Author(s): Belaunde, Mariano; Falcarin, Paolo.
Article title: Realizing an MDA and SOA Marriage for the Development of Mobile Services
Year of publication: 2008
Citation: Belaunde, Mariano and Falcarin, P. (2008) 'Realizing an MDA and SOA Marriage for the Development of Mobile Services', In: Schieferdecker, I. and Hartman, A. (Eds.) *Model Driven Architecture - Foundations and Applications* ECMDA-FA 2008, LNCS 5095, pp. 393–405
Link to published version: http://dx.doi.org/10.1007/978-3-540-69100-6_28
DOI: 10.1007/978-3-540-69100-6_28

Realizing an MDA and SOA Marriage for the Development of Mobile Services

Mariano Belaunde[1] and Paolo Falcarin[2]

[1] Orange Labs, 8 Avenue Pierre Marzin,
22300 Lannion, France
mariano.belaunde@orange-ftgroup.com
[2] Politecnico di Torino, Dipartimento di Automatica e Informatica (DAUIN),
Corso Duca degli,
Abruzzi 24, I-10129, Torino (Italy)
paolo.falcarin@polito.it

Abstract. The paper presents an approach for developing composite telecommunication services running on mobile phones which takes advantage of the use of model driven techniques as well as the loose coupling paradigm in SOA. A domain-specific UML dialect named SPATEL has been developed which serves as the basis for generating applications that can be deployed in distinct terminals and servers technologies. The composite services typically combines telecommunication enablers - like SMS sending and GSM localisation - with traditional IT components accessible over the internet, such as a Yellow Page facility. This work has been conducted in the context of the IST SPICE European collaborative project.

Keywords: MDA, SOA.

1 Introduction

The emergence of SOA[1] and MDA[2] as engineering approaches to build software and systems in the IT world will have a significant impact in the way telecom infrastructures are built and telecommunication services are created. In this paper we describe our experience on combining the strengths of both paradigms to facilitate agile and portable development of telecommunication services running over different server execution technologies and mobile terminals.

1.1 Meaning of SOA in Our Context

The SOA acronym, meaning Service Oriented Architecture, is an over-used term used to describe very different situations and products. Generally speaking it refers to the mechanisms that provide service functionality remotely in a loosely coupled and distributed way over web resources.

Now, what is the impact of SOA in telecom? Telecom operators have realized the importance for the growth of their market to open, in a controlled way, the APIs to access to their telecom network resources, in a similar way as important IT players do. More specifically, various companies like Orange are now offering to third party

service providers the possibility to access elementary functionality, like SMS sending, call control and localization (called telecommunication enablers) through simple SOAP web services.

Hence, from the point of view of services developers that have to build complex composites services that make use of these telecom enablers, effective adoption of SOA principles is of major importance since it represents the ability to program telecom services without having to be necessarily experts in the telecom domain. This is a significant point, since it may imply important cost reduction in development.

1.2 Meaning of MDA in Our Context

The MDA acronym has been defined by the OMG standardization body in 2000. It means Model Driven Architecture but may represent very disparate things. In the context of service creation a *model-driven* approach consists primarily in the ability to generate large amounts of a service implementation from a high-level abstraction definition of the service, exploiting object-oriented modelling techniques – like MOF [3], UML[4] and QVT[5] standards. Apart from this technical aspect – which is mainly intended to improve productivity – another aspect of MDA relevance in telecom domain is the issue on heterogeneity of terminals and execution platforms. A telecom service for mobiles ideally would require to be developed once and yet be able to run, at the client side, in different kinds of mobile terminals and, at server side, be prepared to evolve from a technology to another (like evolving to the IMS architecture [6]).

1.3 Organization of the Paper

The following chapters will describe with some detail the SPATEL service description language, the service creation tool associated with this language and an application use case illustrating the usage of our model-driven framework in a typical *context-aware* telecom service combining IT and TELCO components.

Finally we will discuss some of the interesting issues raised by our experiments.

2 The SPATEL Service Description Language

The SPICE project has defined a *high-level* and *executable* language for describing composite telecommunication services. This formalism, named SPATEL, meaning SPICE Advanced language for Telecommunication services, can essentially, be seen as a customization of the UML language for expressing the definition of service interfaces and service composition logic that is well-suited to the telecom domain.

