Service Creation in the SPICE Service Platform

João Paulo Almeida, Alberto Baravaglio, Mariano Belaunde, Paolo Falcarin, Ernö Kovacs

Abstract—Convergence is driving the uptake of new technologies on networking and service layers. The emerging new network infrastructure centered on 3GPP and IMS will have a horizontal layering with a good separation of control functions, media handling, networking issues, and service layers. This is a first step for fast service creations. In addition, we need a good tool chain for service creation as well as a rich execution environment. This paper describes the approach of the IST project SPICE with respect to Service Creation Environment and Service Execution Environment.

***Index Terms*—Service Creation, Service Execution, Service Description Language, Telecommunication**

I. Introduction

Convergence of networks, services and content is happening at an increasing speed. There are two major drivers: on one hand, technology development has increased competition in the telecommunications market and opened the opportunity for lowering operational expenses and concentrating capital expenditure in the core business. This is used by new market entries as well as by incumbent market leaders. The second aspect is that convergence is increasingly speeding up the introduction of new and/or converged services. New market opportunities like “triple play” of voice, video and data services are emerging from this trend.

The creation of appealing value added services seems to be a key feature to avoid an operator being reduced to a “transport only” provider [1]. This new service market is foreseen to be one important battlefield for the operators. The attractiveness of the service portfolio offered seems to be the key to attract and retain customers, and to increase revenues. At the same time, as the service market is highly competitive with a wider range of potential players, a key feature to compete effectively is to reduce the time-to-market for new services.

More concretely, the service lifecycle process needs to be sped up by an effective Service Creation Environment (SCE) that supports as much as possible the reuse and the composition of pre-existing consolidated components. 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 project SPICE (Service Platform for Innovative Communication Environment) is the design of a SCE for easy 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 [5], to very specific Telco ones, where a plethora of protocols and standards are available (SIP and IMS first of all). What seems to be clear in Telco 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/response paradigm. In such an heterogeneous environment, the approaches to service creation should be as general as possible, supporting a stepwise approach that drives the developer from abstract to concrete definitions targeting a specific Service Execution Environment (SEE).

Looking at the main challenges of the service creation process, the most important characteristics that will be described in the following consist of:

- A language that allows the specification of a Telco/IT integrated services
- Tools that support the definition of services and their deployment to a target Service Execution Environment
- A Service Execution Environment that allows to combine effectively different technologies.

II. The language

To describe SPICE services, a specialized description language named SPATEL has been designed [2]. The approach used is based on the Model Driven Architecture approach (MDA) [3], as defined by the Object Management Group (OMG) [4]. The purpose of a dedicated formalism is to allow agile development of complex telecommunication services on top of the SPICE architecture using state-of-the-art software engineering techniques like software component technology and model-driven engineering. 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 Telco 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 (e.g. BPEL [8]), 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", leaving the translation to specific execution engines, terminals and platforms to

transformations that will be offered by the SCE in a semi-automated way.

SPATEL specifies two kinds of service representation addressing two categories of service developers: the developer formalism and the 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. On the other hand, the SPATEL formalism for end-users is aimed to the less-experienced service end-users or customers, to give them the possibility of creating new services by means of a very much assisted assembly process of pre-existing components.

A. SPATEL formalism for service developers

In the service developer formalism, a service is primarily described through an external view which provides the information that is useful for the clients of the service. 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*. An important feature of SPATEL is the ability to annotate the elements of the interface (like the 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 [6] in that the annotations refer to elements defined in ontologies. The following semantic annotations are pre-defined:

- Annotations on Input and Output parameters of the 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 state achieved by the service or action performed.
- Annotations on the preconditions of a given operation that describe the conditions that have to be satisfied in order to allow its execution.

Non functional features are partitioned on the basis of categories like quality of service, 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 inter-connected 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.

The figure below gives an example of a service composition defined using the developer formalism. On the left the interface of a location aware hotel reservation service is firstly defined with semantics and non functional tags that refer to pre-existing ontologies; 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).

