Author(s): Falcarin, Paolo; Venezia, Claudio.
Article title: Communication Web Services and JAIN-SLEE Integration Challenges
Year of publication: 2008
Citation: Falcarin, P. and Venezia, C. (2008) 'Communication Web Services and JAIN-SLEE Integration Challenges' *International Journal of Web Services Research* 5 (4) 59-78
Link to published version:
http://www.infosci-journals.com/downloadPDF/pdf/ITJ4488_87d0GBTMON.pdf

Communication Web Services and JAIN-SLEE Integration Challenges

Paolo Falcarin, Claudio Venezia
Politecnico di Torino (Italy), Telecom Italia (Italy)
Paolo.Falcarin@polito.it , Claudio.Venezia@telecomitalia.it

ABSTRACT:

Meshing up telecommunication and IT resources seems to be the real challenge for supporting the evolution towards the next generation of Web Services.
In telecom world, JAIN-SLEE (JAIN Service Logic Execution Environment) is an emerging standard specification for Java service platforms targeted to host value added services, composed of telecom and IT services.
In this paper we describe StarSLEE platform which extends JAIN-SLEE in order to compose JAIN-SLEE services with Web services and the StarSCE service creation environment which allows exporting value added services as communication web services, and we analyze open issues that must be addressed to introduce Web Services in new telecom service platforms.

KEY WORDS:
JAIN-SLEE, Telecom platform, Web Service Integration, Communication Web Service

INTRODUCTION

Nowadays telecom service providers are seeking new paradigms of service creation and execution to reduce new services' time to market and increase profitability. Furthermore, convergence of networks, services and content is taking place at an increasing speed. Convergence is increasingly speeding up the introduction of new and converged services.
New market opportunities like integration of voice, video and data services are emerging from this trend; as a consequence, the main goal of telecom service providers is the development of Value Added Services, or next generation services (Licciardi, 2003) that leverage both on the Internet and on telephony networks, i.e. the convergence and integration of services offered by IT providers with telecom operators ones.
The creation of appealing value added services seems to be a key feature to avoid an operator being reduced to a "transport only" provider. The attractiveness of the service assortment offered seems to be the key to attract customers, and to increase revenues (Schülke, 2006).
The reuse and integration of existing IT services in value added ones is even made difficult both by the increasing software systems complexity and the different middleware standards used for communication.
To overcome these constraints the current vertically integrated networks are currently migrating to horizontally layered structures offering open and standard interfaces, i.e. a service platform, based on shared services and network enablers, which can be easily composed in a loosely coupled manner (Pollet, 2006).
Therefore, these goals pose new requirements on the software development process, on the platforms hosting these services, and on the middleware enabling communication among services.
JAIN-SLEE (JAIN - Service Logic Execution Environment) standard specification (JSR-22, 2007) is emerging as a new event-based service platform targeted to telecom domain, aiming at overcoming the performance limitations of J2EE-like application server, mainly designed for enterprise services, based on typical request-response interaction style.

In fact, communication services have strong real-time requirements (e.g. high throughput, low latency time), support mainly asynchronous interactions (e.g., voice-mail, call forwarding), and leverage efficiently on native protocol capabilities.
Web Services standards are emerging as a new middleware standard for providing, composing and integrating IT services (Chung, 2003), but their introduction in the telecom domain means facing up with some open issues.
Generalizing telecom functionalities to more abstract standard interfaces, like Web Services interfaces (WSDL, 2007), is often necessary to allow IT developers to reuse telecom services without mastering all technical issues related to telecom protocols; thus exposing telecom resources as Web Services means losing some of the technical details of the underlying proprietary interface.
Based on the former ideas, a Communication Application Server (named StarSLEE) inspired to the JAIN-SLEE specification has been developed, together with a graphical Service Creation Environment (StarSCE) for helping IT-developers in creating Value Added Services and Communication Web Services (Venezia, 2006).
The service lifecycle process is sped up by means of a Service Creation Environment (SCE) that supports as much as possible the reuse and the composition of pre-existing consolidated components deployed in the telecom platforms, and third-party Web Services.
A SCE must offer an intuitive interface enabling graphical composition and easy configuration of value added services, and it must automatically deploy value added services in the shape of service description languages which can be used to orchestrate and execute services running in a service execution environment.
This paper aims at analyzing the current challenges encountered during a prototyping activity carried on to provide an effective composition and integration of Web Services and Value Added Services deployed on a JAIN-SLEE platform.
In the following sections we describe the JAIN-SLEE standard architecture, the issues regarding the integration of Web Services in JAIN-SLEE, the problems of exporting a JAIN-SLEE value-added service as a communication web service, and the current issues for moving such telecom platform towards telecom service oriented architecture.

VALUE ADDED SERVICES IN JAIN-SLEE

A value added service aims at encompassing either communication or enterprise service components (Glitho, 2003).
The following is an example of a simple information retrieval target service:
1. The user invokes the service using his/her mobile phone by sending an SMS (Short Message Service) whose body contains the information needed to retrieve the closest merchant of a particular category (e.g. restaurant, bar or cinema).
2. The service localizes the user, retrieves the information requested and replies with a SMS containing the information retrieved.
3. Afterwards the user can send another SMS to be connected with the found merchant via an audio call.
The former service combines a communication service (SMS) with an enterprise service, i.e. the information retrieval services (Yellow Pages Web Service).
As communication services have particular performance and availability requirements, it is difficult to realize such service integration using a typical application server, which architecture has been mainly designed for enterprise services.

In fact, enterprise services aim at business processes, which are typically transactional and potentially long running.
Instead, communication services have strong real-time requirements and are based on asynchronous interactions. Voice mail, call forwarding and ring back tone are typical examples belonging to this service category.
Moreover while enterprise services are typically synchronous remote procedure calls (RPC) characterized by coarse-grained events with low frequency, communication services are typically asynchronous and characterized by fine-grained events with high frequency.
Within the communication domain, at the control layer, Session Initiation Protocol (Rosenberg, 2002) is considered the converging protocol for call and message signaling. Either fixed or mobile networks will leverage on SIP for providing integrated capabilities. SIP will improve the ability to build new services and will play the role that Web Services are playing in the IT world (the universal glue).
Although they play a similar role in the respective realms, SIP and SOAP are profoundly different.
For example, a SIP based communication platform (Rosenberg, 2002) is made up of a set of systems which interact through a service bus allowing information push based on a publish/subscribe model.
This platform relies on a SIP Registry which collects relevant information from a SIP network, stores and distributes it. This information regards both service and network elements descriptions.
The SIP Registry is available both for network resources (where services are running), service managers (watching services behavior) and service users (interested in invoking services).
In contrast, next generation service platforms aim at realizing an effective coexistence between enterprise and communication services.
In next sections we evaluate if Web Service technology can help reaching this objective.