In contrast with most IT web services, telecom services are generally transactional, asynchronous, state-full and sometimes long-running processes. In addition a telecom service can be designed to be multi-modal – the ability to achieve a conversation using parallel interaction means like voice, text and image. Also the behavior of a telecom service needs to be often split in two parts: one running in the mobile terminal of the user – dealing with GUI aspects and local activation of telecom resources (like SMS sending) - and the other part running at server side, usually hosted by a telecom operator.

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. 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. Non-functional features are partitioned on the basis of categories like quality of service (QoS), charging, internationalization or resource usage. The annotation mechanism, which is similar to the approach taken in SA-WSDL approach – as it relies on pointers to pre-existing ontologies – is not detailed in this paper – which is focused on static composition.

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. We will see some examples of the usage of this formalism in Section 4.2. The choice of state machines – rather than activity diagrams – is motivated by the idea of integrating "voice-based" dialogs in a service specification, since state machines are the most used paradigm for expressing the complexity that can be found in human-machine voice conversations. We should note however that the scope of SPATEL is much broader than the scope of traditional voice services since we have to deal with remote synchronous and asynchronous invocations, parallel threads of execution and dedicated GUIs definition at terminal level.

Concerning the definition of user interaction at client side, SPATEL provides the ability to represent potentially the usage of different GUI frameworks found in mobile world like, the very constrained J2ME [7] GUI environment or the richer GUI framework available in S60 Nokia [8] *smartphones*. This heterogeneity is enabled in the SPATEL meta-model by the fact that the coding of GUIs elements is generic: a Container contains recursively GUI Elements which in turns define GUI properties – which are name/value pairs. In addition to that GUI events can be connected to service events used within the logic of the service. Hence, thanks to this very pragmatic approach – not trying to model the whole IT world! – in our context, supporting a GUI framework means having the corresponding library of the GUI widget model components instantiated in the SPATEL design tool and having the corresponding code generator targeting the specific GUI framework.

In addition to this GUI aspect, SPATEL provides means to represent typical voice-based interactions: recognition of voice as *utterance* events, buffered construction of voice messages – which are delivered when reaching a stable state and support of specific events – like inactivity or failure recognition. The SPATEL language and execution environment inherits from previous research work done in the field of voice service modeling [9]. Hence, this aspect will not be detailed here. However it is important to point out that the combination of voice interaction modeling with GUI modeling brings the ability to model multi-modal services.

As we can notice, despite the specificity in telecom services, from a design point of view, the concepts needed by the service description language are not significantly different from those exposed by the well-know SOA standards like WSDL [10] and BPEL [11] and formalized in a more abstract way by UML [4]. The SPATEL formalism aggregates in fact well-know constructs coming from different sources (ITU-SDL [12], SA-WSDL [13], VoiceXML[14]) in order to provide the needed subset – not less, not more – that is needed for a high-level and executable formalism usable in telecom context. Among the potentially infinite design choices that UML can offer, SPATEL makes a very precise and exclusive selection like: using simple UML 1.4 state-machines instead of the full UML2 capability, not using collaborations, representing an orchestration as the behavior specification of an operation. Selectivity in the usage of the constructs offered by UML for behavior definition is necessary to have at the end an unambiguous and executable formalism that can be implemented at reasonable cost. Notice that we are not claiming that the choice we made is the only possible one. In our case the state-machine formalism had the advantage not only to be well accepted within the community of service designers but also to have well-established and robust implementations that could be used as entry points for our developments.

Technically speaking the SPATEL formalism has been defined by an EMOF meta-model and is accompanied with a UML2 profile defining the conventions for using the UML graphical notation – like adding an specific icon to represent the invocation of remote service operations. This approach, which makes the distinction between abstract syntax and concrete notation, allows using a rather compact and understandable XMI serialization format as exchange and pivot format, significantly less complex than the one associated with the complete UML2 metamodel. Also it allows attaching alternative notations to the SPATEL language, such as a dedicated textual syntax, and still relying on a common abstract representation in memory and in persistent storage.

3 The SPICE Service Creation Environment

In this section we will describe the overall architecture of the service creation environment developed by the SPICE project, as well as, some highlights on the different components that together provide the necessary ingredients for an agile development of telecommunication services in line with the formalism defined in the previous section.

