Author(s): Belaunde, Mariano; Falcarin, Paulo; Jezequel, Jean-Marc.
Article title: A Model-driven Framework for Professional Service Designers and Developers
Year of publication: 2008
Citation: Belaunde, M; Falcarin, P; Jezequel, J.M. (2008) A Model-driven Framework for Professional Service Designers and Developers, In: ICIN (International Conference on Intelligent Networks): Services, Enablers and Architectures, Adera (FRA), ICIN 2008, Bordeaux, Oct 2008.

A Model-driven Framework for Professional Service Designers and Developers

Mariano Belaunde
Orange Labs,
8 Avenue Pierre Marzin,
F-22300 Lannion, France
mariano.belaunde@orange-ftgroup.com

Paolo Falcarin
Politecnico di Torino, Dipartimento di Automatica e Informatica (DAUIN),
Corso Duca degli Abruzzi 24,
I-10129, Torino (Italy)
paolo.falcarin@polito.it

Jean-Marc Jezequel
Inria/Irisa,
Campus de Beaulieu,
F-35042 Rennes Cedex
jean-marc.ezequel@irisa.fr

Abstract

A Service Creation Environment (SCE) aims to speed-up the process of designing, developing, deploying, testing and maintaining telecommunication services. This paper describes the solution build in the context of the IST SPICE collaborative project. The developed SCE exploits generative techniques to produce implementations from abstract service definitions as well as a flexible execution engine capable to manage variability in component usage and selection.

1. Introduction

Agility in service creation is becoming a crucial mean for a telecom company to differentiate from other telecom competitors as well as to compete against aggressive new actors coming from the IT and internet industry. Service creation agility is also necessary to allow third party service providers to develop their business proactively within the fast-evolving world of today.

A key factor for improving time-to-market time to develop new services is the ability to reuse service components in design and in runtime. Un-surprisingly, the SOA paradigm [2] (Service Oriented Architecture) has gained a major interest within telecom manufacturers, operators and third party players: potentially, it simplifies dramatically the integration of complex communication components like SMS sending and GSM localization within a service logic that combines its use with ordinary Internet/IT components - Personal Agenda, Interest Points and so on. Various telecommunication operators, like Orange, have started to open the access to its network resources through the exposition of their telecommunication enablers in the form of SOAP web services [3]. The expectation is that third party providers will develop a plethora of new services that will generate revenue due to the usage of operator network resources.

Another aspect of agility in service creation is the ability to realize services that can run on top of different execution technologies and that have client code that is compatible with a wide range or mobile terminals. Furthermore, the execution of a service may consist of various components executing in parallel in different locations, each using a different execution technology, and at the end being synchronized. Support of heterogeneity may also be motivated by the rapid evolution of technology which demands frequent changes in the implementation of services.

To achieve this purpose, the principle of separation of concerns between specification and implementation becomes crucial and here is where model driven technologies come to play. The SPICE project has defined a service description language, named SPATEL (SPice Advanced TELecommunication language) which allows to capture service definitions and compositions in a technology "agnostic" way. The SCE Developer Studio implements this language and provides automatic mappings to various technologies using modern transformation technology [1] based on meta-modeling.

2. The SPATEL language

The SPATEL language first of all provides means to describe a service as a *black-box*, that is, merely as an interface exposing: the list of service operations, the list of accepted or emitted asynchronous events and, if relevant, a list of noticeable side-effects (like "an SMS in sent"). In addition each service operation can be annotated either with "semantic" information (like "what's the goal of the service", or the meaning of an

input parameter) as well as annotated with non-functional features (security, cost, response timeouts and so on). These annotations refer to concepts defined in external ontologies, and hence its exploitation relies on the overall consistency of the used ontologies and their acceptance by the different involved actors. The semantic information is intended for service discovery and automatic service composition. Some of the non functional properties are exploited when generating the code implementing the service - mainly for handling security; others are intended for intelligent automatic composition. We should point out however, that a large-scale exploitation of these semantic and non functional property annotations is nowadays restricted by the lack of standardization: there is, for instance, no widely shared way to represent the semantics of the concepts used to describe a translation service.

The SPATEL language also allows describing the service as a *white-box*, that is, exposing the specification of the logic of a service operation as potentially a composition of other services. In contrast with more "traditional" request/response services, a service operation in our context may be long-running and have its execution being stopped waiting for the arrival of asynchronous event occurrences. The selected formalism to represent such kind of behavior are state-machines since they allow to describe quite appropriately multi-modal behavior mixing voice-based interaction with GUI based interaction. Specific constructs are hence included in SPATEL to support voice dialogs.

Last but not least, the SPATEL language allows specifying GUIs for the interface client part. This is done using a generic approach to avoid inventing yet a new widget system: rather than defining concepts like ComboBox and TextInput, the language defines the ability to describe a hierarchy of GuiElements with GuiProperties. This part of the specification is typically used to generate specialized J2ME [4] clients.