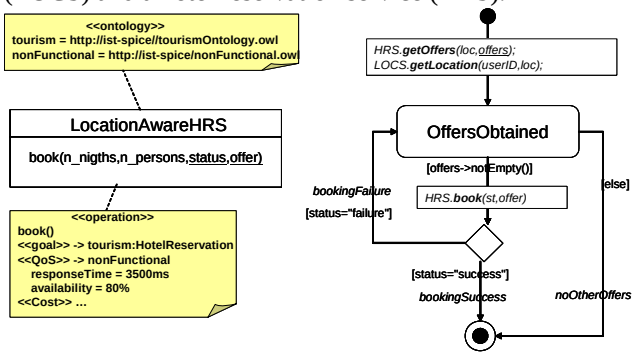

Fig. 1. SPATEL external and internal view of a composite service in the developer formalism

B 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 by simplicity constraints 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 behavior. 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. Figure 2 gives an example of a service composition defined using the end-user formalism. In the 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.

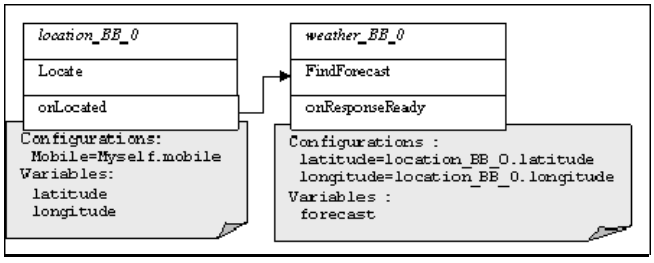

Fig. 2. SPATEL end user representation of a composite service

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 behavior. Both formalisms are further defined by a MOF conformant metamodel [7] from which the XML serialization is derived. The approach taken allows to potentially work with different concrete syntaxes and front-end tools but at the end having the same serialization format.

III. THE SERVICE CREATION ENVIRONMENT

In SPICE, the Service Creation Environment (SCE) is 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 will typically consist of a SPATEL Service Interface and code implementing the exposed operations, while composite components will consist of a Service Interface and a state machine representing the composition. The Developer Studio will be provided with different pluggable transformers that will support the translation of the SPATEL specification in the code for a target execution platform (such as JSLEE, J2EE, .NET, BPEL and so on).
- 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 should be as lightweight as possible, with minimal or no installation requirements and an extremely friendly user interface. An approach based on a very much 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 automatically service compositions 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.
- Deployment Tool: is used to package and deploy a SPICE component and a SPICE service composition in the target SEE. This tool will have to be specialized for the different SEEs and the SCE will take advantage of a standard interface to interact with the SEE specific deployment facilities.

A. 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.

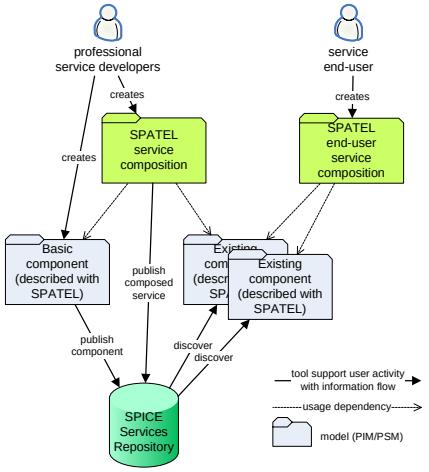

Fig. 3. Main components and actors in developer and end user service creation process.

Figure 3 describes a simplified service creation process, and omits activities such as service emulation, testing, provisioning, moreover it is not mentioned the SPATEL description enrichment by means of semantic descriptions referring to service ontologies; details of the SPATEL metamodel and the SPATEL semantic annotations can be found in [2].

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). Further it 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 profits from the meta-model support for SPATEL.

