.

Step 1: *identify the main roles of the service under specification.* These roles follow from the problem domain. For example, for the transportation case study we can identify four service roles: *Passenger* (P), *Terminal* (T), *Vehicle* (V) and *Control-Centre* (CC).

Step 2: *define the collaborations that each service role have with any of the other roles.* We are interested in two-party collaborations (any multi-party collaboration can be decomposed in several two-party collaborations), where the roles describe interface behaviour. The identified collaborations should have (at least) one concrete and meaningful goal, that we could express in terms of predicates over properties of the collaboration. In our case study, we realize that the *Passenger* role can be involved in three collaborations: it has to interact with *Terminal*, in order to buy a ticket and get the gate displayed; and with *Vehicle*, in order to enter to and exit from the physical vehicle. We name these collaborations *BuyTicket*, *EnterVehicle* and *ExitVehicle*, respectively. Note that each of them has two roles, which can be understood as sub-roles played by *Passenger*, *Terminal* and *Vehicle* (e.g. in the *BuyTicket* collaboration, *Passenger* plays P_{bt}, while *Terminal* plays T_{bt}). If we continue analyzing the requirements, we can identify four other collaborations: *ReqVehicle* and *OrderVehicle*, between *Terminal* and *Control-Center*; and *VehDeparture* and *VehArrival*, between *Terminal* and *Vehicle*. Note that we can associate a well-defined goal with each of the identified collaborations. In the case of *BuyTicket*, for example, the main (state) goal is for *Passenger* to obtain a ticket and for *Terminal* to sell it. This can be expressed as an OCL predicate of the form: $ticketBought = P_{bt}.ticketBought\ AND\ T_{bt}.ticketSold$. In addition, we know that *BuyTicket* may need to be suspended, after the ticket is requested and before the gate is displayed, in order to request the vehicle from the control center. We therefore define a *ticketReqed* event goal associated with the ticket requesting message, which will be used as an interaction point between *BuyTicket* and *ReqVehicle*.

Step 3: *specify the service as a single composite collaboration.* In this step we represent the sub-collaborations defined in Step 2 as collaboration uses and bind them to the service roles that were identified in Step 1. Figure 1 presents the UML 2.0 collaboration diagram for our transportation service. This diagram shows the structure of the service (i.e. the main roles, their multiplicity and their interconnections) and its decomposition into smaller service features. It also states how the binding of sub-roles (i.e. roles of sub-collaborations) to main service roles is performed. For example, it specifies that *BuyTicket*'s role T_{bt} is bound to the service role *Terminal* (T) (i.e. T_{bt} is a sub-role of *Terminal*).

Step 4: *describe dependencies by means of a collaboration goal sequence.* Describing the service as a collaboration (cf. Step 3) enables a high level overview of the service, but the specification is still incomplete. We miss a description of the dependencies between the sub-collaborations of the service, which determine their composition and execution order. In this step we describe these dependencies by means of a collaboration goal sequence, as illustrated in Figure 2 for the transportation service.

Goal sequences, as presented in this paper, are inspired by UML activity diagrams. Each activity (round-cornered rectangle) represents a phase in the execution of the service

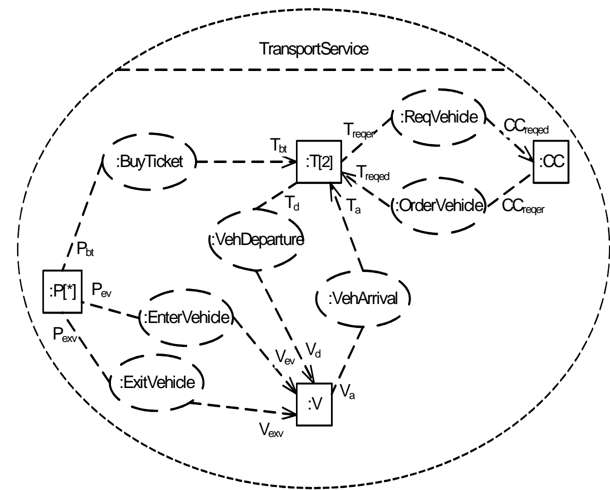

Figure 1: Transport service specified as a UML 2.0 collaboration (Step 3 of the specification approach)

