Service Creation for a Telecom Service Platform

Mariano Belaunde¹, Paolo Falcarin², João Paulo A. Almeida³

¹ Orange Labs, 8 Avenue Pierre Marzin, 22300 Lannion, France, mariano.belaunde@orange-ftgroup.com
² Politecnico di Torino, Corso Duca degli Abruzzi 24, 1-10129, Torino, Italy, paolo.falcarin@polito.it
³ Federal University of Espírito Santo, Av. Fernando Ferrari, s/n, Vitória, ES, 29060-970 Brazil, jpalmeida@ieee.org

Abstract

The current telecommunications scenario requires rapid development of services in the presence of both traditional telecommunications technologies and novel IT technologies. In this scenario, service creation is challenging and should therefore be supported by proper service design notations and service creation tools in a coherent service creation approach. This paper describes the SPICE approach to service creation. The approach consists of a language that leverages service reuse through platform-independent service composition; tools that support the definition of services and their deployment to a target service execution environment.

Keywords: Service Creation, Service Description Language, Telecommunication Services, Semantic Web

1. Introduction

The convergence of (telecommunication) networks, services and content is happening at an ever increasing speed. In this context, the creation of appealing value-added services is key to preventing an operator from being reduced to a “transport only” provider (Schulke, 2006). This new service market is foreseen to be an important battlefield for operators. The attractiveness of the service portfolio is paramount to attract and retain customers, thereby increasing revenues. At the same time, as the service market is highly competitive with a wider range of potential players, it is necessary to reduce the time-to-market for new services.

Shorter time-to-market requires the effective development of services. More concretely, the service lifecycle process needs to be optimized using a Service Creation Environment (SCE) that supports as much as possible the reuse and the composition of pre-existing consolidated components (Glitho, 2003; Falcarin, 2008). In fact, most of the business benefits triggered by these facilities stem from the possibility to reuse services, thereby enabling faster time-to-market and lower costs in the service development process. This leads to direct and indirect benefits to service end-users, service developers, platform operators and service providers.
One of the goals of the SPICE (Service Platform for Innovative Communication Environment) project is the design of an SCE to facilitate the development of services over heterogeneous platforms. The IST project SPICE is part of the Wireless World Initiative (WWI).

The approach to service creation in SPICE is driven by the constraint of being able to address heterogeneous target execution environments, where the technologies range from general Information Technology (IT), where, for example, Web Services are one of the leading technology in Service Oriented Architectures (Erl, 2007) to very specific telecommunication technologies (where a plethora of protocols and standards are available, most importantly SIP and IMS). What seems to be clear for telecommunication services is the need to integrate many resources over different protocols and to be able to represent a set of interactions that are not limited to the classic Request and Response paradigm. In such an heterogeneous environment, the approaches to service creation should be as general as possible, supporting a stepwise process that drives the developer from abstract to concrete definitions targeting a specific Service Execution Environment (SEE).

Considering the main challenges of the service creation process, this paper presents a language that allows the specification of telecommunication and IT services and tools that support the definition of services and their deployment to target service execution environments.

2. The SPATEL language

To describe SPICE services, a specialized description language named SPATEL has been designed (SPICE Advanced language for Telecommunication services). The approach used is based on the Model Driven Architecture approach (MDA, 2008), as defined by the Object Management Group (OMG, 2008). The purpose of SPATEL is to allow agile development of complex telecommunication services on top of the SPICE architecture leveraging state-of-the-art software engineering techniques such as model-driven engineering and software component technology. Such dedicated formalism supports the Service Oriented Architecture paradigm and includes specificities of the telecom domain like voice dialog support and multimedia data types.

The definition of a domain-specific language for integrated telecommunications services is one of the key elements for improving significantly the agility of the service development process. Various domain-specific languages addressing, for example the orchestration of web services exist nowadays (BPEL, 2003), however, high-level design languages coupling support of state-of-the-art IT technology and telecom specificities are difficult to find.

