Validation Aspects of Automatic Service Composition

Mazen Malek Shiaa and Jan Ove Fladmark

Department of Telematics, NTNU university
malek@item.ntnu.no, janovefl@online.no

Abstract. The paper studies the validation aspects of service composition, in particular the goal-based aspects. By service composition we are targeting the automated composition of service components that fit certain demands from a portfolio of available service components – typically referred to as service repository. It is arguable that the use of ontologies and semantic annotations of service components constitute an intelligent way to enhance service discovery and service composition mechanisms. However, validating the intention of a certain composition – goals – and verifying its requirements is still a topic for research development. This paper presents a simple approach to validate service compositions based on goal annotations. The effectiveness of this approach is highlighted through a simplified service example.

Keywords: semantic annotations, automatic service composition, validation, service platforms.

1 Introduction

The main focus of this paper is to study, and develop validation functionality for the service creation environment, which is part of the SPICE service platform being developed by the SPICE consortium [1]. SPICE (Service Platform for the Innovative Communication Environment) is a research project aimed at addressing the design, developing and putting into operation efficient and innovative mobile service creation/execution platforms for networks beyond 3G. The SPICE project vision is to design, develop and prototype an extendable overlay architecture that supports easy and quick service creation/execution of intelligent and ambient-aware services for the above mentioned networks.

The SPICE service platform supports the composition of services from simple service components and from other composite services. The SPATEL language is used to describe both the structure and behaviour of the services. This language does also support annotation of functional and non-functional descriptions of the services like goals, effects, preconditions, Input/Output parameters, and QoS attributes. These annotations are central to SPICE services and used both by the service creation and execution environment. The service validation in this paper will therefore focus mainly on how these annotations, and the goal annotation in particular, can be used when validating a service. The target of this work will be an Eclipse plug-in for validating certain aspects of a SPICE service. The service example used in this paper will

E. Di Nitto and M. Ripeanu (Eds.): ICSOC 2007 Workshops, LNCS 4907, pp. 258–267, 2009.

be the main service scenario of the E-tourism scenario being widely experienced and demonstrated within the SPICE project.

The work accomplished in [2 and 3] is very important with regard to validation aspects and goal-based analysis. [2] has focused on expressing and validating basic safety properties, while [3] focused on modeling and validating services using service roles and service goals – thereby liveness properties. The approaches presented in [2 and 3] have helped us to establish our vision of a goal-based validation approach. However, as the SPICE initial architecture and the applied service composition paradigm brings a lot of new concepts and practices in the service development lifecycle these approaches cannot be applied – fundamentally they use different service model with different service properties and features. After establishing our approach, which is presented here in this paper, we have the perception that certain features of these approaches may be applied. The work of [3] with regard to goal assignments and goal sequencing could in particular give a hand to whatever goal-based expressions and sequencing to be applied in the SPICE platform.

The paper will be structured in the following way. Sec. 2 will provide a short overview of the service development in the SPICE platform, while Sec. 3 will look at the main core of this paper; the validation aspects. Sec. 5 will look with some details at the example service system we handle. Sec. 6 will briefly explain the how the demonstration of our approach is being carried out, before giving some conclusion remarks in Sec. 7.

2 Service Development in SPICE

The Service Creation Environment (SCE) in SPICE is a set of tools for automating as much as possible the service creation process [4]. The creation process is defined from different view points: the basic component developer, the service developer and the end user. In this paper we will only consider the service developer view point. Eclipse is the proposed platform on which SCE will be built. The rationale for this is because it is widely used, and extensible by means of a plug-in architecture.

SPATEL (SPICE advanced service description language for telecommunication services) is a specialized service description language used to describe and develop services for the SPICE platform [5]. SPATEL cover both functional and non-functional aspects of a service. In a service creation environment this language will make the developer able to refer to both abstract and running services by these descriptions, and in the future also by the means of ontology. The semantic notations are an important part of SPATEL. Services (service components), service methods and parameters etc, can all have a number of annotations describing them such as:

- **Input/Output parameters** for a service
- **Goals** describe the overall objective of services or methods exposed by the service
- **Effects** describe outcome of methods with regard to state of the component
- **Preconditions** describe conditions that must be satisfied in order to allow execution of a method
- **Non-functional features** describe features like QoS, cost etc.