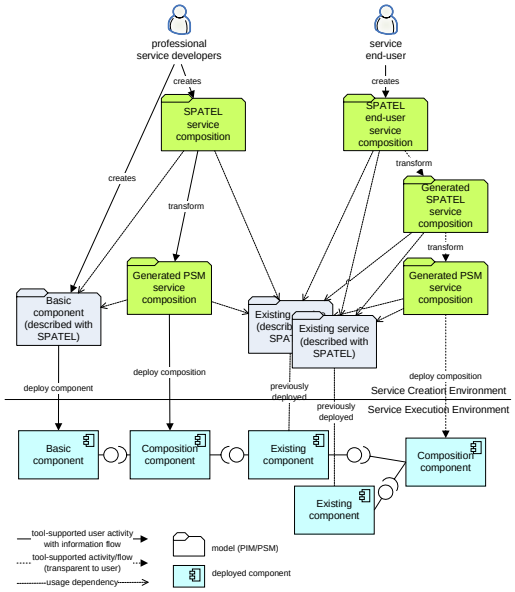

Fig. 4 Relations between the models supported by the Service Creation Environment and the components deployed in the Service Execution Environment

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.

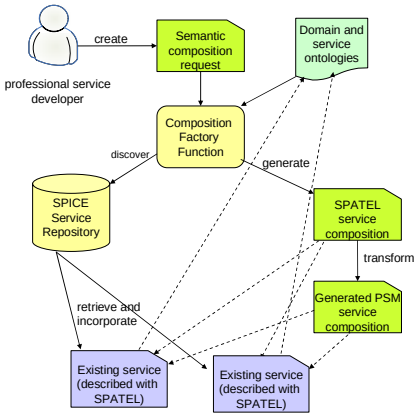

Fig. 5. Automatic Service Composition based on semantic request formulated by service developer

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.

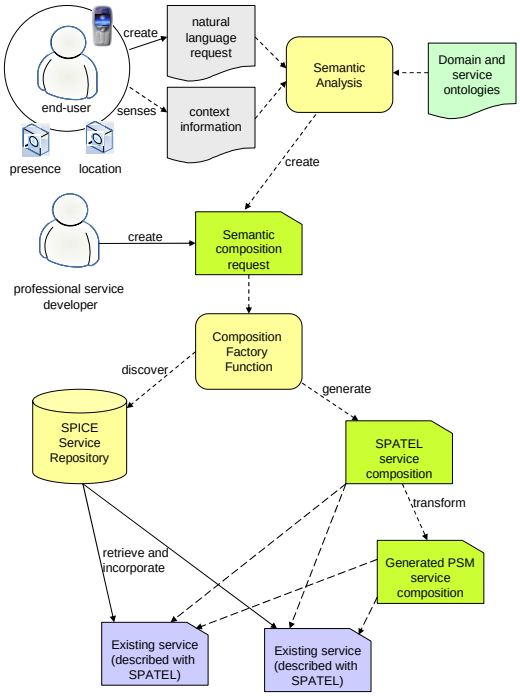

Fig. 6. Automatic Service Composition based on natural language request and contextual information

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

B. 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 architecture of Developer Studio and its sub-components:

- The Basic Service Development is used for developing basic components, which consist of a SPATEL Service Interface and code implementing the exposed operations.
- The Service Composition Development feature is used to specify composite services orchestrating components, which could pre-exist or be developed from scratch; composite components will consist of a Service Interface and a state machine representing the composition.
- The Ontology Browser is used to visualize existing ontologies, which can be linked to the service description if they contain semantic definitions of non-functional properties which are used to annotate service interfaces.

The Developer Studio will be provided with different pluggable transformers that will support the translation of the SPATEL specification into the code for a target execution platform such as JSLEE, J2EE, .NET; for

composite/orchestrated services, BPEL scripts could be produced.

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.

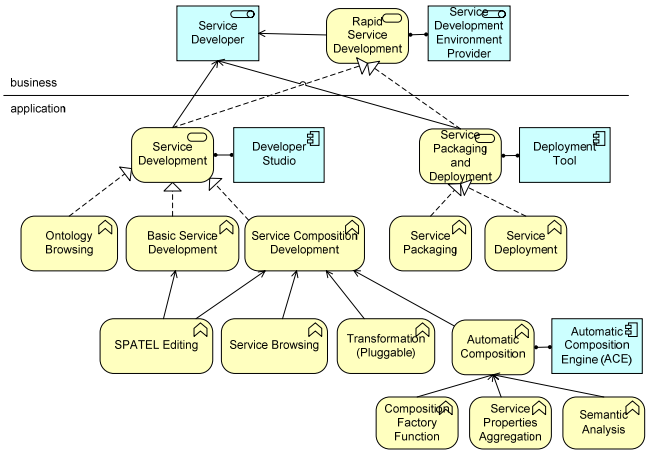

Fig. 7. The Developer Studio Architecture and the dependencies relations between its functions

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 is composed by the following sub-components:

- The Semantic Analyzer produces a service request given a natural language request;
- The Composition Factory 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 ontologies) and constraints for the composition.
- The Property Aggregator aggregates the properties of the services that are composed into a composition. It takes as input the SPATEL service composition.