3. The SPATEL Engine

Having a language sufficiently abstract and expressive for the telecom domain is not sufficient for realizing an effective framework. Most of the intelligence is to be placed in the model transformers, the code generators and in the target execution engines. A specific transformer generates code for a BPEL [5] engine. This transformer works well for simple orchestrations but is indeed not usable for more complex service logic involving voice interactions. A native execution engine for the SPATEL language has hence been developed under the name of SPATEL Engine. This engine, written in Python language [13], benefit from easy-writing and rapid development features characterizing Python applications.

The SPATEL Engine can be seen as a framework that implements *natively* the SPATEL language:

- To the concept of Service Interface in SPATEL corresponds a Python class, sharing exactly the same structure: same visible operations and same exposed attributes. This class acts either as a client proxy for the service - a glue component connected to a potentially remote implementation - or as the "real" implementation, to be defined locally.

- To the concept of State Machine in SPATEL corresponds a State Machine implemented in Python. Similarly to some VoiceXML [6] systems, the state machine is loaded into memory once at the activation of the service. Then each session object - representing the usage of the service by a user - has a pointer to store its position in the execution of the state machine.

The SPATEL engine relies on an HTTP server to offer multi-threaded and asynchronous support. A session mechanism is explicitly maintained by the framework to allow keeping alive the context when dealing with long-running services.

Two forms of execution are supported: one uses CGI protocol [7], the other uses *servlets* on top of TOMCAT [8]. In the first case, the HTTP server invokes Python CGI which rebuilds the saved context at each invocation. In the second case a Jython interpreter [9] is used to connect Java [12] and Python [13].

4. The Clubber Service Use Case

In this section we describe the Clubber service scenario which has been developed in the IST SPICE project to demonstrate the capabilities of the Service Creation Environment. A mobile social networking service for dancing clubs allows clients of clubs to send free-call invitations to friends based on pseudonyms and to receive bonus fees in case of acceptance. This service implies collaboration between components provided by the operator (pseudonym-based access to telecommunication enablers) and components provided by a third party service provider (which develops the orchestration and access a pre-

existing Club database). The free-call scenario is depicted by the Figure 1 below:

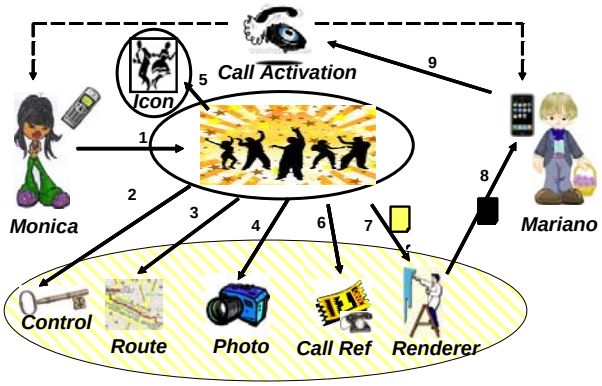

Figure 1: Free Call Clubber Scenario

Firstly, Monica, which is within a dancing club, opens a dedicated widget application in its mobile phone and (1) sends a free call request to one of its friends (Mariano) to invite him to join the club. The composite service logic will build the content of the invitation as follow: (2) checking the pseudonyms; (3) retrieving the coordinates of the invited user to prepare a route map request to access the club; (4) retrieving the Monica photo, (5) retrieving the Club icon, (6) booking the click-to-call reference for the free call, and finally (7) sending the invitation template to the operator so that he can (8) push to the destination user. In fact, there are privacy concerns in this scenario: the 3rd party is not allowed to have means to know who invited who and to look at localization (map) and identification data (Monica photo). For that reason, as indicated in Figure 1, most of the intermediate results are returned encrypted. They are decrypted by the operator prior the final delivering of the invitation page. The last step of this scenario is the (9) activation of the click-to-call link by the target user which provokes the call.

To implement this service, the operator firstly provides to the 3rd party service provider the SPATEL service interfaces of the invoked components that are under is responsibility. Then the 3rd party provider is able to define its composition logic using the specific UML [10] graphical form for SPATEL. The Figure below shows an excerpt of this model:

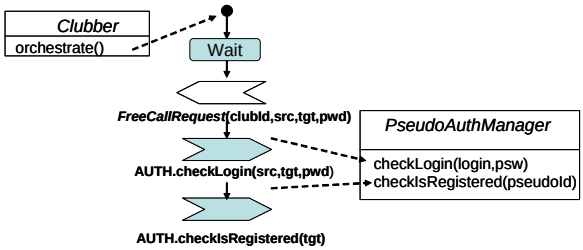

Figure 2: Graphical modelling

In Figure 2 the "orchestrate" operation is specified by means of a state machine in which we see that a FreeCallRequest asynchronous event is expected when the Wait state is reached. Specific nodes represent the invocation of service component like the PseudoAuthManager component to check pseudonyms. This invoked interface is opaque is respect to the 3rd party provider since only the signature is exposed to him.