Ontologies are used to give a semantic description of these annotations. The semantic description will help the developer in building services, and can also be used by the Service Execution Environment [6] in automating the usage of service components.

Annotations on goals describe the overall objective of a service (e.g. goal:Flight-Booking) and/or the specific objective of a method (e.g. goal: CancelBooking); they enable semantic service discovery [5]. Annotations on goals could reside both at the service level and at the method level. The goal of a service component with regard to a composite service is widely denoted as sub-goal. Annotations on goals are always required and if an annotation on goal lacks for an operation it is implicitly assumed that it coincides with the service one (that in this case MUST exist).

3 Validation Aspects

Service, or protocol, validation is a rather wide area of research and a lot of approaches and solutions exist. There has been a lot of influence of Web Services and the Service Oriented Architecture (SOA) [7] concepts in the SPICE project and its methodology. One should look into the specification and the validation efforts, if any, in these fields. At first glance it seems that the focus of these service architectures is on the observable, or consumable, inputs and outputs of the service, e.g. those defined by the WSDL document of the service. WSDL documents describe the ports and operations of the web service, however it is less descriptive when it comes to the behaviour of the web service. Hence when considering validating a web service based on the WSDL, or whether a client can successfully consume a web service, the focus is on the static interface of the service.

In telecommunication there has been more focus on validating the behaviour of the services. Due to the distributed nature of telecommunication there has been a long history of using methods and languages like MSC and SDL (lately roles, goals collaborations and semantic interfaces in UML have also been used) for describing interactions between entities and the state machines of entities in the system. Simulation can then be used to validate the systems. Other methods of validating the services or interaction between the entities are by using state space exploration (using state transition graphs) as well as safety (e.g. no system deadlock and no unspecified reception occur) and liveness (i.e. something good inevitably happens) analysis [8 and 9].

Validation in system development is the process of ensuring that we are building the right system in terms that the system is meeting the requirements and expectations of the owner and users of the system. Validation is a process that encompasses the whole development process, including requirements, design, implementation, and testing (see [10] and [11]). The process of testing whether the system meets is specification is often called verification. The term validation is often used instead of verification for these system tests - we use the definitions for these terms as listed in [11]:

Verification: *To establish the truth of correspondence between a software product and its specification*

Validation: *To establish the fitness or worth of a software product for its operational mission*

Safety and liveness property validation is fundamental to any validation functionality of state machine specifications[1]. The behaviour of a SPICE component is defined by the means of a SPATEL state machine which describes the full behaviour of the component towards the environment and possibly other service components used in the composition. The service components used in the composition have their own state machines which again can interact with other service components. In the classical sense performing a safety or liveness analysis, through the construction of the global state space and then by certain reachability analysis, could quickly build up the so-called state space explosion and therefore complicates the validation process.

Our discussion on validation aspects in this paper will only focus on the composite services in SPICE and the required service goals. We look at a composite service as a SPATEL description, and hence a goal annotation is provided for it. This goal annotation is the goal of the service which might be reached when a user or another component is interacting with the service. Further the composite service is interacting with other SPICE services through its orchestration. These other SPICE services have their own goals that might be reached through the collaboration with the orchestration (or the orchestrator component – note that a SPATEL composite service specification always include such an orchestrator component). Validation of the composite service will look at whether the goal of the composite service could be reached through the orchestration. A number of questions need to be answered during the course of the validation:

1. *How can the goal of the composite service be expressed in terms of the goals of service components involved in the orchestration?*
2. *How to verify that the sub-goals are reached in the composition?*
3. *How to verify that the sub-goals are reached in the correct order and with the correct parameters?* [2]

4 Goal-Based Validation Approach

We are proposing a simple approach as a basis to answer the questions presented in the previous section. This approach will further enhance the automatic composition in the SPICE service platform. Our approach consists of three main phases:

1. *Semantic analysis phase*: during this phase a given service request will be semantically analyzed. This phase results in a list of goals to be achieved by a composition – we call this list *GoalList*. This analysis phase may be performed automatically, on the natural-language request provided by the end-user, or manually by the service developer.

