Author(s): Venezia, Claudio; Falcarin, Paolo.
Article title: Communication Web Services Composition and Integration
Year of publication: 2006
Citation: Venezia, C; Falcarin, P. (2006) 'Communication Web Services Composition and Integration' IEEE Proceedings of International Conference on Web Services (ICWS-06), September 2006, IEEE Press pp. 523 - 530
Link to published version: http://dx.doi.org/10.1109/ICWS.2006.42
DOI: 10.1109/ICWS.2006.42

Communication Web Services Composition and Integration

Claudio Venezia
Telecom Italia Lab
claudio.venezia@telecomitalia.it

Paolo Falcarin
Politecnico di Torino
paolo.falcarin@polito.it

Abstract

Nowadays, the development of services that span over both the Internet and telephony networks is driving significant efforts towards the integration of services offered by IT providers with telecom operators ones .Web Services have often been recommended for providing, composing and realizing Telecom services but introducing them means facing up with several challenges.

This work sharpens benefits and drawbacks of Web Service applications within a Telecom environment focusing in particular on JAIN SLEE architecture, which defines a standard environment targeted at communication-based applications..

1. Introduction

Nowadays telecom service providers are seeking new paradigms of service creation and execution to reduce new services time to market and increase profitability.

The main goal of telecom service providers is the development of Value Added Services (or next generation services [1]) that leverage both on the Internet and on telephony networks, i.e. the integration of services offered by IT providers with telecom operators ones.

The main problem is that development of telecom services has always been constrained by proprietary interfaces, which increase development and maintenance costs.

To overcome these constraints the current vertically integrated networks are expected to migrate to horizontally layered structures offering open and standard interfaces; consequently, these goals pose new requirements on the software development process, on the platforms hosting these services, and on the middleware enabling communication among services.

Moreover the reuse and integration of existing IT services is even made difficult by the increasing software systems complexity and the different middleware standards used for communication.

These integration problems can be addressed by Web Services standards [12,15,16], which are emerging as a new middleware standard for providing, composing and integrating IT services [10], but their introduction in the Telecom domain means facing up with several challenges.

This paper aims at showing the research and prototyping activity carried on to provide an effective composition and integration of Value Added Services and Web Services.

2. Towards Value Added Services

A value added service [3] aims at encompassing either communication or enterprise service components.

The following is an example of a simple information retrieval target service:

1. The user invokes the service by sending an SMS 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.

IEEE International Conference on Web Services (ICWS'06)
0-7695-2669-1/06 $20.00 © 2006 IEEE

Moreover while enterprise services are typically synchronous (RPC Calls) 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 [4] 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 (WSDL [16] and SOAP [15]) 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 [8] 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 would help reaching this objective.

3. Communication Web Services

Communication services are usually triggered by signaling messages like a SIP INVITE or an Instant message. Communication Web Services [5] 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 widely published and used on the Internet.

For example, this is the list of Communication Web Services provided in compliance with the relevant standards (whenever available):

• 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.

OMA [14] and ParlayX [11] have been specifying Web Services interfaces to the most common Telecom functionalities.

Adapting Telecom functionalities to standard interfaces (APIs) is not trivial and means losing the granularity of the proprietary interface. In other words we gain in terms of service interoperability what we lose in terms of service capability.

However, the main issue is that most of the specifications don't provide support for one of the main requirement of a Telecom service: asynchronous interaction.

As a consequence, besides the Service Oriented Architecture model there is an increasing interest in Event Oriented Architectures, based on asynchronous interactions.

In order to obtain fully asynchronous Communication Web Services, it is crucial having both an event-based service platform and an event-based middleware.

For example, the new emerging standard JAIN-SLEE (Service Logic Execution Environment) [7] aims at designing an event-based service platform, overcoming the limitations of J2EE-like application server, designed only for enterprise services.

On the other hand, many standardization efforts are currently going on for extending SOAP for implementing asynchronous interaction style.

Based on the former ideas, a Communication Application Server (named StarSLEE) inspired to the JAIN-SLEE specification has been developed, together with a Service Creation Environment (named StarSCE) for creating Value Added Services and Communication Web Services [5].