3.1 Architecture

The SPATEL Developer Studio contains four main macro constituents depicted in Figure 1:

- The *SCE Service Designer* is a graphical editor to edit SPATEL service interfaces as well as to edit the logic of composite services,
- The *Service Design Repository* is a catalog of re-usable service descriptions,
- The *Analysis and Testing tools* is basically a "native" execution engine capable of interpreting almost directly the SPATEL definitions. It is used in particular to simulate and test the services before real deployment,

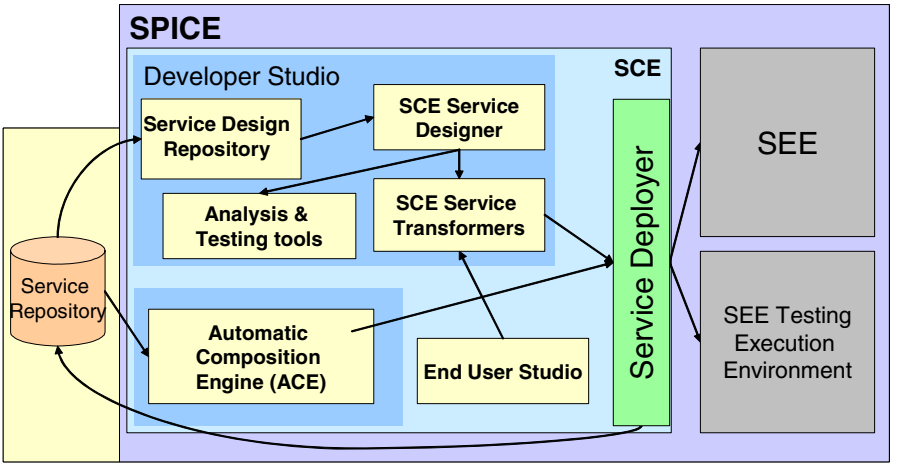

Fig. 1. Data flows between macro components in the Service Creation Environment

- The *SCE Service transformers* module represents a set of model transformers and code generators producing service implementations on top of the supported platforms.

In this figure we see a list of complementary macro components which are out of the scope of this paper: these are the Automatic Composition Engine (ACE) to create automatically SPATEL service compositions from semantically annotated SPATEL service descriptions and the End-User Studio which offers a dedicated user interface for non-professional service designers.

3.2 Implementation of the Developer Studio components

The actual implementation of the SCE Service Designer is build on top of the UML StarUML tool [16]. This component may be replaced by any other graphical facility capable of creating SPATEL XML files – the pivot format used by SCE for service definitions (see Section 2). An alternative editor based on GMF/Eclipse framework [17] is currently under construction.

The model transformations and the code generators are implemented using two alternative techniques: firstly through direct usage of the meta-modeling APIs generated from the SPATEL meta-model, and secondly using the dedicated QVT [5] model to model transformation language. Two general purpose languages are supported for the APIs: firstly Java, using the ECLIPSE/EMF [18] generated APIs for the Java language and, secondly Python, using the PYMOF [19] framework. These two APIs have in common to use the same storage format, which is the XMI format used by ECLIPSE/EMF. The SPATEL to BPEL and the SPATEL to Nokia S60 terminals are developed using the Python API whereas the WSDL to SPATEL importer is developed using the *QVT Operational* formalism by means of the SmartQVT open source tool [20]. In general QVT usage makes a transformation definition much more readable and compact. It requires however that the sources and targets are already expressed in the form of meta-models. Otherwise an extra work is needed to comply with this requirement.

The Developer Studio is connected to two other SPICE components:

- The SPICE repository, to import and export the definition of running spice services. This differs from the Design Repository, which only contains re-usable fragments of SPATEL service definitions.
- The SPICE Life Cycle Manager, to deploy and activate services in the SPICE Service Execution Environment.

3.3 Process for Using the Developer Studio

The typical steps when using the Developer Studio are:

- The service designer opens the graphical front end, initializes its design model using the available SPATEL-specific menus, such as a command to initialize model contents with some pre-defined constituents. In general the use of the tailored menus is perceived as very important by practitioners since it avoids having to deal with all UML complexity,
- The service designer imports from a catalog of pre-existing service designs the service components to be re-used in the composition. This service design repository is generally linked to the SPICE service repository which provides deployment information on the list of available running services,
- The service designer defines the interface of any additional non existing service that would need to be invoked within the logic of the composite service,
- The service designer opens the pre-initialized behavior diagram to edit the state-machine expressing the logic of the composite service,
- The service designer uses the SPATEL specific menu to generate the terminal-side code and the server-side code for one or more target technologies – for instance a BPEL engine at server side and a S60 Nokia phone at terminal side,
- The service developer completes the generated code – in general, the body of the declared operations and, if necessary, the *glue code* implementing the invocation of a given service component. Manual completion of the glue code is not needed when dealing with standardized protocols like SOAP-based web services since, in that case, the invocation code is generated automatically based on the deployment descriptors,
- The service designer and service developer executes locally the service logic using the default web interface provided by the SPATEL native execution engine. This step is useful to debug the logic and the glue code,
- The service designer iterates over these steps until it obtains the required functionality,
- The service integrator deploys the service generated files into a remote server or into a specific terminal using the Service Deployer component.

4 Use Case: E-Tourism Dinner Planning Service

We describe here the definition and the implementation of a specific context-aware *Dinner Planning Service* example which has been developed as an illustration of the model-driven approach taken to develop services.

4.1 Scenario Definition

The E-tourism dinner planning scenario is as follow:

– An End User is on travel in a city. Because he does not want to waste time trying to find a good restaurant for his dinner he will delegate this task to a specialized dinner planning service. In the morning, he sends an SMS to the Service dinner planning requesting for finding a "recommended" restaurant at 20:00 near to the location where he will at that time, and respecting some criteria (type of food),
– At dinner time (20:00) the Service locate a suitable restaurant list based on the end user geographic position,
– The Service sends a message to the End User containing the list of restaurants located in the surroundings including the contact points for reservation,
– The End User activates a call to the restaurant of choice using the restaurant contact point information.

The components that need to be in place for this scenario are:

- A Personal Agenda, to store from the user its willingness to be notified at dinner time,
- A Localization service, which will found the user's location relying on GSM network information,
- A SMS or Instant Messaging enabler to notify the user when the list of restaurants is found,
- A Yellow Pages service to found the restaurants near the location of the user,
- A Third Party Call component to activate the call to the selected restaurant.

The figure below shows the interaction between the different composed components and the orchestration engine:

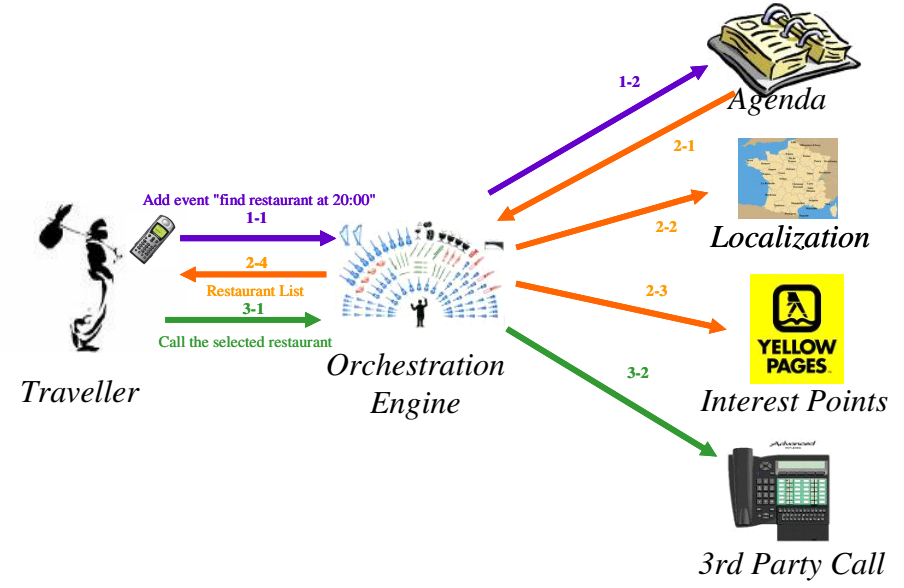

Fig. 2. Dinner planning scenario overview

From the point of view of the orchestrator, the scenario has three temporal phases:

- The orchestration engine receives the user request (1-1) and registers the event in the personal agenda (1-2),
- At dinner time, the orchestrator receives the reminder from the personal agenda (2.1) which invokes the localization services (2.2) to obtain the location information of the traveler. Then it request the interest points of the yellow pages services (2.3), collects the responses and sends the results to the traveler (2.4).
- Finally, if the user selects a restaurant, the orchestrator receives the request (3.1) and invokes the 3rd party call service to establish the communication.