[1] Safety properties can be formalized as properties on states or assertions about conditions to be met, while liveness properties are most conveniently formalized as invalid temporal sequences of events [8].

[2] Even though the discussion will focus on composite services, the tasks involved in validating whether a single goal is reached during the collaboration between a user or component and a single SPICE service component is similar to validating whether a single sub-goal is reached in the composition.

2. *Pre-validation phase*: in this phase the designer provides a list of goal-based expressions to be validated by the validation tool.
3. *Validation phase*: this phase will reason about the expressions provided by the user assuming a set of composition alternatives and goal ontology are given.

In Fig. 1 a simple diagram illustrates our approach to goal-based validation of automatic service composition. Basically the validation tool in our SPICE platform (which is considered an add-on to the service creation environment) obtains two sets of inputs: composition alternatives (specified in SPATEL) and certain goal expressions to be validated. The tool refers to the used goal ontology and produces validation results that show whether a goal expression is met by these composition alternatives or not.

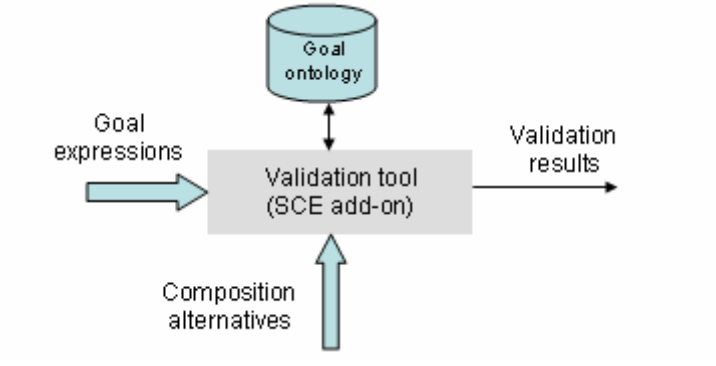

Fig. 1. Goal-based validation of automatic service composition

Goal expressions could be denoted using various mathematical/algebraic notations, e.g. regular expressions. We choose to use simple Boolean and pseudo-code expressions. The following examples give quick look:

- *GoalA* = *GoalB* AND (*GoalC* OR *GoalD*)
- *GoalA* = *GoalB* THEN *GoalC*
- *GoalA* = 2(*GoalB*) THEN LOOP (1..Max) *GoalC*
- NOT {*GoalB*, *GoalC*}
- *etc.*

These expressions are self-explanatory and are being constantly worked out and experienced with in our demonstration platform. In these expressions *GoalA* stands for the goal of a composite service.

In this paper we are arguing that reasoning about goals of a given composition needs to be approached with goal sequences (sequencing) in mind – this is obvious given the fact that goal-based analysis is aiming for liveness properties. Goal sequence means an expression of how several goals to be achieved in sequence. If a list of goals (or sub-goals of the service components constituting a composite service) is assumed to comprise the overall goal of a composite service then their ordering is equally crucial. These goals may be achieved either interleaved, sequential or both.

Based on this argument, we propose to further annotate the goals of a composite service with goal sequences. The annotation of goal sequences is basically a UML activity graph that shows the alignment of goals with respect to a SPATEL specification. The next section will present a simple example of such diagram.

5 The Service Example

Our service example is a simple service that finds restaurants of certain types at certain locations. A typical service request to trigger the composition process is:

"*Find a Chinese restaurant at MyLocation*"

Such request could be generated by the end-user in natural language or could be considered by the service developer – in both cases we assume the semantic analysis phase to produce a list of goals that constitute a subset of the used goal ontology.

In Fig. 2 we show a subset of the used goal ontology that corresponds to the goals that correspond to the service request shown above. Generally, such ontologies (expressed in OWL in our case) define concepts, e.g. *FindRestaurant*, and relationships between them necessary in the reasoning process.