According to MDA principles, the SPATEL language is “platform independent” (Almeida, 2006), leaving the translation to specific execution engines, terminals and platforms to transformations executed by the SCE in a semi-automated way.
SPATEL specifies two kinds of service representation addressing two categories of service developers: a developer formalism and an end-user formalism.

The SPATEL formalism for developers is aimed at professional service developers that, expressing compositions as state machine diagrams, will be able to define complex composition patterns. The SPATEL formalism for end-users is aimed to the less-experienced service end-users or customers, giving them the possibility of creating new services by means of an assisted process which consists in the assembly of pre-existing components.

2.1 SPATEL formalism for service developers

In the service developer formalism, a service is primarily described through an external view which provides information that is useful for service clients. The external view is basically an interface declaring a list of operations, input and output events, multimedia streams and relevant side-effects. The constraints on the service interface such as the ordering of operation invocations can be precisely defined through a contract. Additionally, an important feature of SPATEL is the ability to annotate the elements of the interface (such as operations and the parameters) with semantics tags and non-functional features to enable rich scenarios for service discovery and dynamic composition.

The approach is similar to WSDL-S (WSDL-S, 2007) in that the annotations refer to elements defined in ontology files. The following semantic annotations are pre-defined:

- Annotations on Input and Output parameters of service operations.
- Annotations on goals that describe the overall objective of a service or the objective of a single operation exposed by the service.
- Annotations on the effects of a given operation that describe the outcomes of its execution in terms of the state achieved by the service or action performed.
- Annotations on the preconditions of a given operation that describe the conditions that must be satisfied in order to allow its execution.

Non-functional features are partitioned on the basis of categories like quality of service (QoS), charging, internationalization or resource usage. The service developer formalism also allows representing the internal view of a service (white box representation) by means of a set of interconnected service components. Two distinct views are available: an architectural view showing the list of involved components and their connections and a behavioral view consisting of state machines that define precisely the logic of an operation – an orchestration of components being a particular case. Figure 1 shows an example of a service composition defined using the developer formalism.
On the left side of the figure, the interface of a location aware hotel reservation service is firstly defined with semantics and non-functional tags that refer to pre-existing ontology files; on the right, the precise behavior of the “book” operation is defined as an orchestration of two sub-components: a location service (LOCS) and a hotel reservation service (HRS). Notice that the composite service depicted in Figure 1 could be reused in another service composition. Recursion in composition is based on the fact that any composite component exposes an interface and, as such, can be invoked as any elementary service component.

2.2 SPATEL formalism for end users

The end-user formalism is designed to enable users to assemble pre-existing service components to produce a new service addressing their needs. The formalism proposed is designed emphasizing ease-of-use and by strict controls on which components can be used and how they can be configured. A composite service in the end-user notation is defined as an orchestration of elementary building blocks, each of them defining a set of configuration parameters that can customize its behaviour. Moreover, building blocks offer a set of actions that can be triggered by event notifications produced by other building blocks or network resources.

The end-user formalism has been designed so that service compositions can be automatically transformed in the SPATEL developer notation. This allows us to reuse transformations for the SPATEL developer notation to realize service compositions defined with the end-user formalism. Figure 2 gives an example of a service composition defined using the end-user formalism. In this example, the composition of a location and a weather forecast service is expressed through dependencies, linking notifications (like onLocated) to actions (like findForecast).
The property configurations are used to prepare the context of a service invocation, whereas the variables allow retrieving the relevant outputs.

The end-user formalism is depicted with a user friendly and dedicated representation; for the developer formalism a UML2 profile has been provided in order to reuse existing notational conventions for depicting composition and behaviour. Both formalisms are further defined by a MOF conformant metamodel (MOF, 2007) from which the XML serialization is derived. The approach taken allows one to work with different concrete syntaxes and front-end tools with ultimately the same serialization and interchange format.

3. The Service Creation Environment

In SPICE, the Service Creation Environment (SCE) can be seen as a set of integrated tools that support the service creation process and is characterized by the following macro-components:

- Developer Studio: is used by professional developers for designing arbitrarily complex services by using the SPATEL/developer formalism for high-level design, in combination with general purpose languages for completing the non-generated parts of the code of the service. In particular the tool will be used to specify composite services orchestrating other components, which could pre-exist or be developed from scratch.