C. End-User Studio

It is well known that Telco 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 very difficult to provide an exhaustive set of services. That's why it seems to be interesting investigating the possibility to allow end users to define their own services as long as they need them. This goal is very challenging and poses a set of constraints mainly related to the simplicity of the way the end user can create his own service and the security issues related.

In the End-User Studio, the service creator will be the end user himself. Users will be able to satisfy specific needs, creating new services by means of a very much assisted assembly process: the services will be the result of a composition of a set of pre-existing components.

An assembly process targeting end users, can be supported by different tools with different degrees of friendliness. These tools can range from natural language interpreters to wizard based rules editors to graphical component assembly tools. An approach based on a very much assisted graphical composition tool can mediate between implementation complexity and effectiveness of the solution; moreover it can be a core solution to be extended in a stepwise approach with natural language interpreters (e.g. to propose initial automatic compositions to be manually visualized, verified and corrected), or with wizards to define some details of the composition.

The End User Studio should provide a strong support for Telco services, that often have the characteristic of being asynchronous and event based (e.g. whenever I get an incoming message…); these considerations lead to the conclusion that the service notation should support an event driven specification paradigm, i.e. building blocks generate and process events and the service definition specify the flow of events inside the service. From a graphical notation perspective, the service control flow is expressed by means of arcs connecting components. The notation abstracts all protocol details; these are completely masked by the component implementation.

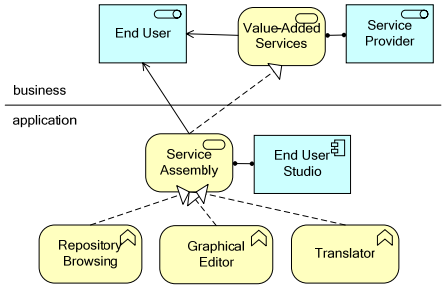

Fig. 8. End User Studio core components and their relations

As described in the former figure, the End User Studio will be provided with Translators that will allow the service definitions to be transformed in the SPATEL developer oriented notation. This will allow reusing the common functions of service lifecycle management previously described, concentrating the End User Studio to the service assembly functions

1) Support for Service implicit configurations

The End User Studio will provide functions to address a set of security constraints, 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, in the 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 it is quite evident that there is a security concern: the destination of the instant message containing the weather forecast and the caller of the third party call must be forced to the end user himself (to avoid non proper 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 could be, for example: name, nickname, e-mail address, sip address, phone numbers, gender, preferences...

2) The End User Studio UI

The user interface of the End User Studio has to be very friendly: service composition must be intuitive and self-explaining and no software development skills must be required to develop a service.

Here is a list of requirements that should be fulfilled by the End User Studio

- Short learning curve: it should be possible to learn the tool hands-on, just by going through few examples.
- Service composition assistant: the composition could be assisted in many ways; for example whenever the user picks a block, the End User Studio can suggest a selection of other blocks that can be used or, when making a connection, it can suggest a list of meaningful endpoints. This requirement can be met in two stages: in a first phase this can have a simple syntactical implementation (i.e. the user is presented with a set of building block instances, among those that are already in the service definition, that will yield a syntactically correct representation), while in a second phase, the selected set of building blocks could be further restricted to those that will also produce a semantically correct overall service definition.
- No installation required: the End User Studio should be accessible through lightweight interfaces like web applications.