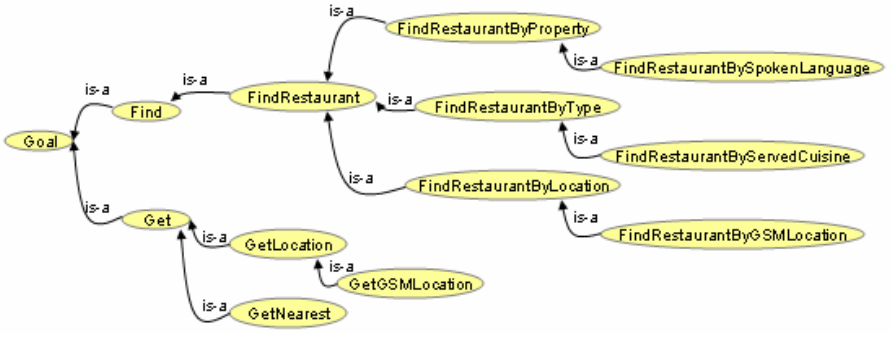

Fig. 2. Subset of the used goal ontology – our example *GoalList* would be: {*FindRestaurant*, *FindRestaurantByType*, *FindRestaurantByServedCuisine*, *GetLocation*, *GetGSMLocation*, *GetNearest*}

These goals mostly have association with finding the proper restaurant – all the sub-goals under *FindRestaurant*. *GetLocation* and *GetGSMLocation* are goals that reflect obtaining the location of the user via the used device, e.g. mobile terminal. The last goal in this ontology subset is *GetNearest*, which achieves the exploration of a nearest choice out of a list, e.g. restaurant list, given the user location.

The service platform is supposedly built around a set of mechanisms that facilitate the process of (automatic) service discovery. The essential part of such process is how to search the repository of existing (and available) service components – note that we are assuming the SPICE architecture and its discovery facility [4].

In Fig. 3 we show a very simplified set of service components and their semantic annotations as suggested by the SPATEL notation. This set of service components is

only shown for demonstration purpose – some of them have been implemented as part of the E-tourism scenario. As shown in the figure the semantic annotation is limited to the goals of the methods of these service components. Each public method has some goal, which exists in the subset of the used goal ontology shown above, annotated to it. It is assumed that the user has certain subscription constraints possibly included in the user profile – again we are considering the SPICE architecture where such user context information is maintained and handled [4]. A user with a PREMIUM subscription has unrestricted access to all these service components – accordingly an automatic service composition module may use whatever service component in the composition. Other users have restriction with regard to using ***RestaurantFinder*** component. This component in particular has a general method, *findRestaurant*, which achieves three goals based on what parameters it is provided. The other find restaurant components, ***RestaurantLocator***, ***RestaurantDB***, and ***RestaurantCatalog***, happen to have the same method name – findRestaurant – with different invocation parameters, achieve only one goal each. We assume there exists an ontology for inputs and outputs where certain parameters are classified. For instance *Cuisine* is some sort of *Type* and *Language* is some sort of *Property*, where Type and Property are also used as goals. ***NearstCalculator*** component has two methods: *getNearest1* that finds the nearest restaurant to the current user's location, while *getNearest2* only performs that given a certain location parameter. ***GSMLocator*** is straightforward finding the location of the user's mobile terminal.

The following listing shows few possible service compositions for this service example. These compositions may be obtained from an automatic service composition implementation[3]. In this listing only the invoked methods are shown with the corresponding service components (these methods are separated by "-->" to show the order of invocation), details such as the invocation parameters are not depicted.

1. *GSMLocator.getLocation --> RestaurantFinder.findRestaurant*
2. *GSMLocator.getLocation --> RestaurantDB.findRestaurant --> NearestCalculator.getNearest2*
3. *RestaurantDB.findRestaurant --> NearestCalculator.getNearest1*

The SPATEL representation of the first and the second compositions are shown in a diagram in Fig. 4. In the same figure a representation of the proposed goal diagram is illustrated also. It is important to notice that given this particular list of components/methods and the service request the ***RestaurantLocator*** and ***RestaurantCatalog*** components would not be usable in any service composition (there could be some work-around – e.g. performing a localized search on the restaurant list obtained from *RestaurantLocator.findRestaurant* – to be implemented within the composition logic itself, however this is not considered in our case). Certain composition alternatives are totally out of question. For instance the ***RestaurantCatalog*** component would return a restaurant that provides the Chinese language to its customers, and in extreme cases ***RestaurantLocator*** component would return a list of restaurants in China if the user is accessing SPICE platform from there.