collaboration the goal sequence is associated with. Each activity is annotated with a collaboration use that is enabled during the service phase that the activity represents. We refer to this collaboration use as the active collaboration of the activity. Optionally, activities may be annotated with collaboration uses that are temporarily suspended and/or waiting for inputs from the active one. In order to differentiate between these two kinds of collaboration uses, we draw the active one in black and the passive one in grey. We also adorn suspended collaboration uses with a round-cornered rectangle showing the state the collaboration use is suspended in (see, for example, the activity on the upper right corner of Figure 2, where *r:ReqVehicle* is the active collaboration use and *b:BuyTicket* is a collaboration use that is suspended at state *ticketReqed*). Annotating activities in this way helps to see the sub-roles that a given service role plays at each service phase, as well as the execution order of these sub-roles.

As in UML activity diagrams, we use a token-passing semantics to describe the flow of control among activities. That is, when an activity receives an input token, its active collaboration is enabled. Thereafter the collaboration can begin or resume execution, when one of its roles takes the appropriate initiative, and evolve until an interaction point with other collaborations is eventually reached. This could imply that the collaboration has finished its execution, or that it needs to be suspended (e.g. due to a goal dependency with another collaboration). For the sake of differentiating these two cases, we represent interaction points with exit points of two different types. A crossed-circle exit point ($\otimes$) is used for interaction points representing end of execution (e.g. as in the last activity of the goal sequence in Figure 2), while an empty-circle exit point ($\bigcirc$) is used for interactions points representing suspension (e.g. as in the first activity of the goal sequence in Figure 2). Exits points have are annotated with conditions expressed in terms of predicates describing event/state goals. According to our semantics, an activity passes on the control token to a subsequent activity when the condition of one of its exit points is satisfied. As we have already mentioned, when an activity