Some of the upcoming technologies in Web 2.0, like AJAX [9], seem to be promising for the implementation of a service editor of the End User Studio

IV. The Service Execution Environment

The Service Creation in SPICE, as previously explained, is based on a service specification that is Platform Independent, leaving the transformation to a target Execution Engine to a following phase; coherently to this approach, the SPICE SEE is designed to allow the integration of different containers, specialized to address specific functions. Even if the specific containers integrated in the overall SEE are not very much affecting the service creation process (due to the separation in platform independent and specific models), it is relevant to describe how the overall SPICE architecture is designed to facilitate the definition of reusable components available on SEE for the SCE.

The following figure shows the layered structure of the SPICE architecture.

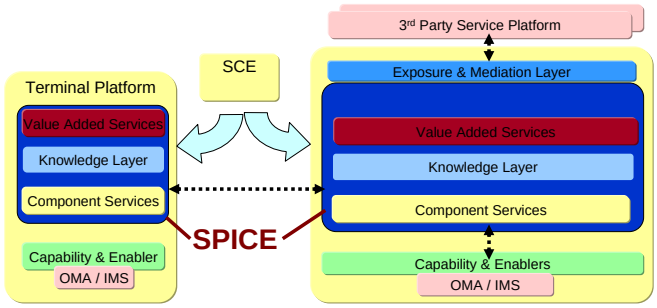

Fig. 9. Layers in the SPICE Architecture

The SPICE SEE offers a component model that allows to install SPICE components in a variety of service execution engines (such as J2EE and JSLEE) with a moderated effort to adapt to the specific runtime requirements of these execution engines. Based on their offered interfaces, the components are logically divided into the basic “Component Services”, the “Knowledge Layer” and the “Value-Added Services (VAS)” layer. The SCE usually deploys its service compositions in the VAS layer.

Basic Components offer access to the underlying capabilities and enablers of the network. For example, there are basic components for access to IMS (e.g. by intercepting the SIP messages exchanged), to OMA enablers (e.g. by utilizing existing interfaces to the respective Application Servers) such as presence, IM or Third-Party Call Control. The interfaces to these enablers are usually wrapped into SPICE components thus allowing to fast use the SPICE mechanisms for access control, policy enforcement, semantic publishing and discovery, as well as making components aware to knowledge and context.

V. Conclusions

In this work we have described the approach of the SPICE project with respect to Service Creation. The approach consists of a language that leverages reuse through platform-independent service composition; tools that support the definition of services and their deployment to a target Service Execution Environment and a Service Execution Environment that allows combining effectively different technologies.

The SCE architecture 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, .NET, as well as BPEL scripts for orchestrating service compositions in the SEE.

The Service Creation Environment eases fast service creation both of basic service and of complex service

compositions; selection of reusable services can be now based on actual QoS, cost, and other non-functional properties which are now made accessible, thanks to the SPICE SCE, both to end users and to a broad public of application developers.

The SPICE SCE, together with the SPATEL language and the SPICE SEE, will broaden up the developers community, thanks to a new way of reusing components to provide advanced telecommunication services. This will ease the service creation process diminishing time-to-market for the new services.

REFERENCES

[1] Anett Schülke, Daniele Abbadessa, Florian Winkler. Service Delivery Platform: Critical Enabler to Service Providers' New Revenue Streams. World Telecommunications Congress 2006.

[2] M. Belaunde et al, “Advanced Language for Value added services composition and creation”, IST Spice Project Deliverable D5.1, August 2006.

[3] OMG, Model Driven Architecture, http://www.omg.org/mda/ (October 2006)

[4] OMG, The Object Management Group (OMG), http://www.omg.org/ (October 2006)

[5] Thomas Erl, "SOA : Service-Oriented Architecture (SOA): Concepts, Technology, and Design", Prentice Hall, 2005

[6] Rama Akkiraju et al, “Web Service Semantics - WSDL-S”, http://www.w3.org/Submission/WSDL-S (October 2006)

[7] OMG, Meta Object Facility (MOF) Core Specification, http://www.omg.org/docs/html/06-01-01/Output/06-01-01.htm (October 2006)

[8] OASIS, “Business Process Execution Language for Web Services (Version 1.1)”, OASIS, May 2003

[9] Jesse James Garrett, “Ajax: A New Approach to Web Applications”, Adaptive Path, http://www.adaptivepath.com/publications/essays/archives/000385.php (October 2006).