  Basic components typically consist of a SPATEL Service Interface and code implementing the exposed operations, while composite components consists of a Service Interface and a state machine representing the composition. The Developer Studio is provided with different pluggable transformers to support the translation of the SPATEL specification to code for a target execution platform, such as JSLEE (JSR-22, 2007) and BPEL at server side and J2ME at terminal side.

- End User Studio: is a graphical tool that, supporting the SPATEL end-user formalism, enables the end user to create new services by composing high level representations of Basic Components.
Addressing a wide population of non-professional service designers, this tool is as lightweight as possible, with minimal or no installation requirements and a friendly user interface. An approach based on an assisted graphical composition tool can be a core solution to be extended in a stepwise approach with further facilities like natural language interpreters (e.g. to propose initial automatic compositions to be manually visualized, verified and corrected), or with wizards to define the details of the composition.

- Automatic Service Composition Engine: produces service compositions automatically based on a formalized service request. Formalized service requests include a semantic description of the desired service or service composition, and, optionally, required non-functional properties or constraints. In SPICE, the composition generated is expressed using the SPATEL/developer formalism. One of the applications of automatic composition is the creation of composed services which consider the current situation or user’s context, for example to enhance or personalize the user experience (Shiaa, 2008).

- Deployment Tool: is used to package and deploy a SPICE component and a SPICE service composition in the target SEE. This tool has specialized sub-components for each of the different SEEs supported technologies. The SCE takes advantage of a standard user interface to interact with the SEE specific deployment facilities.

3.1 Supported Activities and Main Information Flows

Figure 3 depicts service creation schematically, both for professional service development and end-user service development. The Developer Studio supports the following main service creation activities for a professional service developer: (i) creation of basic components, which offer services that are described with SPATEL; (ii) creation of service compositions in SPATEL (these compositions involve basic components developed from scratch and previously defined components); (iii) discovery of components and their services for incorporation in service composition (offering an interface to the SPICE service repository); and, (iv) publication of service components and compositions. For the professional service developer, the service creation environment is said to provide design with reuse (through service discovery) and design for reuse (through service publication). The End-user Studio supports the creation of end-user service compositions with reuse of existing services. The existing services and the end-user composition are described with SPATEL for end-users.
Figure 3 Service creation with reuse and for reuse

Figure 3 describes a simplified service creation process, and omits activities such as service emulation, testing, and provisioning. Further, it does not reveal the aspect of semantic annotation of a SPATEL description with references to service ontologies; details of the SPATEL metamodel and the SPATEL semantic annotations can be found in (Belaunde, 2008). The relations between the various models supported by the Service Creation Environment and the components deployed in the Service Execution Environment are represented in Figure 4 where the deployment activities and information flows are identified, together with the transformation of SPATEL service compositions into Platform-Specific Models (PSM) and corresponding code. The figure also shows the transformation of a SPATEL end-user service composition into a SPATEL developer composition. This transformation is transparent to the user, and allows the reuse of transformations from the SPATEL service composition into the target technologies supported by the Service Execution Environment. The transformations are defined using Model-Driven Architecture (MDA) technologies and profit from the meta-model support for SPATEL.
Figure 4. Relationships among SCE and components deployed in the SEE

The main activities and information flows for Automatic Service Composition Engine (ACE) based on a semantic request formulated by service developer are represented in Figure 5. The Composition Factory Function is realized by the ACE component of the SCE. Domain and service ontologies are a key component in the ACE because they are related to the SPATEL descriptions that have to refer to concepts defined in the ontologies to annotate services with goals and to provide semantics for operation parameters. Moreover, the same ontologies are used by the Composition Factory Function to relate the semantics of a request and the semantics of composed constituent services.
It is also possible to envision an Automatic Service Composition process activated by natural language requests formulated by the end-user, eventually enriched by some additional contextual information (such as, e.g. the user’s location); the main activities and information flows for this scenario are depicted in Figure 6. Both the Semantic Analysis Function and the Composition Factory Function are realized by the ACE component in the SCE.