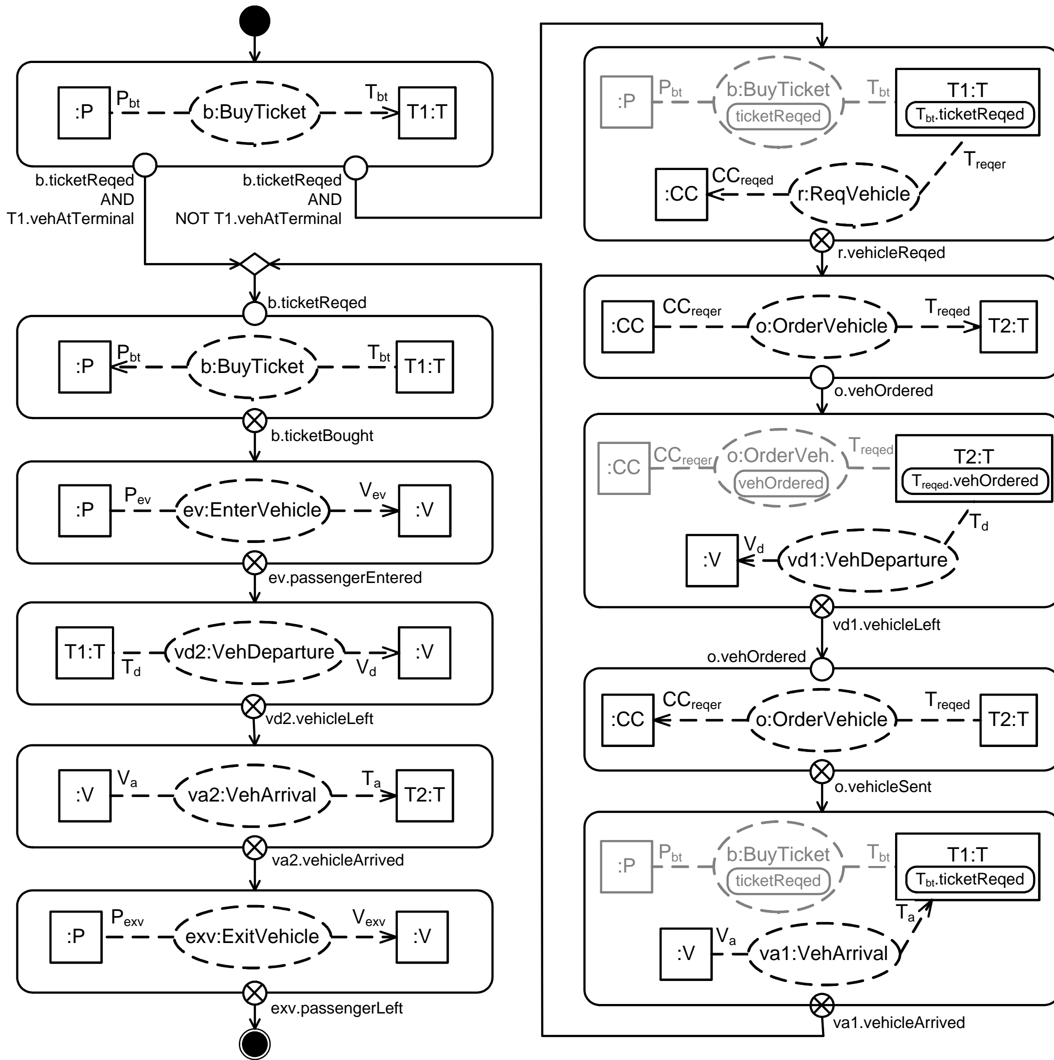

Figure 2: Goal sequence for *TransportService*

receives the control token its active collaboration can either begin execution or resume it, if it had been previously suspended. In order to differentiate between these two cases, we use, or do not use, entry points. When an activity does not have an entry point, its active collaboration starts execution from the initial state. However, if an entry point (i.e. an empty circle: ◯) is used, the active collaboration resumes execution from the state represented by the entry point's condition. Edges (i.e. directed connections between activities) and control nodes (i.e. decision/merge, fork/join, initial and activity final nodes) are respectively used to allow and coordinate the flow of control among activities. An activity may have several outgoing edges, while we restrict the number of incoming edges to only one (so multiple incoming edges must be AND- or OR-joined).

Step 5: *detailing the behavior of the collaborations.* This is the final step of our methodology and consists on describing the behaviour of the sub-collaborations by means of sequence diagrams. These diagrams are adorned with goal information contained in continuations. Note that continuations in the sequence diagrams correspond to interaction points (i.e. entry/exit points) in the goal sequence. Figure 3 shows some of the sequence diagrams for the case study.

After Step 5 the specification of the service is finished, and we can use it as input for the design process, where state machines for the roles will be built. This could be done automatically by using the synthesis algorithm that we presented in [2][2]. However, the specification may contain inconsistencies and implied scenarios arising from the concurrency of role executions. It is preferable to detect those inconsistencies and implied scenarios at the specification level, before the design starts, since the earlier we do it, the smaller the cost of eliminating them is. In the next section we discuss how this can be performed.

[2]A simple modification to work with collaboration goal sequences, instead of Use Case Maps, would be necessary.