JAIN-SLEE Architecture

JAIN SLEE (also known as JSLEE) aims at defining a new kind of application server designed for hosting value added services. In particular, a JSLEE container is designed for hosting communication applications while typical application servers have been designed for enterprise applications: such applications typically invoke one another synchronously (e.g. via Remote Procedure Call or Remote Method Invocation) and they usually do not consider high-availability and performance concerns.
Instead, the JAIN-SLEE specification has been designed for communication applications, and a JSLEE container relies on an event based model, with asynchronous interactions among components.
The design of a JSLEE container must meet the requirements of a telecommunication services, e.g. handling different kind of events with low latency, supporting lightweight transactions. Furthermore, a service deployed on a JSLEE container has to be composed of lightweight components with a short lifetime, which can be rapidly created, deleted and updated.
Another important feature of a JSLEE service is the ability of accessing multiple data sources with high independence of network protocols elements.
Therefore, it must be possible to deploy applications in the SLEE application environment that use diverse network resources and signaling protocols.
The integration of a new type of network element, or external system is satisfied by a Resource Adaptor Framework that supports integration of network resources; for example, a SIP server for voice-over-IP calls and instant messaging, a SMS (Short Message Service) gateway for communicating with mobile phones.

JAIN-SLEE specification requires that each telecom network resource is wrapped by a standard Resource Adaptor interface, in order to be connected to the event bus of the JAIN-SLEE container.
Figure 1 depicts the StarSLEE platform architecture, and a possible example scenario: the SIP resource adaptor may trigger the platform with events originating from the underlying SIP network; an event router dispatches these events to existing or new service instances; then a service instance is composed by various components which interact by means of events.

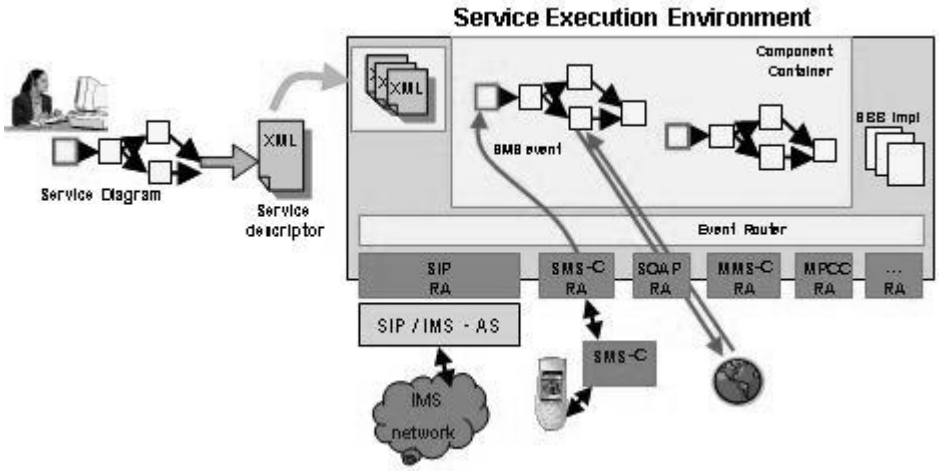

Figure 1. StarSLEE communication server

JAIN SLEE provides a standard programming model that can be used by the Java developer community. The programming model has been designed to simplify application development, promoting software reuse, and ensure that robust services can be developed rapidly with minimum configuration effort.
A standard JAIN-SLEE container should be able to clone application components between processing nodes in the system as particular processes and nodes may fail; it has to manage concurrent execution of application components, and allow application components to be dynamically upgraded. JAIN SLEE defines its own component model, which specifies how service logic has to be built, packaged, and executed, and how it interacts with external resources.

JAIN-SLEE Component Model

The JAIN-SLEE specification includes a component model for structuring the application logic of communications applications as a set of object-oriented components, and for assembling these components into higher level and more complex services.
The SLEE architecture also defines how these components interact and the container that will host these components at run-time. The SLEE specification defines requirements of availability and scalability of a SLEE platform, even if it does not suggest any particular implementation strategy.
Applications may be written once, and then deployed on any application environment that implements the SLEE specification. The system administrator of a JAIN SLEE controls the lifecycle (including deployment, un-deployment and on-line upgrade) of a service.

The atomic element defined by JAIN SLEE is the Service Building Block (SBB). An SBB is a software component that sends and receives events and performs computations based on the receipt of events and its current state.

Every SBB subscribes to a given type of event. Whenever such an event is triggered by the network, or internally by some other SBB, StarSLEE container creates SBB instances, able to manage those events by means of specific handlers.
The event router, specified by JAIN-SLEE specification, is the engine which routes event, either created by action performed on SBB, or coming from Resource Adaptor (RA).
The RA architecture provides a standard way for enabling events on different networks to be received and transmitted from the event processing engine that is the Service Logic Execution Environment. The goal of the RA architecture is to allow Resource Adaptor implementations that use and fulfill contracts defined in the JAIN-SLEE 1.1 specification to be deployed and run in any compliant JAIN-SLEE application server.

The RA layer is useful to decouple the SLEE container from rest of the world in terms of other protocols i.e. SIP protocol, SOAP protocol or others.
Each SBB is defined by its own SBB-descriptor, an XML file including information that describes it (e.g. its name, vendor and version), the list of events it can fire and receive, and the names of Java classes implementing the logic of the SBB itself.
In addition, StarSLEE the SBB descriptor contains:

- The list of SBB properties, e.g. the input action parameters;
- The list of SBB variables, e.g. the output set in the Activity Context;
- The list of SBB handlers, e.g. the events managed by SBB;
- The list of actions performed by SBB;

While SBB properties are input parameters for actions/operations performed by SBB, SBB variables are the results of those operations. Actions are triggered by events, SBB properties can be valued by SBB variables belonging to an Activity Context (AC), which is a portion of memory shared by all SBB instances running in a service instance, and that is used to read and write properties and variables of all SBB instances involved in the same service instance.
SBBs are stateful components since they can remember the results of previous computations and those results can be applied in additional computations. SBBs perform logic based on events received.
An event represents an occurrence that may require application processing. It contains information that describes the occurrence, such as the source of the event. An event may asynchronously originate from a number of different sources, for example an external resource such as a communications protocol stack, from the SLEE itself, or from application components within the SLEE.
Resources are external entities that interact with other systems outside of the SLEE, such as network elements (Messaging Server, SIP Server...). A Resource Adaptor wraps the particular interfaces of a resource into the interfaces required by the JAIN SLEE specification.
JAIN-SLEE specification does not define any particular service description language to be used for composing different SBB instance.
Therefore, we defined StarSDL, a specific service description language for StarSLEE used for describing an event-oriented service scenario, which well suits the telecommunication service domain; in particular, the StarSDL must allow asynchronous activation of the service, and the service session is typically open-ended and with long-duration, because it often involves different services, and, of course, users.

The language specification aims at satisfying the abovementioned requirements; hence it allows developers to model asynchronous interactions to the service both for service activation or and for other actors' involvement, through the event router contained in the StarSLEE execution environment.
As a set of SBB is available, the service developer can build service without worrying about single SBB programming, only considering service composition, once he knows each SBB interface, in terms of handlers, actions and properties.

Service Composition with StarSCE

The approach to service creation in telecom domain with Web Services infrastructure 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 (Baravaglio, 2005), to very specific telecommunications ones, where an overabundance of protocols and standards are available (e.g. SIP, IMS). What seems to be clear in telecom 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 a 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.
Looking at the main challenges of the service creation process, the most important requirements are:

- A service description language that allows the specification of a telecom/IT integrated services
- Tools that support the graphical composition services and their deployment to a target Service Execution Environment
- A Service Execution Environment that allows combining effectively different technologies.