All the activities and information flows are integrated in the SPICE concept of the Service Creation Environment and are implemented by a set of tools, among which the most important ones are the Developer Studio and the End User Studio, whose characteristics will be briefly described in the following sections.
3.2 Developer Studio

The Developer Studio helps the professional service creator in creating both new basic service components and complex compositions of a set of published reusable service components.

Figure 7 shows the functions of the Developer Studio and its sub-components, using the Archimate notation (Archimate, 2008). Three main functions are identified to realize the “Service Development” service:

- Basic Service Development, employed in the development of components, which consist of a SPATEL Service Interface and code implementing the exposed operations.
- Service Composition Development, employed in the specification of composite services orchestrated components which could pre-exist or be developed from scratch; the result as captured in composite components consist of a Service Interface and a state machine representing the composition.
- Ontology Browsing, required to visualize existing ontology files, which can be linked to the service description to annotate service interfaces.

The Developer Studio is provided by extending the open-source project (StarUML, 2006) with different pluggable transformers that supports the translation of the SPATEL specification into the code for a target execution platform such as JSLEE, BPEL and J2ME.
Whenever a service (basic or composite) is ready to be released, the Service Packaging will pack related code and SPATEL descriptions and deploy them in the SEE by means of the Deployment tool.

Finally, the Automatic Composition Engine produces service compositions based on (formalized) service requests. Formalized service requests include a semantic description of the desired service or service composition, and optionally required non-functional properties or constraints. This component also aggregates properties of service compositions (which are produced manually or automatically), to derive the resulting composition’s properties.

The Automatic Composition Engine implements the following functions:
- Semantic Analysis, which produces a service request given a natural language request;
- Composition Factory, which produces a number of alternative SPATEL compositions that match a composition request. The composition request includes a semantic description of the required service (with references to URLs of public ontology files) and constraints for the composition.
- Property Aggregation, which aggregates the properties of the services that are composed into a composition. It takes as input the SPATEL service composition.

3.3 End User Studio

It is well known that telecommunication operators and service providers offer an always increasing number of services to their customers, but the impossibility to foresee all the conditions and needs
that a customer can experience in his daily life, makes it very difficult to provide an exhaustive set of services. For this reason, we have investigated the possibility of allowing end users to define their own services. The main challenges to enable service creation by the end user are related to the simplicity with which the user can create his own service and the security issues involved. The End User Studio provides strong support for telecommunication services, which often have the characteristic of being asynchronous and event based. These considerations lead to the conclusion that the service notation should support an event driven specification paradigm, i.e., service definition should consist of: (i) a number of building blocks that generate and process events, and (ii) a specification of the flow of events between these building blocks inside the service. From a graphical notation perspective, service control flow (ii) is expressed by means of arcs connecting components (i). The notation abstracts all protocol details; these are completely masked by the component implementation.

3.3.1 Support for Implicit Service Configuration

The End User Studio provides functions to address a set of security constraints, and to maintain control on the service that has been created. Not all the attributes, actions or notifications available for developers will be available for end users during service definition: some parameters will have to be implicitly derived from the user profile.

Two practical service examples could clarify the previous statement:

1) An end user wants to create a service that retrieves the weather forecast and sends the result in an instant message.
2) An end user wants to create a service that finds a restaurant with given characteristics and then performs a third party initiated voice call between the end user and the restaurant.

In both examples, we can identify security concerns: the destination of the instant message containing the weather forecast and the caller of the third party call must be the end user himself (to avoid unsolicited communication from ill-defined or malicious services).

The End User Studio will help to fill the configuration information of the components assembled in the service definition. One way to achieve this goal is to introduce an implicit “myself” building block, containing the user profile information. Going back to the examples, the “caller” of a ThirdPartyCall or “sender” of a SendIM can be forced to the attribute of Myself that represents a user’s SIP address.