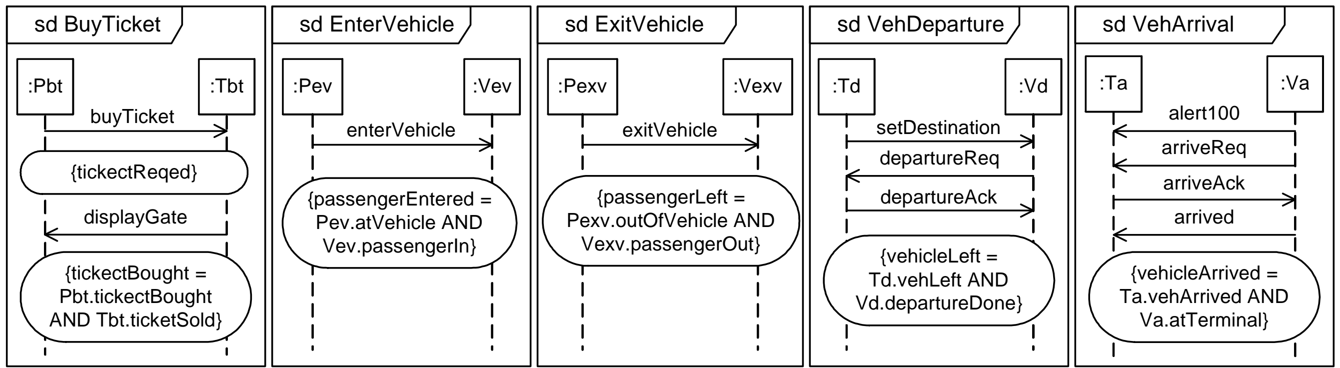

Figure 3: Interaction diagrams for the `TransportService`'s sub-collaborations

4. SERVICE SPECIFICATION VALIDATION: DETECTION OF IMPLIED SCENARIOS

In this section we show how the use of collaborations, goals and goal sequences open some interesting possibilities for the validation of service specifications in search of implied scenarios. These scenarios correspond to service behaviour that has not been explicitly described in the service specification, but that will be present in any implementation of it [1]. Implied scenarios are a direct consequence of concurrency in systems. Having this in mind, we propose to search for implied scenarios by analysing concurrency in the service's goal sequence.

An important aspect in our analysis methodology is the classification of (sub-)roles as either *initiating* or *offered*[3]. In a collaboration the *initiating role* is the one taking the initiative to start the collaboration, while the *offered role* is the one accepting the initiative. In collaboration diagrams and goal sequences we distinguish offered sub-roles by means of an arrow-head pointing to the bound role (note that this is not standard UML). This can be appreciated, for example, in Figure 1, where: T_{bt}, T_a and T_{reqed} are offered sub-roles played by *Vehicle*; V_{ev}, V_{exv}, V_d are offered sub-roles played by *Terminal*; and CC_{reqed} is the only offered role played by *Control-Center.*

We base our detection approach on two facts: (i) a role cannot control when it will be requested to play an offered sub-role (although it is its decision to play or not to play the requested sub-role); and (ii) the execution of offered sub-roles will normally trigger the execution of other sub-roles, that is, a sequence of sub-roles will normally be played following the request of an offered sub-role. With this in mind we can easily realise that unexpected scenarios can arise if a role is requested to play a certain offered sub-role, while already "busy" serving a previous request. In order to detect these unexpected scenarios we analyse the sub-role sequences that service roles execute as part of the service.

Sub-role sequences can be directly derived from the collaboration goal sequence by projecting it onto one specific (instance of a) service role. This is done by traversing each possible path of the goal sequence looking for occurrences of the same role. Once all occurrences are localized, the sequence of sub-roles played by that role is extracted, together with entry/exit points conditions expressed in terms of role goals. Note, however, that in collaboration goal sequences the conditions on entry/exit points are expressed in terms of collaboration goals. Therefore, we need to somehow extract role goal information from collaboration goals. For this purpose we look at the sequence diagrams built on Step 5 of the specification approach. The continuations in these diagrams specify how the collaboration goals are decomposed in role goals (e.g. the last continuation in the *BuyTicket* sequence diagram specifies that the *ticketBought* collaboration goal is decomposed in role goals *ticketBought* and *ticketSold*). Figure 4 shows a selection of sub-role sequences for our case study. Figure 4a shows the two sub-role sequences that *Vehicle* can play. The first one corresponds to the case where *T1.vehAtTerminal* is true, while the second corresponds to the case where *T1.vehAtTerminal* is false. Figure 4b shows two of the sub-role sequences that *Terminal* can play. The first one represents the sub-roles played by *T1:T* when *T1.vehAtTerminal* is true, while the second sequence considers the sub-roles played by *T2:T* when *T1.vehAtTerminal* is false.

Sub-role sequences are subject to two separate analyses. First we consider them in isolation (i.e. each of them individually) and, thereafter, we turn our attention to how they interact with each other.