In order to provide Value Added Services (VAS), the service platform must be enhanced with a graphical service composition engine, i.e. a service creation environment which easily allows building new services by means of a collection of internal components or third-party Web Services. StarSCE allows the developer to choose the SBBs (Service Building Block) and link them in a graph structure which is a graphical representation of the service. A Service Building Block is either an External IT Web Service wrapper or a signaling network functionality provider. In JAIN-SLEE component model variables and properties can be set for each SBB: StarSCE allows to visually composing those SBBs in a service description diagram which can be translated in StarSDL representation, which describes the service control flow and the usage of these variables and properties.
In particular, starting from the WSDL interface of a Web Service, StarSCE consents to automatically create the correspondent SOAP client wrapped in a new SBB.
The following figure (realized using the StarSCE graphical service creation environment) shows an example of a simple service which can be deployed on the StarSLEE service platform. Moreover given a set of Web Services wrapper SBB, StarSLEE can actually behave as a web service orchestration engine.

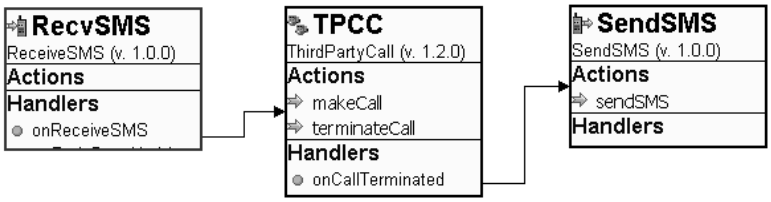

Figure 2 Example of graphical Service description

Once a service is graphically composed and the SBBs have been configured, StarSCE generates an XML file, called service descriptor.
A service descriptor represents the control-flow graph of the service composed of different SBBs, each one defined by its own SBB descriptor.
A service is made up of loosely coupled components, and it may provide different starting points, i.e. different SBB instances, each one triggered by a different kind of event, coming from the resource adaptors pool.

Each service instance is then made up of different SBB instances and one activity context holding shareable attributes that SBB instances want to share.

Therefore the state of a service instance can be represented by attributes stored in an activity context.

Using StarSCE it is possible to manage the automatic configuration, dynamic deployment, and publication of a Value Added Service in a JAIN-SLEE container.

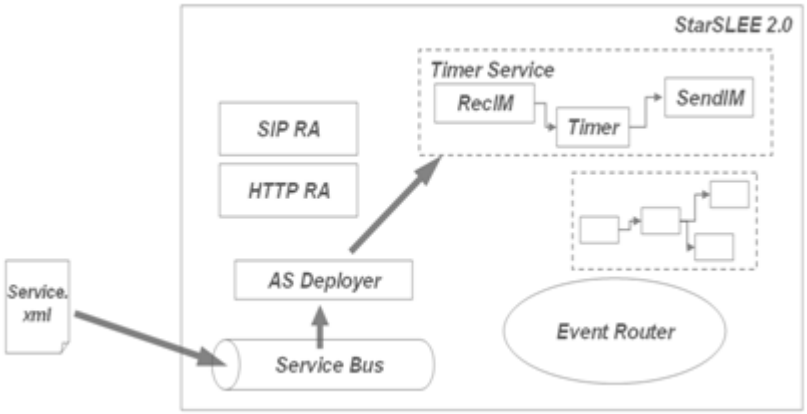

Figure 3. Service deployment on StarSLEE

Figure 3 shows main entities of StarSLEE container: a XML service descriptor is sent through the Service Bus to the Application Server Deployer, which creates the corresponding service instance (e.g. TimerService): this service is then running and listening on the event router, waiting for events coming from networks underlying the resource adaptors (e.g. HTTP, SIP).
StarSCE helps the developer in creating complex compositions of SBBs to be executed in a JAIN-SLEE container, which use Resource Adaptors to abstract interfaces of telecom networks resources, and it is not bounded abstracts and to represent the orchestration of such resources our approach aims at extending JAIN-SLEE platform and related SCE in two directions:

1. To compose both JAIN-SLEE service building blocks and Web Services.

2. To export JAIN-SLEE service as an asynchronous web service compliant to the WS-Notification specification.

COMMUNICATION WEB SERVICES

Traditional communication services are usually triggered by signaling messages like a SIP message (Rosenberg, 2002) or an instant message (IMS, 2006); instead Communication Web Services (Baravaglio, 2005) are usually Web Service interfaces of common telecom functionalities which are triggered by a SOAP message. They can also exploit different network resources within the telecom domain and be published and used on the Internet.
For example, here is a list of telecom services which can be exposed as Communication Web Services:
• Third party call: provides the capability to initiate a call between two actors generated and managed by a third party.
• Multi media conference: provides the capability to initiate an audio/video conference with two or more actors within a session.
• Messaging: a set of Web Services which provide the capability to send Instant Messages, SMS and MMS
• Presence: provides the capability to retrieve user availability information in a network domain.
• Users' provisioning: provides the capability to interact with a Data Provisioning DB System by means of retrieving and storing user profiles information supporting various communication protocols and devices.
Using standard Web Services to abstract telecom network capabilities, service providers can offer to IT developers outside of the telecom industry, new possibilities for building innovative services.
Web Services standards enable telecom operators to provide third parties with controlled, reliable access to telecom network capabilities such as presence and call control.
Different international consortia have proposed standards for using Web Services in telecommunications domain:

- Parlay X Standards (ETSI, 2007) are a set of simplified telecom APIs based on Web services published by the Parlay Group and the European telecommunications Standards Institute (ETSI); Parlay-X has been standardizing WSDL interfaces for the most common signaling services, like Third Party Call, Call Notification, SMS, MMS, etc..
- Open Mobile Alliance (OMA, 2007) has proposed the OMA Web Services Enabler to define the means by which OMA applications can be exposed, discovered and consumed using Web Services technologies, and it specifies how mobile terminal can interact with Web Services.
- W3C (W3C, 2007) is providing WS-Addressing specification (WS-Addressing, 2007), which provides transport-neutral mechanisms to address Web services and messages. Specifically, this specification defines XML elements and namespaces to identify Web service endpoints and to secure end-to-end endpoint identification in messages. This specification enables messaging systems to support message transmission through networks that include processing nodes such as brokers, firewalls, and gateways in a transport-neutral manner.
- OASIS consortium has defined a set of specifications like WS-Notification (WSN, 2007), WS-ResourceFramework, and WS-ReliableMessaging (WS-ReliableMessaging, 2007) which are important basis for developing communication Web Services.

In the following section we describe how StarSCE helps developers in transforming a JAIN-SLEE value added service in a communication web service.

From Value Added Service to Communication Web Service

Transforming a Value Added Service (in this context a JSLEE service) in a Communication Web Service requires some enhancements to the JAIN-SLEE architecture.
First of all a Web Server must be added for hosting a SOAP Server (Apache Axis, 2007). Then, for any service to be exposed, a Web Service implementation with related WSDL is provided and it has to interact with the actual service deployed in the SLEE container.
Therefore, the JAIN-SLEE architecture must be extended adding a SOAP Resource Adaptor (SOAP-RA), which acts as a communication bridge between a SLEE service and its correspondent Web Service implementation.
An alternative design may be based on adding a service-specific resource adaptor for each service to be exported as a web service, but this solution is not viable: in fact in JAIN-SLEE architecture a resource adaptor is a wrapper of an external network entity, and it is designed to be service-independent, because it must be unaware of which kind of services are deployed in the SLEE container.