Data accessible through the Myself building block are, e.g., name, nickname, e-mail address, SIP address, phone numbers, gender and preferences.

4. Related Work

Different service description languages have been specified both in telecom and IT domain. In the telecom world such languages have been often designed for domain-specific applications and
protocols (Licciardi, 2003), namely: Call Processing Language (CPL) (Rosenberg, 1999), Language for End System Services in Internet Telephony (LESS) (Wu, 2003), SCML - Service Creation Markup Language (Bakker, 2002), Call Control XML (CCXML, 2007), and SPL - Session Processing Language (Burgy, 2006).

The expressiveness of CPL and LESS has been intentionally limited to make them accessible to end-users without programming expertise, while SCML and CCXML require more technical knowledge, thus targeting expert users. Both SCML and CCXML are based on XML formats with no graphical notation.

SPL is a language which aims at raising the level of abstraction by introducing domain-specific constructs, such as sessions and branches. SPL hides SIP protocol complexity into appropriate language abstractions and makes programming telephony services accessible to more programmers. However, most of existing scripting languages for programming telephony services are limited, because they do not provide typical programming constructs such as loops and variables and they are tightened to a specific telecom network protocol, like SIP.

SPATEL has been defined as a high-level and executable language for describing composite telecommunication services. The language is a customization of UML for expressing the definition of service interfaces and service composition logic that is well-suited to the telecom domain. SPATEL supports critical workflow patterns, like arbitrary cycles and multi-merges, which are not supported by BPEL language and related engines (Wohed, 2003). Further, similarly to SAWSDL (SAWSDL, 2008), SPATEL supports semantic annotation of service descriptions, which is a basis to enable automatic service composition.

Finally, the SOAP Service Description Language (Parastatidis, 2006) has been proposed, enabling contract specification on WSDL 2.0; however, the language is better suited to precisely specify a single web service interface and it is not designed to describe service compositions and orchestrations.

5. Conclusions and Future Work

Service composition has become a hot topic for all telecommunication players. The ability for professionals and end users to compose telecom services, strongly depends on the availability of tools capable of hiding the complexity to access telecommunication network resources.

The SPICE project has contributed to this challenge by developing a Service Creation Environment, which supports the definition of services and their deployment to a target Service Execution Environment. The Service Creation Environment eases fast service creation both of basic service and of complex service compositions; the selection of reusable services can be based on actual QoS, cost, and other non-functional properties which are now made accessible both to end users and to a broad public of application developers.
The combined use of a Service Oriented Architecture (SOA) and Model-Driven Engineering is a distinguishing characteristic of the work carried out by this project; in fact, the SCE supports different pluggable transformers to automate the translation of the SPATEL specification into platform-specific code for different target execution platforms, such as JSLEE, J2EE and BPEL at server side and J2ME at terminal side.

We have defined a new language, named SPATEL, that is designed to create services that are easily portable to different platforms at server and terminal side; future work will focus on a possible alignment of our language with responses to the UPMS Request for Proposal (UPMS, 2009). In this respect, we tend to position SPATEL as a specialization of UPMS where only simple UML interfaces are used (no explicit notion of required interfaces) and where a composite service is represented by an opaque implementation or a state-machine.

On the platform side, future work will focus on targeting new telecom-oriented platforms like the Android environment from Google (Android, 2009) or FlexLite from Adobe Technologies (FlexLite, 2009): an interesting point for the future will be to see whether the heterogeneity in mobile terminals will continue to be a problem even with the emergence of a limited number of de facto Web 2.0 standards technologies in the mobile world.

ACKNOWLEDGMENTS

This work has been partially funded by the European Commission, under contract IST-027617, project SPICE (Service Platform for Innovative Communication Environment).

Paolo Falcarin has developed part of this work while affiliated to Politecnico di Torino University in Italy; he is currently Senior Lecturer at University of East London (UK).

João Paulo A. Almeida has developed part of this work while affiliated to the Telematic Instituut (now Novay) in the Netherlands. He is currently supported by CNPq with grant number 309059/2008-9.

References