In the isolated analysis we consider each sub-role sequence separately, and search for possible inconsistencies due to concurrency aspects and/or non-determinism. In this paper we just focus on concurrency conflicts, which can appear when the sub-role sequence contains two or more consecutive offered roles. In this case, a conflict may exist if the offered roles are played in collaborations with different parties, and these collaborations maintain some kind of dependency (e.g. one collaboration should not finish before the other does). In such a situation it might not be possible to ensure that the dependency between the collaborations is respected, since the initiatives to start them are taken by different parties. For example, a conflict of this type can be detected in the two sub-role sequences of *Vehicle.* According to them, role V_{ev} is to be played before role V_d. This corresponds to the initial requirement that the vehicle departs (i.e. *VehDeparture* is executed) after the passenger has embarked (i.e. after *EnterVehicle* is executed). This behaviour cannot be ensured since *Terminal*, which takes the initiative in *VehDeparture*, cannot know if *Passenger* has taken the initiative

[3]The classification of roles into "initiating" and "offered" is due to Richard Sanders.

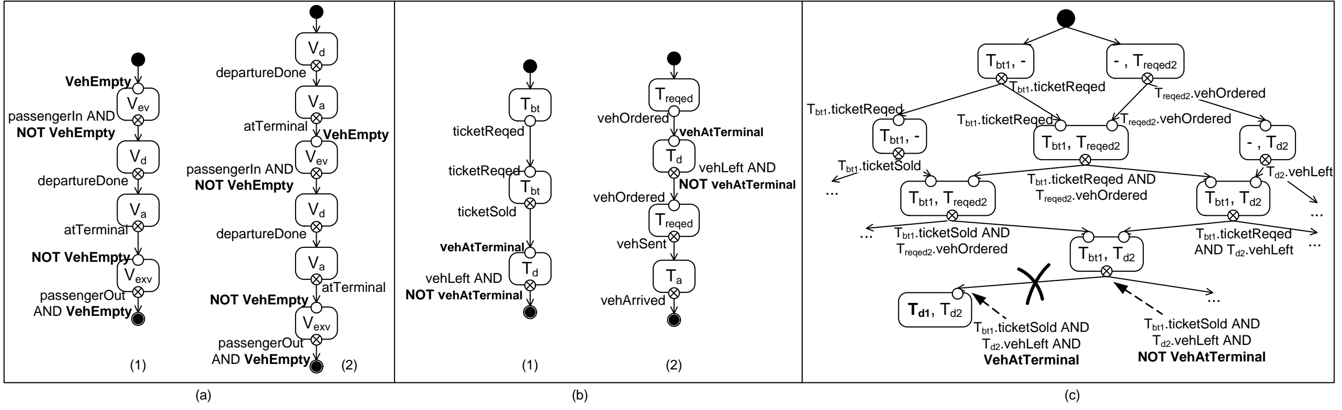

Figure 4: Sub-role sequences for (a) *Vehicle* and (b) *Terminal*; (c) Cross-product of *Terminal*'s sequences

to start *EnterVehicle*, and when this has finished (i.e. the condition *ev.passEntered* is not visible for *Terminal*). Thus *Terminal* may initiate *VehDeparture* before *Passenger* has initiated *EnterVehicle*. This conflict may be solved by modifying the service specification (e.g. by changing the behaviour of *VehDeparture*). Alternatively, we may postpone its resolution to the design phase. Note that it might be tempting to design *Vehicle* in such a way that, when involved in a *VehDeparture* collaboration, it would wait for *EnterVehicle* to start (if not yet started) and finish. However, this would be a wrong decision, since the collaboration goal sequence tells us that *VehDeparture* does not only occur in conjunction with *EnterVehicle*, but it also maintains a sequential dependency with *OrderVehicle*. This dependency might easily have passed unnoticed without the dependency information and overview provided by the collaboration goal sequence.

After analysing sub-role sequences individually, we study how they interact if executed concurrently. For that purpose, we first impose extra constraints on the execution of roles. These constraints follow from the requirements and the service domain, and help to further specify when roles can be executed. For example, in our case study we have further restricted the execution of role T_d by including *VehAtTerminal* and *NOT VehAtTerminal* as part of its pre-condition and post-condition, respectively (see Figure 4b, with added constraints in bold). After this assignment of constraints, we build the cross-product of the sub-role sequences, and search for points where constraints are violated. These points correspond to arcs in the cross-product that connect an exit point and an entry point with non-matching conditions.