[3] SPICE service platform is also considering an Automatic Composition Engine (ACE) module to be responsible for such automatic composition scenarios.

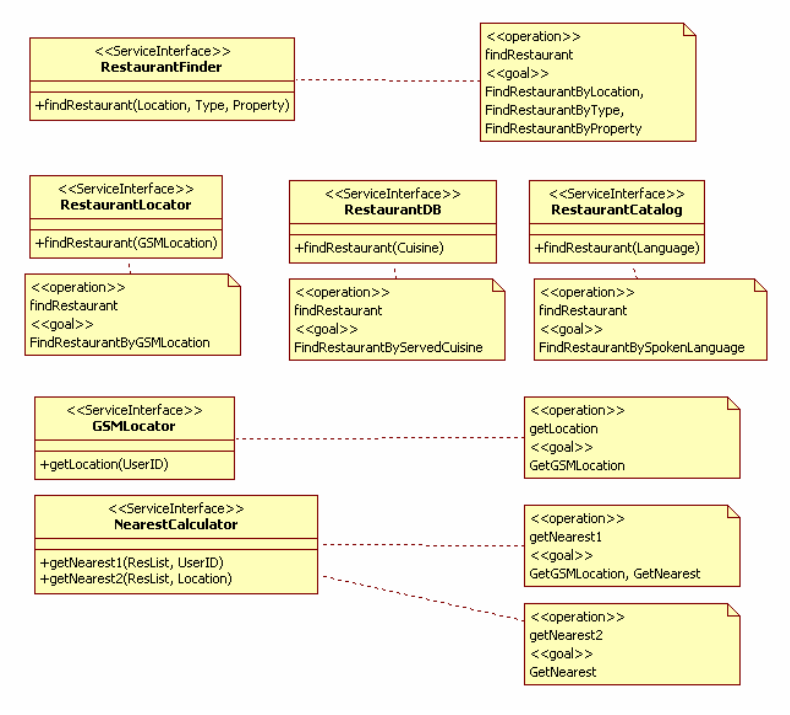

Fig. 3. Service components and their semantic annotations

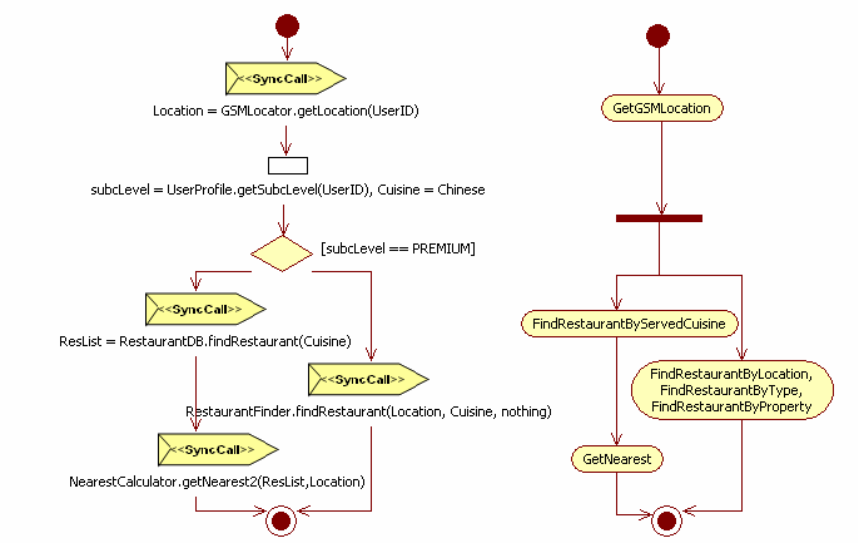

Fig. 4. An example SPATEL service composition, and its corresponding goal diagram

The validation runs for this simple service example have been performed to show that the resulting compositions: achieve certain goals (e.g. *FindRestaurantByType*), obey certain Boolean expressions on goals (e.g. *FindRestaurantByServedCuisine* AND *GetNearest*), and that it does not achieve certain goals (e.g. *FindRestaurant-BySpokenLanguage*).