4.2 Design of the Composite Service

In our experiment the SPATEL language, described in Section 2, has been used to develop the dinner planning service. In practice, following the SPATEL language philosophy this means:

- Declaring the interfaces for all the invoked components (Agenda, Localization, Yellow Pages, 3rd Party Call),
- Declaring the composite component – with a single 'orchestrate' operation – and defining the logic of this operation through a state machine.

All of the components to invoke already exist in some form. The Localization component is provided by Orange in the form of a web service, the Interest Points restaurant inspection can be obtained using an HTTP GET request on the French "Pages Jaunes" web site (after some filtering an parsing of the HTML output), the 3rd Party Call is another web service, and the agenda on line web component role can alternatively be played by Google Calendar application of a specific Orange Personal Calendar service.
So at this level, various questions arise, like:

- When a web service, is available should I directly derive the SPATEL interface from the WSDL interface or should I try to make some filtering to simplify it?
- When we have more than one candidate, should I try to define an interface that works for all the available possibilities?

Taking the WSDL file "as is" – through the WSDL to SPATEL importer – could be a comfortable solution but has some drawbacks. For instance, it could have an impact in the complexity of service logic definition, due to the fact that additional parameters - not really relevant to the designed composite service - may need to be constructed and passed anyway to have a valid service invocation.

Concerning the second issue, abstracting a common interface implies that there is the possibility to make the adaptation somewhere – maybe at deployment, when generating code from the model of the logic, or, at runtime, when executing the service through an intermediate object that performs the argument conversion. The best choice really depends on the target execution technology. When using the BPEL engine we tend to favor the first solution relying on code generator intelligence to perform the interface adaptation, since adding an intermediate web service would be costly. In the case of the Spatel Engine, for which an intermediate *local* proxy class is always generated, the second solution is much more convenient.

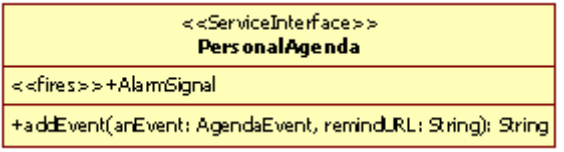

Fig. 3. Interface of the Personal Agenda component

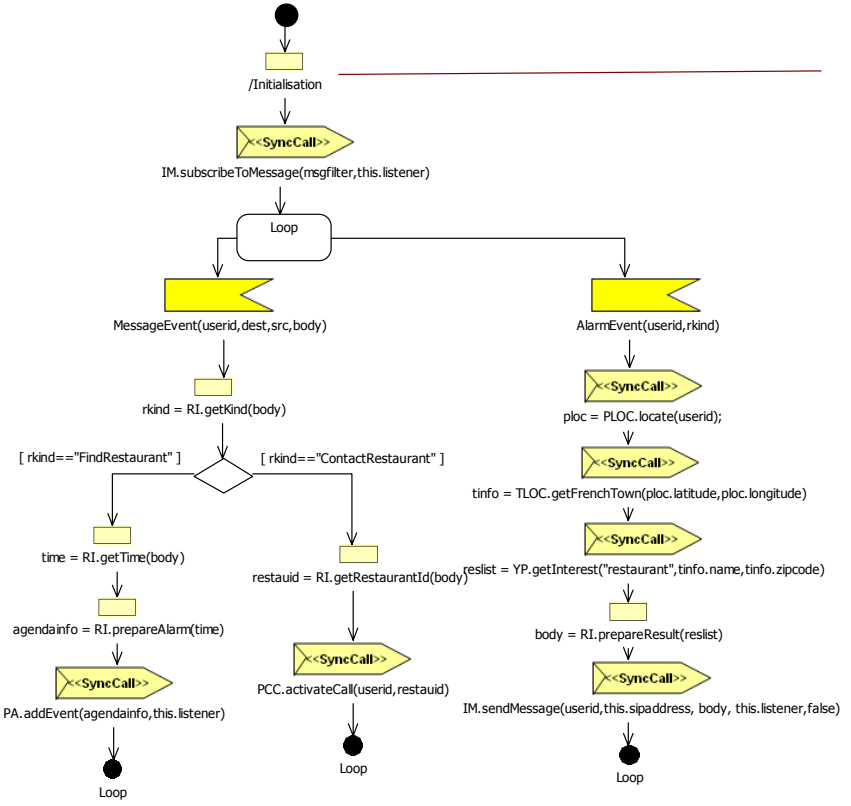

Fig. 4. Logic of the dinner planning service orchestration

In the case of the Dinner Planning service we followed the strategy of abstracting and simplifying as much as possible the interfaces of the invoked services. At the end, this had some implications in the design of the Service Repository: a unique SOAP web service may be associated to one or more registered SPATEL interfaces.