Cross-product construction is done on a role basis. That is, for each service role we build the cross-product of its sub-role sequences, without taking into account sub-role sequences of other roles. Note, however, that not all the sub-role sequences of a service role need to be used in the cross-product. From our understanding of the service we can discard sequences that cannot happen concurrently with others.

Figure 4c shows an example of cross-product construction and conflict detection for our transportation service. This cross-product has been built from the two *Terminal*'s sub-role sequences presented in Figure 4b[4]. We can see that a conflict between the executions of role T_d as part of the "buying" sequence and as part of the "requesting" sequence ((1) and (2), respectively, in Figure 4b) has been discovered (see bottom of Figure 4c). This conflict happens because after the occurrence of T_d as part of the requesting sequence, *VehAtTerminal* is set to false, which prevents T_d from being executed, next, as part of the buying sequence. Now we can use the path leading to the conflict to build a scenario showing the high-level reason and/or consequence of the conflict. With this scenario we realise that the terminal may get a request for the vehicle from the control center (i.e. requesting sequence) while a passenger is buying a ticket (i.e. buying sequence). If this happens, the passenger may miss the vehicle, which would depart empty obeying the request from the control center. Further analysis of the cross-product would reveal the possibility of the opposite case, that is, that the vehicle may depart with the passenger before the control center's request had been completely processed.

Following the same procedure with *Vehicle* as with *Terminal*, we may detect an implied scenario where a requested vehicle arrives at the requesting terminal already carrying a passenger, and the new passenger tries to enter the vehicle before the occupying passenger has disembarked.

5. RELATED WORK

Service-oriented specification has been addressed in several works. Rößler et al. [8] suggested collaboration based design with a tighter integration between interaction and state diagram models, and created a specific language, CoSDL, to define collaborations. CoSDL is inspired by SDL, so it fails at providing the high-level service vision offered by UML collaborations and goal sequences. Krüger et al. [4] propose an approach to service engineering that has many commonalities with our own. They consider, as we do, services as collaborations between roles played by components, and use a combination of Use Cases and an extended MSC language to describe them. Liveness is expressed by means of the operators provided by their MSC language, while service structure and role binding are described with, so-called, role and deployment domain models. In our approach UML

[4]For the sake of clarity only the conditions associated to some of the entry and exit points are depicted.

collaboration diagrams are used to provide a unified way of describing service structure and role bindings, and to provide a framework for expressing liveness with goal sequences. Goal sequences provide interesting opportunities for analysis, as we have discussed.

Goals have been extensively used in the requirements engineering domain [5]. However, to the best of our knowledge, no one has used goals before to drive the decomposition of services into smaller services or features, and to express dependencies between them.

The concept of implied scenarios was first introduced by Alur et al. in [1], where they presented an algorithm to detect this kind of scenarios from MSC specifications. This work was later extended by Uchitel et al. [12], who proposed an approach for the incremental specification (using both MSCs and HMSCs) of systems, driven by the detection of implied scenarios. The main drawback of Uchitel et al.'s work is, however, that it is subject to the state explosion problem (although they limit it by applying heuristics). Munccini [6] has proposed an approach for the detection of implied scenarios based on the analysis of HMSCs. This work builds over a previous work of Uchitel et al., and avoids the state explosion problem. Our method also limits the state explosion problem and it is applicable to UML collaboration-based specifications, while Munccini's approach applies to HMSC-based specifications.

6. DISCUSSION AND CONCLUSIONS

We have presented a service specification approach based on UML 2.0 collaborations. It aims to be a constructive approach, rather than a corrective one, where feature dependencies are explicitly documented. To achieve this we propose a goal-oriented understanding of the service, and a subsequent decomposition of the service into features with concrete goals. We have suggested the use of goal sequences to describe dependencies between these service features. Goal sequences provide an intuitive understanding of the dynamics (i.e. liveness) of the service.