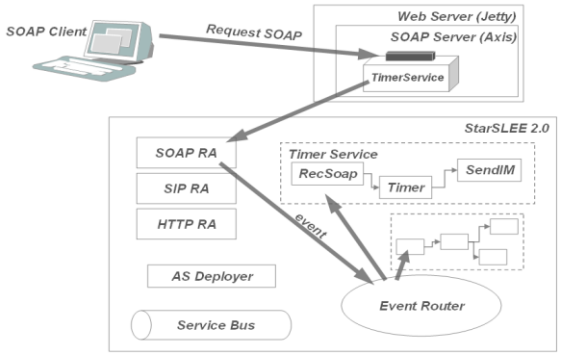

Figure 3 Extended JAIN-SLEE architecture

Once defined the new extended architecture (Figure 3), we can consider two different strategies to export a SLEE service in a Web Service: a wrapping strategy and a reengineering strategy.
Using the wrapping approach means considering the service as a single "black-box" entity which receives and sends events, while the reengineering approach consists in automatically modifying some parts of the service in order to be exported as a Web Service.
Following the wrapping strategy implies that the SOAP-RA should be able to send events the service is listening to; for example if the target service must be triggered by means of a SMS, the SOAP-RA should be able to send this event. Under these assumptions, this kind of SOAP-RA could send whichever kind of SLEE events, but this is in opposition to JAIN-SLEE design, where each Resource Adaptor must only exchange events related to its own underlying network element.
The SOAP-RA can only send events related to its own underlying network protocol, thus a SOAP-Event has been introduced to represent information coming from whichever Web Service implementation.
As a consequence, our approach is based on reengineering the value added service in order to automatically obtain its new web service version. Every service requires a root SBB which represents the service entry point (e.g. the SBB on the left side in Figure 5). Only when a root SBB is triggered a new service instance is created.
As the Value Added Service has been previously designed for listening to a particular event type, adding a new root SBB becomes necessary, i.e. the ReceiveSOAP SBB. The SOAP request is

received from the SOAP-client through the Web Service Implementation and forwarded by the SOAP RA to a root SBB by means of a SOAP event.

The new root SBB (ReceiveSOAP) is then the service entry point which extracts data from the SOAP-Request and put them in the activity context.

Figure 4. ReceiveSOAP SBB

The introduction of a new ReceiveSOAP SBB in place of the former root is not enough. In fact, reengineering a service by adding a new SBB requires a deeper analysis of service structure to find out dependencies among SBBs, both at the control-flow level and at the data-flow one.

For example, looking at the service in figure 2, we can identify different types of SBBs. The TPCC (Third-Party Call-Control) is a type of SBB representing the actual service logic implementation (we can call it "core SBB") and other SBBs whose main activity is the communication with external entities, that we call "connector SBBs". Among these ones we can further distinguish SBB receiving data (i.e. the RecvSMS which receive an SMS coming from the SMS-resource adaptor) from other ones sending out data (i.e. the SendSMS which sends an SMS to the SMS-resource adaptor): the ones receiving data can be labeled "service heads", because they are typically performed at the beginning of service execution, while the others sending data can be labeled "service tails", because they are typically performed at the end of service execution.

A service can be described by a direct cyclic graph, where a node corresponds to a SBB instance, and there is an arc from node A to node B only if the same type of event is sent by A and received by B.

Thus "service heads" are nodes with no incoming arcs, while "service tails" are nodes with no outgoing arcs.

For example in the service of figure 5 there are three service heads (ReceiveIM_0, ReceiveIM_1, and ReceiveIM_2) and two service tails (SendIM_0, and Echo_0).

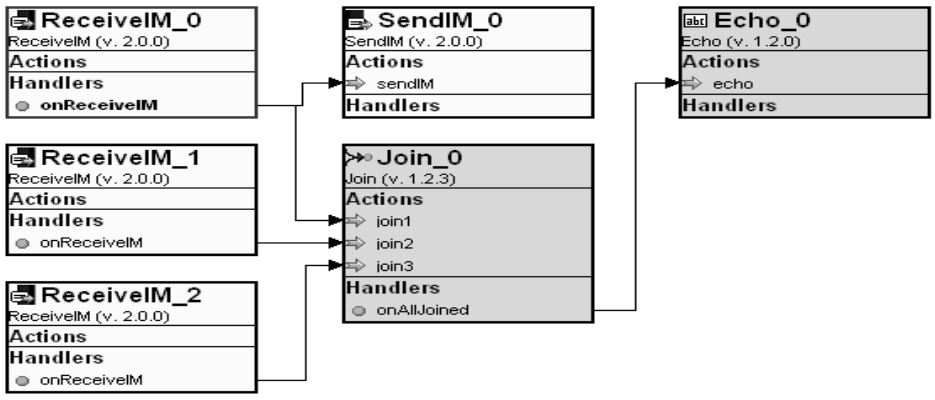

Figure 5. Service heads and service tails

On the other hand, the data-flow of a service instance can be deduced analyzing which attributes are read from (or written in) the activity context. The content of the activity context instance represents the state of the service instance at a particular time.
The attributes stored in the activity context by each SBB instance can be obtained from the XML service descriptor file.
In practice, reengineering the service to be transformed in a Web Service means making some design decisions:
1. Which service heads must be replaced by a ReceiveSOAP SBB.
2. Which service attributes in the activity context must be mapped to parameters of Web Service operations.
3. Which interaction style to use between SOAP clients and the Communication Web Service.
4. Which service attributes in the activity context must be considered as a result to be sent back to SOAP clients.
5. Depending on the chosen interaction style, how service results should be transferred to the SOAP clients.

Once the developer makes these decisions, StarSCE can automatically generate the corresponding WSDL interface, the Web Service Implementation Java code to be deployed on the Web container, and the code of the ReceiveSOAP SBB.
Once a Communication Web Service has been deployed, it is provided with as many operations as the number of the available service heads. Invoking an operation mapped to a root service head means activating an instance of the corresponding service. The user can then interact with the service instance by means of invoking operations mapped on any of the other service heads.
For example, in figure 6 the original service of figure 5 has been reengineered, applying the following changes: the root SBB has been replaced by the SBB ReceiveSOAP_1, the other service head ReceiveIM_1 has been replaced by another ReceiveSOAP SBB instance, the two service tails have been substituted two SendSOAP SBBs.

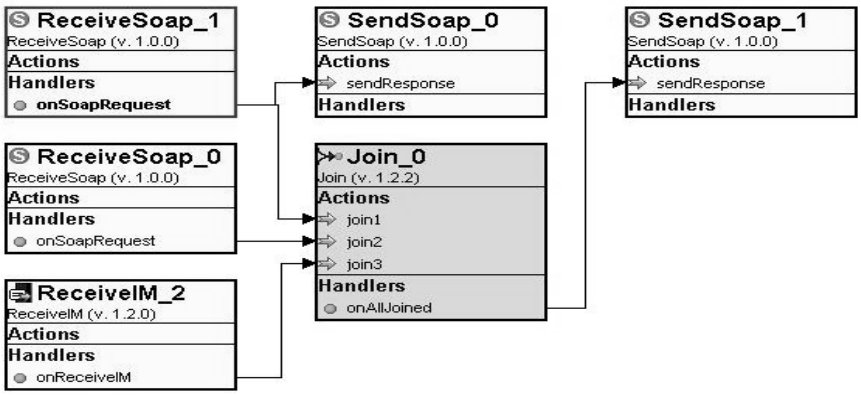

Figure 6. New Service heads and tails

A SendSOAP SBB is used to send service results back to the SOAP-RA sending a SOAP event containing attributes in the activity context, previously selected as service result.