The figure below shows the interface of the Agenda component which abstracts a piece of functionality common to the Google Calendar and the Orange Personal Agenda component.

The following figure shows the modeling of the logic of the orchestration operation: we see the three threads of execution as described in the scenario definition

(in Section 5.1). On the left, we have the reception of the user initial request, on the right the treatment of the event triggered at dinner time and in the middle the final phone call. Note that this state machine uses the new UML2 transition centric view - in fact taken from ITU SDL – in which the list of actions executed during the triggering of a transition are explicitly represented as rectangles. In this diagram a specific icon is used to denote a remote service invocation, similar to an asynchronous signal sending symbol in UML. For the comprehension of this diagram, we should also mention that a Service Call in the SPATEL formalism is not an action but a State node, which gives the possibility for defining explicit exceptions transitions in case of invocation errors - overriding the default mechanism for handling errors.

4.3 Implementation and Deployment of the Composite Service

We generate two alternative implementations: one on top of the BPEL engine and the other on top of the SPATEL engine. In our development process, the implementation is the engineering phase where code generators are invoked and code completion is done when necessary. Because the state machines used in SPATEL have unambiguous execution semantics, the code corresponding to the state machine was completely generated. The part that required some manual code completion was the code related to the realization of "non standard" remote service operation calls, like the one performed to connect to Google Calendar [15] since this follows a proprietary protocol. Also all intermediate computations – like the formatting of the message containing the list of restaurants, which were modeled as invocations of local black-boxes operation calls – need to be completed, since only the skeletons were generated. The percentage of generated code in our dinner planning application was 80%. Notice however that in situations were all invoked components represent already existing components - registered as implemented components in the SPICE service repository - this generation factor may be of 100%! The richer is the catalogue of services, best are the chances to produce composite services without any code writing.
The client part for the Nokia N80 phone was generated using a simple description of the GUI in the form of Python code directly interpreted by the phone. The figure below represents the screen to activate the service.

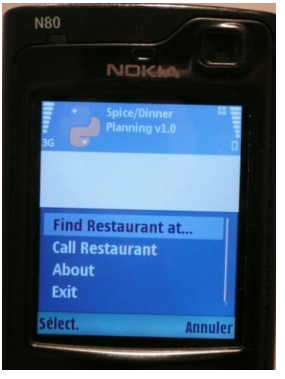

Fig. 5. Activation menu for the dinner planning service

5 Discussion

In this section, we will discuss an interesting question which concerns the legitimacy for using graphical notations like SPATEL for defining service logic. This is a controversial question and we will try to respond on the basis of our experience.

5.1 Does It Make Sense to Define Service Logic by Means of Graphic Models?

The target users of the SPATEL graphical language are professional service architects and service developers. The first population of users will probably not have to deal with the implementation tasks. However, for the second category of users, we can legitimately ask whether it make sense to develop the logic of a service using a graphical notation instead of using directly a general purpose programming language.

Our, experiments yields us to the observation that, for sure, for a programmer, using a graphical notation is much more time expensive than direct coding. However, if the time for providing an implementation is not a dramatic issue, there are clear advantages to make use of graphical notation to develop service logic that have good quality:

- Firstly, in formalisms like SPATEL, the designer is free to decide where to put the border between "graphical design" and "textual coding" of service logic: any intensive computation can be encapsulated by means of a black-box local operation. Also, some components may be completely implemented using opaque code and still have a well-defined SPATEL interface to allow its reference in other services. This emphasizes the fact that the choice between graphics and text is not black or white. The good balance between both is the responsibility of the service writer.
- Use of graphical notation helps in defining a "clean" logic, which can be understood by others, and hence facilitates the design of a logic that can be valid in different platforms. Quality of abstraction is an important feature for those that want to apply model-driven transformations to create multiple implementations from the same specification.

We believe the problem of the border between design and code will always exist. However we notice that model-driven technology is effectively pushing in the way of making more and more design and less coding and this is particularly true in the domain of service development.