We have also addressed the detection of implied scenarios from collaboration-based service specifications. The detection is done by static analysis of individual sub-role sequences, as well as by constructing their cross-product. In spite of that, the proposed analysis is not significantly affected by the state explosion problem. The main reason is that the sub-role sequences of each service role are analysed separately, so the complexity is linear with the number of service roles. In addition, we work at a high-level of abstraction (i.e. we analyse role sequences and not message sequences). Moreover, not all the sub-role sequences of a service role need to be used in the cross-product, as we have previously explained. In the case study we have presented, we needed to construct two cross-products: one for *Terminal*, with only 20 states after discarding some sub-role sequences (or 240 states considering all sequences); and the other for *Vehicle*, with 35 states. As a comparison, the detection method by Uchitel et al. [12], which is of exponential complexity with the number of service roles, needs to build a safety property for the presented case study of 4414 states, if heuristics are used.

The proposed implied scenario detection approach demonstrates, in addition, that we have much to gain from the explicit description of features dependencies, and from the analysis and understanding of concurrency on interfaces.

There is still much work to do before we can consider the proposed approaches as complete solutions. We are working on the formalization of goal sequences, as well as on the formalization of the implied scenario detection algorithm. This will allow us to provide tool-support for goal sequences and automatic analysis of service specifications. Another interesting issue we plan to address is how to address the elimination of the implied scenarios. One possibility might be to specify negative goal sequences (as the the negative scenarios in [12]), but we need to study this more carefully.

7. ACKNOWLEDGEMENTS

We would like to thank our colleagues Cyril Carrez, Frank Krämer and Richard Sanders for helpful comments and discussions. We would also like to thank the anonymous reviewers for their interesting suggestions.

8. REFERENCES

[1] R. Alur, K. Etessami, and M. Yannakakis. Inference of message sequence charts. In *22nd Intl. Conf. on Software Engineering (ICSE'00)*, pages 304–313, 2000.

[2] H. N. Castejón. Synthesizing state-machine behaviour from UML collaborations and use case maps. In *12th Intl. SDL Forum, LNCS 3530*, pages 339–359, 2005.

[3] M. Jackson. Problems, subproblems and concerns. In *Early Aspects 2004: Aspect-Oriented Requirements Engineering and Architecture Design*, 2004.

[4] I. H. Krüger, D. Gupta, R. Mathew, P. Moorthy, W. Phillips, S. Rittmann, and J. Ahluwalia. Towards a process and tool-chain for service-oriented automotive software engineering. In *ICSE'04 Workshop on Software Engineering for Automotive Systems (SEAS)*, 2004.

[5] A. van Lamsweerde. Goal-oriented requirements engineering: A guided tour. In *5th IEEE Intl. Symposium on Requirements Engineering (RE'01)*, page 249, 2001.

[6] H. Muccini. Detecting implied scenarios analyzing non-local branching choices. In *6th Intl. Conf. Fundamental Approaches to Software Engineering (FASE'03), LNCS 3530*, pages 372–386, 2003.

[7] Object Management Group. *UML 2.0 Superstructure Specification*, August 2005.

[8] F. Rößler, B. Geppert, and R. Gotzhein. Collaboration-based design of SDL systems. In *10th Intl. SDL Forum, LNCS 2078*, pages 72–89, 2001.

[9] R. T. Sanders and R. Bræk. Modeling peer-to-peer service goals in UML. In *2nd IEEE Intl. Conf. on Software Engineering and Formal Methods*, 2004.

[10] R. T. Sanders, R. Bræk, G. von Bochmann, and D. Amyot. Service discovery and component reuse with semantic interfaces. In *12th Intl. SDL Forum, LNCS 3530*, pages 85–102, 2005.

[11] R. T. Sanders, H. N. Castejón, F. A. Kraemer, and R. Bræk. Using UML 2.0 collaborations for compositional service specification. In *ACM/IEEE 8th Intl. Conf. on Model Driven Eng. Languages and Systems (MoDELS), LNCS 3713*, pages 460–475, 2005.

[12] S. Uchitel, J. Kramer, and J. Magee. Incremental elaboration of scenario-based specifications and behavior models using implied scenarios. *ACM Trans. Softw. Eng. Methodol.*, 13(1):37–85, 2004.