6 Demonstration Using Eclipse Plug-in

A tool for demonstrating our approach has been developed as a validation wizard that interactively reason about the required service goals (as goal expressions). The tool is implemented as an Eclipse plug-in that asks the user for these required service goals and validates them. The plug-in checks the selected SPATEL model (which is saved as an xmi file – in SPATEL format) for goals in the composition and compares them to the required service goals. This plug-in has been integrated into the existing SPATEL Eclipse-EMF development environment as a menu item. A context menu called *Validate Goal* is one of the menus displayed when right clicking on a *.spatel file. There are three main use cases for the plug-in:

- Describe the expected service goals.
- Parse a model definition for service goals.
- Validate the found goals with the expected goals.

Currently the tool has the following limitations:

- Only Boolean expressions are accepted (e.g. GoalA = GoalB AND (GoalC OR GoalD))
- No handling of goal ordering or goal sequences
- No handling of multiple usage of goals (e.g. GetGSMLocation goal is achieved twice to find location of two users)
- No specification of loops (i.e. a goal could be reached several times – 1..* – until a specified state is reached)

7 Conclusion

The paper has demonstrated how it is possible to perform simple validation analysis or checks on service composition based on goal annotations. The presented approach is quite simple to understand and implement, yet easily extendable and make broaden to include other aspects. The goal expressions used so far have been fairly simple, but covered most of the validation aspects of the simple service examples we experimented with. These expressions need to be elaborated further. The goal diagram, or the annotation of goal sequences, as presented in this paper is just an activity diagram that has no strict connection, or semantics, with a SPATEL specification. It is important to consider a more coherent way of linking these two items – a goal sequence and a SPATEL specification diagram, e.g. using assignments of goal values and assertions in SPATEL specifications.

As a continuation of this work we are considering a more reachability-like analysis of the goal ontology. Such analysis would make the following objectives possible:

- Make sure that certain classes of goals never been considered. Such goals could be obtained by service components that require certain class of subscription, could use roaming, etc.
- Make sure that the given compositions are the only possibilities to achieve certain combinations of goals.

We are also considering extending our work to involve other SPATEL annotations, particularly *effects*, which has strong connection to service component states.

References

1. SPICE Consortium, the SPICE project website, `http://www.ist-spice.org/`
2. Floch, J.: Towards Plug-and.-Play Services: Design and Validation, Ph.D. thesis 2003:47 NTNU (2003)
3. Sanders, R.T.: Collaborations, Semantic Interfaces and Service Goals: a way forward for Service Engineering. Ph.D. thesis 2007:68 (NTNU (2007)
4. SPICE Consortium, Initial Architecture Design – SPICE Architecture, SPICE Deliverable D1.3, Goix, W. (ed.), SPICE Internal Document
5. SPICE Consortium, Advanced Language for Value added services composition and creation (SPATEL) SPICE Deliverable D5.1, Belaunde, M. (ed.), SPICE Internal Document
6. SPICE Consortium, Distributed Multiplatform Execution Engine, SPICE Deliverable D5.2, Kovacs, E. (ed.), SPICE Internal Document
7. OASIS, Reference Model for Service Oriented Architecture 1.0, `http://www.oasisopen.org/committees/tc_home.php?wg_abbrev=soa-rm`
8. Holzmann, G.J.: Design and Validation of Computer Protocols. Prentice Hall, Englewood Cliffs (1991)
9. Alpern, B., Schneider, F.B.: Defining liveness. Information Processing Letters 21, 181–185 (1985)
10. Pehrson, B.: Protocol Verification for OSI. Computer Networks and ISDN systems 18, 185–201 (1989/1990)
11. Bræk, R., Haugen, Ø.: Engineering Real Time Systems. Hemel Hempstead, Prentice Hall International 0-13-034448-6 (1993)