6 Conclusions

Service composition has become a hot topic for all telecommunication players. The ability for professionals and, even more for end users, to compose efficiently running telecom components, depends a lot on the availability of tools capable of hiding the complexity to access the telecommunication network resources. Many initiatives are currently launched in the telecom arena to try to solve the complexity of distribution and heterogeneity, especially now that the operators tend to open their access to their network resources.

The SPICE is contributing to this challenge by developing a set of powerful inter-related tools which are integrated within its Service Creation Environment. The combined used of SOA and MDA is a distinguishing characteristic of the work conducted by this project.

In this paper we have limited the scope to the case of the service creation for developers. We presented a subset of UML named SPATEL that is designed to create services that are easily portable to different platforms at server and terminal side.

On the language side, future work will focus on a possible alignment of our formalism with the UPMS specification [21], which is still under construction at the OMG. In this respect, we tend to perceive 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 a default "service participant" containing the operation behaviours in the form of either 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 [22] or FlexLite from Adobe Technologies [23]. An interesting point for the future will be to see if the heterogeneity in mobile terminals will continue to be an issue with the emergence of a limited number of de facto Web 2.0 standards technologies in the mobile world.

References

1. OASIS, OASIS Reference Model for Service Oriented Architecture V 1.0 (August 2, 2006) http://www.oasis-open.org/committees/download.php/19679/soa-rm-cs.pdf
2. OMG, Model Driven Architecture, document ormsc/2000-11-05, (November 2000), http://www.omg.org/mda/
3. MG, Meta Object Facility V2.0, document formal/2006-01-01 (January 2006), http://www.omg.org/spec/MOF/2.0
4. OMG, Unified Modeling Language V 2.1.2, document: formal/2007-11-04 (November 2007), http://www.omg.org/spec/UML/2.1.2/
5. OMG, MOF 2.0 Query/Views and Transformations, document ptc/07-07-07 (July 2007), http://www.omg.org/cgi-bin/doc?ptc/2007-07-07
6. 3GPP, Service Requirements for the IP Multimedia System, Core Network Subsystem, release 5, document 3GPP TS 22.228 V5.6.0 (2002-2006)
7. Sun, Java 2 Micro Edition, Connected Limited Device Configuration 1.0, JSR 30, http://java.sun.com/javame/index.jsp
8. Nokia, Operating System Symbian S60, http://www.s60.com
9. Belaunde, M., Presso, J.M.: Vision for an industrial application of MDD in the Telecommunications Industry. In: ECMDA 2005 Conference. Springer, Heidelberg (2005)
10. W3C, Web Service Definition Language (WSDL), document (March 2001), http://www.w3.org/TR/wsdl
11. OASIS, Web Services Business Process Execution Language Version 2.0 (BPEL) (April 11, 2007), www.oasis-open.org/committees/wsbpel/
12. ITU-T, Specification Definition Language (SDL), http://www.itu.int/ITU-T
13. W3C: Semantic Annotations for WSDL and XML Schema, W3C Recommendation (August 28, 2007), www.w3.org/2002/ws/sawsdl/
14. W3C/VoiceXML Forum: Voice Extensible Markup Language, http://www.w3c.org/TR/2007/REC-voicexml21-20070619/ www.voicexml.org/
15. Tool Google Calendar, http://www.google.com/calendar

16. Tool StarUML, http://www.staruml.org
17. Tool Graphical Modeling Framework (GMF), http://www.eclipse.org/gmf
18. Tool Eclipse Metamodeling Framework (EMF), http://www.eclipse.org/emf
19. Tool: Python Meta Object Facility framework (PYMOF) distributed with SmartQVT tool, http://smartqvt.elibel.tm.fr/
20. Tool SmartQVT, http://smartqvt.elibel.tm.fr/
21. OMG, Uml Profile And Metamodel for Services RFP (September 2009), http://www.omg.org/cgi-bin/doc?soa/06-09-09
22. Tool : Google Android http://code.google.com/android/
23. Tool: Adobe Technologies FlexLite, http://www.adobe.com/fr/products/flex/
24. Venezia, C., Falcarin, P.: Communication Web Services Composition and Integration. In: Proceedings of International Conference on Web Services (ICWS 2006), Chicago, USA, pp. 523–530. IEEE press, Los Alamitos (2006)