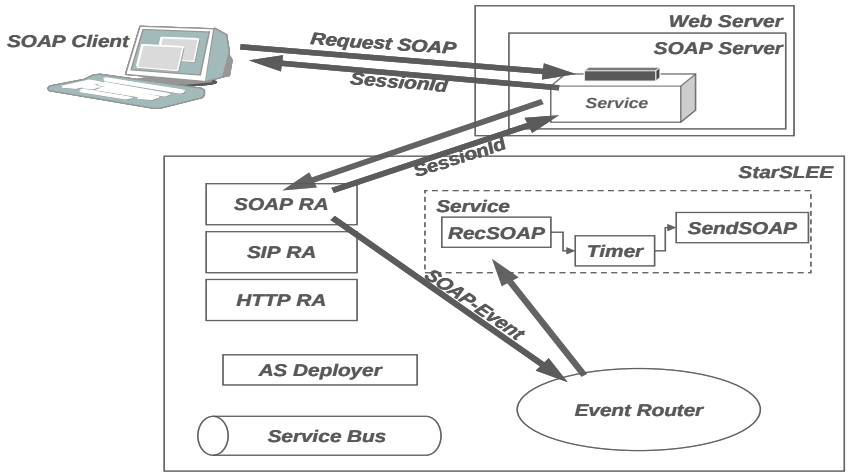

Figure 7. Web Service Request and SLEE Dispatching

Figure 7 shows how a Communication Service is accessed via SOAP:

1. The SOAP client request reaches the Web Service implementation of the SLEE Service;
2. The Web Service collects the parameters and delivers them to the SOAP RA;
3. The SOAP RA in turn identifies the corresponding reengineered SLEE Service and triggers it by generating the proper SOAP event, containing the service name and the parameters of the invoked Web Service operation;
4. The root SBB (ReceiveSOAP) receives the SOAP event, creates the service instance, and then it copies the operation's parameters in the service activity context;
5. The service is executed and results are sent to the SOAP-RA with a SOAP event.

At this point, using a synchronous interaction style implies that SOAP-RA has to provide two more operations: getStatus and getResult. Invoking the former operation, while a service instance is running, allows SOAP clients to gain information on the state of its execution, polling on the latter returns service results whenever available.

Another important feature of SOAP RA is keeping a service session. In fact a session-ID is created and delivered to the SOAP client and it has to provide it for any further operation invocation. This session-ID is used to keep the link between the Web Service client and the corresponding StarSLEE service instance.

JSLEE - WEB SERVICES INTEGRATION ISSUES

There are other implementations of JAIN-SLEE specification along with StarSLEE: the Rhino (OpenCloud, 2007) and MobiCents open source project (MobiCents, 2007). While they both provide a rich set of resource adaptors compliant to recent JAIN-SLEE specification, they do not implement a SOAP Resource Adaptor, like in StarSLEE.

Moreover they do not specify a service description language for defining JSLEE service descriptor, and, like in StarSLEE, they do not face with inter-communication between SBBs deployed on different JSLEE containers.

In particular, main lacks of JAIN-SLEE specification are the following ones:

- SBB descriptor and Service descriptor are required to be based on XML, and to contain some basic information like name, version, owner, but there is no XML-Schema to refer to.

- There is no specification on how to implement the event bus, neither how to define events namespaces, nor how this event bus can enable communications between SBBs deployed in different JAIN-SLEE containers.
- There is no specification on how to define services whose SBBs are running on different JAIN-SLEE containers, and how (i.e. with which middleware or protocol) these containers have to communicate.
- The JAIN-SLEE specification does not specify a service description language but it just suggest that SBB work-flow could be represented with a tree-structure, which limits the composition possibilities.

In order to fill these gaps, StarSLEE has been developed to implement and extend the JAIN-SLEE specification.
In this section we discuss important issues to be solved in order to obtain a smooth integration between JAIN-SLEE and Web Services world. In particular we will discuss our approach and we compare it with related work on service description languages, asynchronous interactions, and service discovery.

Service Description Languages

Telecom domain offers several service description languages, but they have been designed for domain specific applications and protocols (Licciardi, 2003); many languages have been proposed to enable service programming on telecom networks, namely: Call Processing Language (CPL) (Rosenberg, 1999), Service Creation Markup Language (SCML) (Bakker, 2002), Language for End System Services in Internet Telephony (LESS) (Wu, 2003), CCXML (CCXML, 2007), Session Processing Language (SPL) (Burgy, 2006), XTML (eXtensible Telephone Markup Language) (Pactolus, 2001).
The expressiveness of LESS and CPL have been intentionally limited to make them accessible to end-users without programming expertise, while SCML and CCXML require more technical knowledge and thus they target expert users.
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 it makes programming telephony services accessible to more programmers, but it is bound to SIP-based services, it does not foresee integration with Web Services, and it still lacks a related service creation environment, able to automatically generate code and configuration files.
Both LESS and SPL use program analysis to check particular properties on the designed service: LESS use it to detect feature interactions, SPL to check for safety and robustness properties exploiting the high-level domain-specific constructs of the language.
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.
Using a network-independent service platform like JAIN-SLEE requires the definition of a service description language not coupled to the underlying network.
With respect to the service description and creation facets, JAIN-SLEE does not specify a language to describe services. As a consequence the new StarSDL language for StarSLEE platform has been defined to enable description of value added services less coupled to the underlying network resources, thanks to JSLEE event-based architecture.
StarSDL is inspired to a well known standard language in IT world to define service orchestrations, such as Business Process Execution Language (BPEL, 2003) for WS.

Several BPEL4WS implementations (ActiveBPEL, 2007) consist in environments which allow creating, deploying, executing and monitoring BPEL4WS services. Although these solutions encompass almost all the functional requirement for creating communication Web Services they still cannot fulfill essential telecom services requirements like low latency and high throughput which implies that they do not easily scale for telecommunication Services environments.
Even if BPEL4WS language offers several workflow features, like Sequence, Parallel Split, Synchronization, Exclusive Choice, Simple Merge, Multi Choice, Synchronizing Merge, and Implicit Termination, it does not allow to define interactions based on publish-subscribe pattern: as a consequence, it is not possible to activate several process instances with the same event notification. In particular, StarSLEE execution environment requires to express and enact critical workflow patterns, like arbitrary cycles and multi-merges, not supported by BPEL language and related engines (Wohed, 2003).
Same limitations are observable in XPDL (XPDL, 2007), the standard language defined for easing interchange among business process languages in the Workflow Management domain. The goal of XPDL is more oriented to store and exchange the process diagram, in order to allow one tool to model a process diagram, another to read and edit the diagram, and another to run the process model on an XPDL-compliant engine.
The Web Services Choreography Description Language (WS-CDL, 2004) is an XML-based language that describes collaborations among different Web Services by defining, from a global viewpoint, their overall behavior, in order to achieve a common business goal.
While WS-BPEL represent the workflow orchestrated by a dominating entity (the BPEL engine running the BPEL script), WS-CDL defines the same workflow as a protocol between services which are independent peers working together to realize a collaboration: WS-CDL definition can be decomposed in different BPEL scripts, each one executed by a peer in the collaboration.
Recently, two other languages have been proposed: SOAP Service Description Language (Parastatidis, 2006) which enables contract specification on WSDL 2.0 and it is better suited to precisely specify a web service interface than representing an orchestration language; Taverna (Wolstencroft, 2005) is a data-centric workflow language which uses data dependencies to describe a workflow of GRID processes.
Any of these XML-based languages is not so useful without a related service creation environment, used to generate such languages from a high-level, possibly graphical, representation
Regarding the service creation environment, WebSphere Studio Application Developer (IBM, 2007) and RapidFLEX Application Server (Pactolus, 2001) provide a graphical SCE easing the service creation process of value added services.
The former strongly relies only on those telecom network resources exposed as Web Services compliant to the Parlay-X specification.
The latter allows creating workflows of elements representing telecom resources exposing their native interfaces or Java code snippets; such workflows are then represented with XTML files which are executed by the proprietary application server. In this platform web service invocations might be inserted manually in Java code snippets.
More specifically to JAIN-SLEE, Eclipslee (Eclipslee, 2007) is also available as a SCE for MobiCents services, but it does not support any web service facilities, as its underlying platform does not offer any SOAP Resource Adaptor.