5. Model-driven process

We describe here briefly the model-driven process to run a composite SPATEL service like the one defined in the Clubber use case. Initially, (1) the service designer defines and/or imports service interfaces and state machines. (2) Service interfaces and other entities are translated as Python class skeletons, and SPATEL state machines are translated into the equivalent state-machine in Python. Opaque operations which are not connected to existing remote services require manual completion. The following steps are (3) the testing of the service using a fully generated web-based interface, (4) the generation of various widget applications running on different mobile phones. Finally, (5) if reuse of the composite service is relevant, the new service can be promoted as a new service available as a SOAP web service.

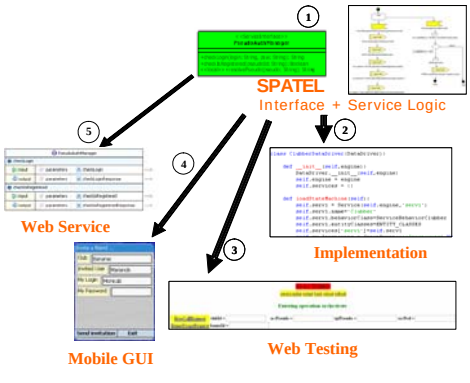

Figure 3: Service Creation Generative Process

Apart from an important increase in productivity, the two interesting capabilities offered by this model-driven process are to enable two kinds of variability: a *vertical flexibility* which allows service logic to run in potentially various execution platforms. In the case of our scenario, the logic runs in an instance of the SPATEL engine as well as on top of a BPEL engine. A second kind of variability, that we call *horizontal flexibility* allows replacing an invoked component by another, by simply adapting the implementation on the basis of a neutral common interface. In our scenario example, the click-to-call component used to provide the phone call facility has two alternative implementations that can be called by the composite logic: an implementation based on Asterisk platform [11] or a dedicated enabler offered by Orange. To facilitate horizontal flexibility, when dealing with SPATEL Engine, the code generator produces in fact various variants for each service operation: a fake version - which does nothing but is useful for earlier simulation, a local version - which is supposed to contain the local default implementation, and finally a "remote" version, which merely links to an existing SOAP or HTTP GET/POST service. To activate the last variant, the service designer needs to populate a web service registry and indicate, through a declarative deployment descriptor, how parameters are mapped to those expected by the connected web service.

Another noticeable capability that is obtained thanks to code generation is the ability to insert automatically code to handle security constraints on the invocation of operations: an operation may be visible and yet not be available to unregistered users.

6. Conclusion

In this paper we have presented a framework for agile development of telecommunication services which make use of model driven technologies. This framework is firstly based on the definition of a domain specific language, named SPATEL, which uses UML interfaces and UML state machines for describing services in a technology agnostic way. The framework exploits the specific capabilities of the SPATEL Engine to allow testing, simulating and/or performing a real execution of the modeled composite service.

Unsurprisingly, we found out that is the tightly combination of a domain-specific language with a dedicated *native execution technology* allows to obtain more effective results in terms of service creation agility. The link between the two is done thanks to generative techniques. However, maintaining a clean conceptual separation between the design language and the execution technology remains of a major importance since, in case of replacement of a support technology, it potentially allows preserving the investment done to design new services. Also the fact that telecommunications services are essentially distributed and involve potentially different kinds of terminals is another reason for maintaining the distinction between design and implementation, and between executable models and executable code.

ACKNOWLEDGMENTS

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

7. References

[1] [MDA]. OMG, "Model Driven Architecture", document ormsc/2000-11-05, Nov 2000.
Web link: http://www.omg.org/mda/

[2] [SOA]. OASIS, "OASIS Reference Model for Service Oriented Architecture V 1.0", Aug 2, 2006. Web link: http://www.oasis-open.org/committees/download.php/19679/soa-rm-cs.pdf.

[3] [SOAP] Simple Object Access Protocol, version 1.2.
Web link: http://www.w3.org/TR/soap12-part0/

[4] [J2ME] Sun, Java 2 Micro Edition, Connected Limited Device Configuration 1.0, JSR 30.
Web link: java.sun.com/javame/index.jsp

[5] [BPEL] OASIS, Web Services Business Process Execution Language Version 2.0 (BPEL), 2007 11 April,
Web link: www.oasis-open.org/committees/wsbpel/

[6] [VoiceXML], W3C/VoiceXML Forum: Voice Extensible Markup Language, doc:http://www.w3c.org/TR/2007/REC-voicexml21-20070619/

[7] [CGI], Common Gateway Interface, W3C standard
Web link: http://www.w3.org/CGI/

[8] [TOMCAT], Apache Software Foundation, Tomcat project. Web link: http://tomcat.apache.org/

[9] [JYTHON], Python interpreter in Java
Web link: http://www.jython.org

[10] [UML]. OMG, "Unified Modeling Language V 2.1.2", document: formal/2007-11-04, Nov 2007.
Web link: http://www.omg.org/spec/UML/2.1.2/

[11] [ASTERISK] Open source PBX & telephony platform
Web link: http://www.asterisk.org/

[12] [JAVA] Java language, at http://java.sun.com

[13] [PYTHON] Python language, at http://www.python.org