Facing with asynchronous interactions

The aim of overcoming the limitations of SOAP toolkits in addressing asynchronous interactions with Web Services it is not just a Telecom Operator prerogative. Recent standard specification like WS-Notification (WSN, 2007) defines how to apply publish-subscribe interaction pattern among Web Services for implementing asynchronous interaction style among loosely coupled services, relying on standard SOAP protocol; for this reason there are a few implementations addressing these issues.
Apache Muse project (Muse, 2007) is a Java-based implementation of OASIS standard specifications, like Web Services Resource Framework (WSRF, 2007), Web Services Notification (WSN, 2007) and Web Services Distributed Management (WSDM, 2007).
Muse is a framework upon which developers can build web service interfaces for manageable resources: it hides the complexity of dealing with the specifications cited above. Furthermore the applications built with Muse can be deployed in both Apache Axis2 (Axis2, 2007) and OSGi (OSGi, 2007) environments. Axis2 is a recent SOAP implementation which supports standards like OASIS WS-ReliableMessaging (WS-ReliableMessaging, 2007) and W3C WS-Addressing for providing asynchronous Web services.

The main requirement of a Communication Web Service Platform is to manage asynchronous interactions with clients by means of a fully asynchronous WS management system. Along with the synchronous SOAP solution described in the former section, we have implemented a full asynchronous management of client-server interactions.
This was obtained implementing an enhanced version of the SOAP resource adaptor in accordance with WSN family of specifications. WSN defines a set of specifications that standardize the way Web Services can interact using the Notification pattern, which specify a way for consumers to subscribe to a producer for notifications whenever a particular event occurs. This set of specifications includes WS-Base Notification (WSN, 2007), WS-Topics (WS-Topics, 2007), and WS-Brokered Notification (WS-Brokered Notification, 2007).
Web Services can act asynchronously as long as they make their own state persistent. This was reached referring to the Web Service Resource Framework family of specifications (WSRF, 2007). WSRF defines a generic and open framework for modeling and accessing stateful resources using Web Services. It provides mechanisms to describe views on the state, to support management of the state through properties associated with the Web Service, and to describe how these mechanisms are extensible to groups of Web services.
A SOAP server redirects inbound messages to the said SOAP resource adaptor which in turn creates:

- A SOAP service context
- A client formal subscription to further outbound messages and
- A SOAP event to be dispatched by means of the event router.

On the other hand, whenever a service needs to contact back the client it triggers an event to the SOAP RA which calls back the client.
In the following figure is shown what happens when a service instance in the SLEE container terminates its execution: before the service instance is released it notifies the router with an End of Service event which is forwarded to the SOAP RA. Then the SOAP RA looks for the related SOAP service context, and the corresponding web service implementation is notified: at this point the web service provider notifies the Service Requestor with a SOAP message, containing the web service response.

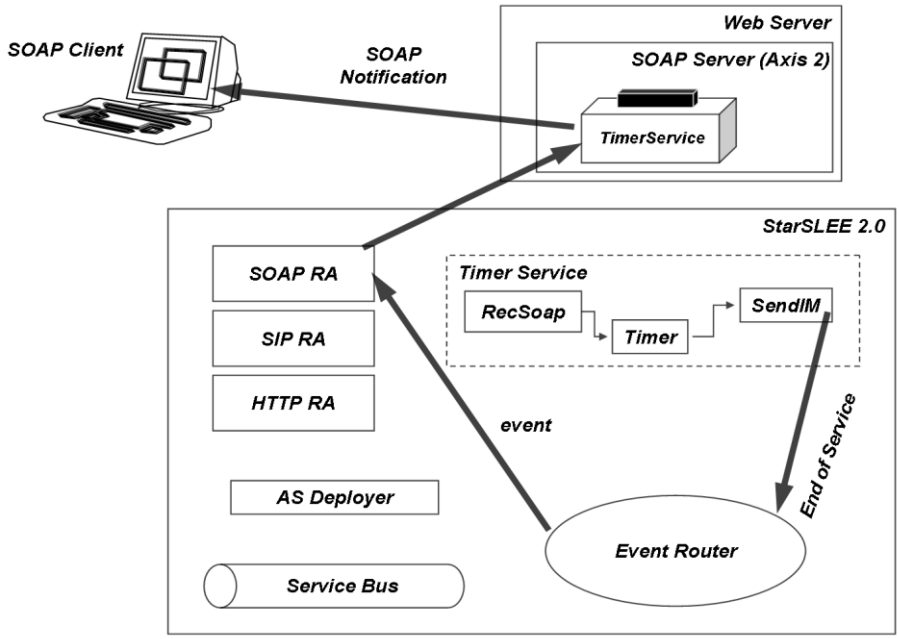

Figure 8. Web Service Response: SOAP Notification

Service Discovery

Service discovery and advertising are key facets in a telecom environment: for example, a SIP network leverages on its native publish-subscribe model to "push" new services information to clients belonging to a given network domain.

A communication service platform aiming at composing and integrating Web Services is fully concerned with static and dynamic discovery of web-services. Furthermore the discovery process has to sort candidate services that fulfils given functionality and quality parameters, and can be combined in order to realize value added services.

Therefore, new processes, methods, and tools need to be provided to extend current software development practices to support these requirements. Discovering Web Services dynamically consists in identifying alternative services to replace services already participating in a given composition that may become unavailable or fail to meet specific functional or quality requirements during service execution. It is a challenging activity since it requires efficient discovery of alternative services that precisely match the functional and quality requirements needed and replacement of these services during run-time execution in an efficient and non-intrusive way.

At its foundation, Universal Description Discovery and Integration (UDDI, 2007) is a group of specifications that lets Web service providers publish information about their Web Services on a public UDDI registry and it lets Web service discoverers or requesters search that information to find a Web Service and run it.

UDDI specification is then focused on the information model that enables a suitable categorization of the published services, but it does not address the following important requirements in telecom domain:

• Late binding: since service references are published as static data, Web Services are forced to be up and running continuously on a given URL. No dynamic instantiation of services and references is therefore possible.
• Personalization: UDDI does not support any form of personalization, i.e. the result of a specific query is the same for any requestor.
• Authorization: there is no mechanism in UDDI that allows defining and enforcing complex authorization policies for service requestors when inquiring the registry and retrieving the details of the services.
• Reference validity: UDDI does not guarantee that the service reference returned to the application (in response to a Get Service operation) really points to a Web Service.
In order to meet these requirements a "UDDI proxy" has been prototyped (see Figure 9). The proxy routes queries from a client application to the UDDI registry and provides additional and personalized capabilities, mediating the access to the actual UDDI registry.
The proxy can control the access to the information contained in the UDDI Registry allowing/denying the access, basing on a Service Requestor's Authorization Profile. The UDDI proxy is also able to dynamically create the Web Services instances, guaranteeing the existence of the Web Service, and to personalize the Web Service instances based on the Service Requestor identity.
The proxy exposes standard UDDI interfaces to the applications, so that the interactions with it are right the same as the ones with ordinary UDDI registry (i.e. UDDI clients use the same UDDI API). The solution has minimal impact on the pre-existing architecture since it does not require modifying the existing elements. In fact it only implies to add a separate node (the proxy), reconfiguring the applications by providing the reference to the new node and by configuring the UDDI registry to accept inquiries from the proxy.

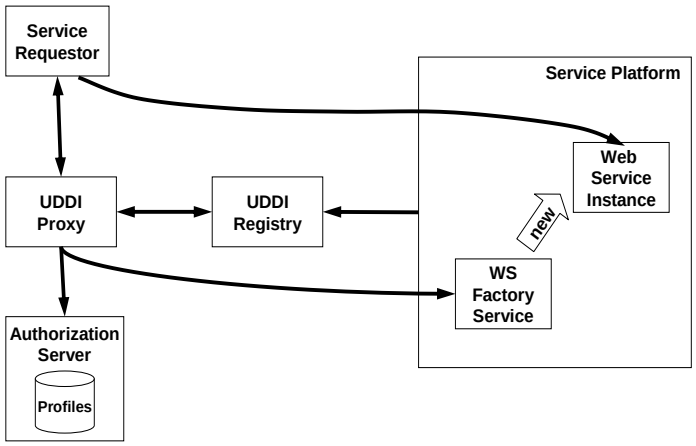

Figure 9. UDDI proxy architecture

CONCLUSIONS AND FUTURE WORK

There is an increasing interest in introducing Web Service technology in telecom service platforms. On one hand it is an opportunity to enable new business models and reach new markets, nevertheless it points out that to get to a successful applicability to telecom domain many weaknesses have still to be overcome. A communication web service platform would be

more familiar for Internet application developers, but it could imply some limitation in the usage of the network capabilities in term of provided features.
The process of integrating Web Services in telecom platforms and services has shown that the Web Service orchestration approach has some limitations: the critical requirement for publish-subscribe interaction model is not supported, even if recent standards like WS-Notification (WSN, 2007) and recent W3C submissions like SOAP over JMS (SOAP-JMS, 2008) are improving Web Service applicability to telecom platforms.
Meanwhile emerging event based containers (such as JAIN-SLEE) are designed for telecom environment but may be extended to be capable of integrating Web Services.
Our work shows both benefits and drawbacks in supplying a telecom application server (inspired to JAIN-SLEE) with Web Services facilities to enable Value Added Services composition and execution. We defined StarSDL, a new service description language to cover the lacks of JAIN-SLEE specification and we developed a SOAP resource adaptor (which is essential for exposing JAIN-SLEE service as Web Services) able to forward SOAP requests both in the typical request-response interaction style and in the emerging asynchronous one, based on the recent implementation of WSN provided by Axis 2.
At the service creation level we defined in StarSCE service creation environment a semi-automated way for generating SOAP-related SBBs, and for modifying automatically all service descriptors, in order to interact with the SOAP resource adaptor.
The SCE, together with the StarSDL language, and JAIN-SLEE will ease IT and telecom service integration, thanks to a new way of reusing components and Web Services to provide advanced value added services which can also be exported as communication Web Services. This will ease the service creation process reducing time-to-market for the new services.
Future work is focused on how to integrate recent WS standards and the rest of IT-research trends in a value added service platform, namely: Web Services discovery, dynamic Web Services composition, Web Services monitoring and management, Web Services security (Naedele, 2003), and semantic Web Services.

ACKNOWLEDGMENTS

The authors want to thank Gianpiero Fici, Carlo Alberto Licciardi, Anna Picarella, Alessia Salmeri, and Massimo Valla for their valuable contribution to this work. which has been partially funded by the European Commission, under contract IST-2002-2.3.2.3, project SeCSE (Service Centric Systems Engineering).

REFERENCES

ActiveBPEL (2007). Retrieved November 30, 2007, from ActiveBPEL project website: http://www.active-endpoints.com/active-bpel-engine-overview.htm .

Apache Axis (2006). Retrieved November 30, 2007, from Apache AXIS project website: http://ws.apache.org/axis/.

Apache Axis2 (2007). Retrieved November 30, 2007, from Apache AXIS2 project website: http://ws.apache.org/axis2/.

Bakker, J.L., & Jain, R. (2002, April). Next generation service creation using XML scripting language. In *Proceedings of The IEEE International Conference on Communications (ICC2002), New York, USA* (pp. 2001-2007). Washington: IEEE Computer Society.

Baravaglio, A., Licciardi, C.A., & Venezia, C. (2005, August). Web Service Applicability in telecommunications Service Platforms. In *Proceedings of the International Conference on Next Generation Web Services Practices (NWeSP), Seoul, Korea* (pp. 39-44). Washington: IEEE Computer Society.

BPEL (2003). *Business Process Execution Language for Web Services (Version 1.1)*. Retrieved December 5, 2007, from http://www.ibm.com/developerworks/library/specification/ws-bpel/.

Burgy, L., Consel, C., Latry, F., Lawall, J., Palix, N., & Reveillere, L. (2006, June). Language Technology for Internet-Telephony Service Creation. In *Proceedings of the IEEE International Conference on Communications (ICC2006), vol. 4, Istanbul, Turkey* (pp. 1795–1800). Washington: IEEE Computer Society.

CCXML (2007). *W3C, Voice Browser Call Control: CCXML version 1.0 specification*. Retrieved December 5, 2007, from http://www.w3.org/TR/ccxml/.

Chung, J.-Y., Lin, K.-J., & Mathieu, R.G. (2003). Web Services Computing: Advancing Software Interoperability. *IEEE Computer, 36(10),* 35-37.

Eclipslee (2007). Retrieved December 30, 2007, from Eclipslee project website, https://eclipslee.dev.java.net/

ETSI (2002), *The ETSI OSA Parlay-X 3.0 Specifications*. Retrieved December 5, 2007, from http://portal.etsi.org/docbox/TISPAN/Open/OSA/ParlayX30.html.

Glitho, R.H., Khendek, F., & De Marco, A. (2003), Creating Value Added Services in Internet Telephony: An Overview and a Case Study on a High-Level Service Creation Environment. *IEEE Transactions on Systems, Man, and Cybernetics - Part C: Applications and Review, 33(4),* 446-457.

IBM (2007). *WebSphere Telecom Web Services Server*. Retrieved December 5, 2007, from http://www-306.ibm.com/software/pervasive/serviceserver/.

IMS (2006). *3GPP TS 23.228: IP multimedia subsystem (Stage 2) standard specification*. Retrieved December 5, 2007, from http://www.3gpp.org/ftp/Specs/html-info/23228.htm

JSR-22 (2007). *JAIN™ SLEE API Specification*. Retrieved December 5, from Java Community Process website: http://jcp.org/aboutJava/communityprocess/final/jsr022/index.html.

Licciardi, C.A., & Falcarin, P. (2003). Analysis of NGN Service Creation Technologies. In *IEC Annual Review of Communications, 56* (pp. 537-551). Chicago: IEC (International Engineering Consortium).

MobiCents Project (2007). *MobiCents: The Open Source VoIP Middleware Platform*. Retrieved December 5, 2007, from https://mobicents.dev.java.net/

Muse (2007). Retrieved November 30, 2007, from Apache Muse project website: http://ws.apache.org/muse/.

Naedele, M. (2003). Standards for XML and Web Services Security. *IEEE Computer, 36(4),* 96 – 98.

OMA (2007). *Open Mobile Alliance*. Retrieved December 5, 2007, from http://www.openmobilealliance.org.

OpenCloud (2007). *Rhino 2.0 Developer Preview Release*. Retrieved December 5, 2007, from http://www.opencloud.com/products/rhino-kit/dp1/docs/index.html.

OSGi (2007). OSGi™ - The Dynamic Module System for Java. Retrieved December 5, 2007, from http://www.osgi.org/.

Pactolus (2001). RapidFLEX™ Service Creation Environment. Retrieved December 5, 2007, from http://www.pactolus.com/.

Parastatidis, S., Woodman, S., Webber, J., Kuo, D., & Greenfield, P. (2006). Asynchronous messaging between Web services using SSDL. *IEEE Internet Computing, 10(1),* 26–39.

Pollet, T., Maas, G., Marien, J., Wambecq, A. (2006, April). Telecom services delivery in a SOA. In *Proceedings of 20th International Conference on Advanced Information Networking and Applications (AINA 2006), 2* (pp. 529-533). Los Alamitos: IEEE Computer Society.

Rosenberg, J., Lennox, J., & Schulzrinne, H. (1999). Programming Internet telephony services. *IEEE Network, 13(3),* 42-49.

Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M., & Schooler, E. (2002). *SIP: Session Initiation Protocol, RFC 3261*. Retrieved November 21, 2007, from http://www.ietf.org/rfc/rfc3261.txt.

Schülke, A., Abbadessa, D., & Winkler, F. (2006, April). Service Delivery Platform: Critical Enabler to Service Providers' New Revenue Streams. In *Proceedings of World Telecommunications Congress (WTC 2006), Budapest, Hungary* .

SOAP (2007). *Simple Object Access Protocol, version 1.2*. Retrieved December 5, 2007, from http://www.w3.org/TR/soap12-part0/.

UDDI (2007). *UDDI Specification version 3.0.2*. Retrieved December 5, 2007, from http://uddi.xml.org/specification.

Valetto, G., Goix, L.W., & Delaire, G. (2005, October). Towards Service Awareness and Autonomic Features in a SIP-enabled Network. In LNCS 3854: *2nd IFIP TC6 International Workshop on Autonomic Communication (WAC 2005) , Athens, Greece,* (pp. 202-213). Berlin: Springer-Verlag.

Venezia, C., & Falcarin, P. (2006, September). Communication Web Services Composition and Integration. In *Proceedings of International Conference on Web Services (ICWS-06), Chicago, USA* (pp.523-530). IEEE press.

Wohed, P., van der Aalst, W.M.P., Dumas, M., & ter Hofstede, A.H.M. (2003, October). Analysis of Web Services Composition Languages: The Case of BPEL4WS. In *Proceedings of the 22nd International Conference on Conceptual Modeling (ER), Chicago, USA* (pp. 200-215), Springer LNCS vol. 2813, ISBN 3-540-20299-4.

Wolstencroft, K., Oinn, T., Goble, C., Ferris, J., Wroe, C., Lord, P., Glover, K., & Stevens, R. (2005, December). Panoply of utilities in Taverna. In *First International Conference on e-Science and Grid Computing (e-science), Melbourne, Australia,* (pp. 156-162). Washington: IEEE Computer Society.

WS-Addressing (2004). *W3C WS-Addressing specification*. Retrieved December 5, 2007, from http://www.w3.org/Submission/ws-addressing/.

WS-Brokered Notification (2006). *OASIS WS-Brokered Notification version 1.3 specification*. Retrieved December 5, 2007, from http://docs.oasis-open.org/wsn/wsn-ws_brokered_notification-1.3-spec-os.pdf.

WS-CDL (2005). *Web Services Choreography Description Language Version 1.0*. Retrieved December 5, 2007, from http://www.w3.org/TR/ws-cdl-10/.

WS-ReliableMessaging (2007). *OASIS WS-Reliable Messaging version 1.1 standard*. Retrieved December 5, 2007, from http://docs.oasis-open.org/ws-rx/wsrm/200702/wsrm-1.1-spec-os-01.pdf.

WS-Topics (2006). *OASIS Web Services Topics version 1.3 specification*. Retrieved December 5, 2007, from http://docs.oasis-open.org/wsn/wsn-ws_topics-1.3-spec-os.pdf.

WSDL (2007). *Web Service Description Language version 1.1 specification*. Retrieved December 5, 2007, from http://www.w3.org/TR/wsdl.

WSDM (2006). *OASIS Web Services Distributed Management standard specification*. Retrieved December 5, 2007, from http://www.oasis-open.org/committees/wsdm/.

WSN (2007). *OASIS Web Service Notification specifications version 1.3*. Retrieved December 5, 2007, from http://www.oasis-open.org/committees/wsn.

WSRF (2007). *OASIS Web Service Resource Framework standard specification*. Retrieved December 5, 2007, from http://www.oasis-open.org/committees/wsrf.

Wu, X., & Schulzrinne, H. (2003, May). Programmable end system services using SIP. In *Proceedings of The IEEE International Conference on Communications (ICC 2003), Anchorage, Alaska, USA* (pp. 789-793). IEEE press.

XML (2006). *Extensible Mark-up Language specification 1.0 (4th Edition)*. Retrieved December 5, 2007, from http://www.w3.org/XML/.

XPDL (2007). Workflow Management Coalition: XML Process Description Language, version 2.0 standard. Retrieved December 5, 2007, from http://www.wfmc.org/standards/xpdl.htm .

ABOUT THE AUTHORS

Paolo Falcarin is research assistant in the Software Engineering Group at the Department of Computer Science and Automation of Politecnico di Torino, one of the leading engineering universities in Italy. He received his M.S. degree in Computer Science Engineering in 2000, and his Ph.D in Software Engineering, in 2004, from Politecnico di Torino.
He is involved in European projects, part of the FP6 EU research program on service engineering, working on service creation technologies and service description languages. His current research interests include automated software engineering, service engineering, software modeling, Aspect-Oriented Programming, and Rule-based Programming.

Claudio Venezia is researcher at Telecom Italia Lab since 2002. He received a degree in Economics with an experimental addressing in computer science along with further computer science certifications from University of Turin (Italy) in July 1998. He worked for three years in Ernst and Young critical technologies in several domains (Banking, Automotive, and E-commerce). He has been contributing to standardization activities (JAIN-SLEE, W3C) and internal and international Projects. His research interests include enhanced SOA paradigms meshing up IT and telecom capabilities, (Semantic) Web Services technologies and XML-based languages towards Web 2.0.