In the following sections we describe the JAIN-SLEE standard architecture and the issues regarding the transformation of a JAIN-SLEE service in a web service.

4. The JAIN-SLEE Architecture

JAIN SLEE aims at defining a new kind of application server tailored for deploying value added services.

In particular, a SLEE container is designed for hosting communication applications while typical application servers have been designed for enterprise applications.

Enterprise applications strongly rely on databases and they are made of heavyweight objects (e.g. EJB) with a persistent lifetime, transferring data by means of slow transactions

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 SLEE specification has been designed for communication applications, a SLEE container relies on an event based model, with asynchronous interactions among components.

The design of a SLEE 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 SLEE 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 SLEE 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.

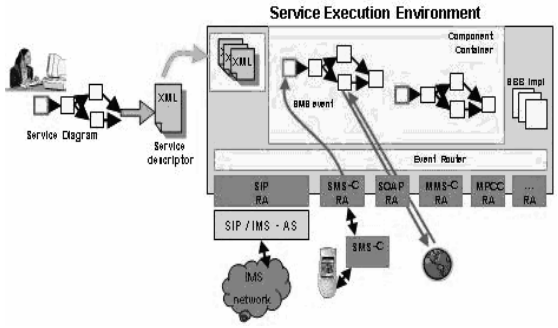

Figure 1. StarSLEE communication server

Figure 1 depicts the StarSLEE platform architecture, which implements the JAIN-SLEE specification: the SIP resource adaptor triggers the platform with events originating from the underlying SIP network. An event router dispatches these events to existing or new service instances. A service is composed by various components which interact by means of events. JAIN SLEE provides a standard programming model that can be used by the Java developer community. The programming model has been designed to simplify the work of the application developer, 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.

5. 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 complicated 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.

Each SBB is defined by its own SBB-descriptor, an XML [6] 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. 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.

6. Service composition with StarSCE

In order to provide Value Added Services (VAS) the service platform must be enhanced with a composition engine, i.e. a service creation environment [3] which easily allows building new services by means of a collection of components. 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 signalling network functionality provider. 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.

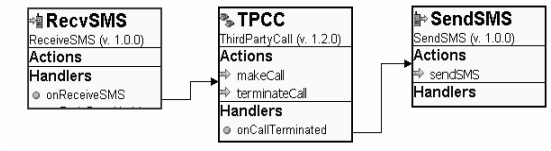

Figure 2. Service description

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.

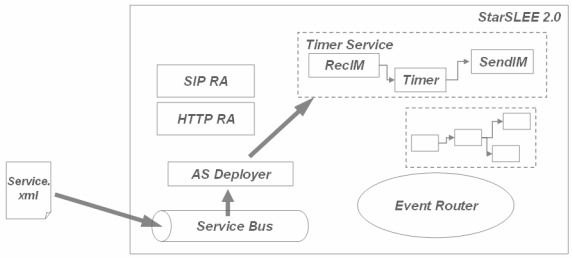

Figure 3. Service deployment on StarSLEE

Figure 3 shows main entities of StarSLEE container: a XML service descriptor is sent through th 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).

Telecom domain offers several service description languages [1], but they have been designed for domain specific applications and protocols [2]. In the IT domain, service composition has also been investigated by the standard body OASIS [13], which has specified a language to describe orchestration, namely Business Process Execution Language (BPEL) for WS.

Web Service Orchestration is a standard way to describe interactions and connections among Web services defining a higher level business process. This language is suitable to describe a workflow that is executed on a central BPEL engine, which controls execution and message flow.

Our evaluation of BPEL4WS language and related service execution environments emphasized some drawbacks in the implementation of many Communication services using a workflow paradigm based on Web Service:

• BPEL4WS suits long running business processes with loose requirements in terms of performance, while most of telecom services have strong performance requirements such as low latency time and high throughput typically not met by BPEL engines.

• Component interactions are strongly asynchronous and Web Services still lack in supporting this interaction model.

7. Towards Communication Web Service

Transforming a Value Added Service (i.e. a SLEE service) in a Communication Web Service requires some modifications to the JAIN-SLEE architecture.

First of all a Web Server must be added for hosting a SOAP Server (e.g. AXIS [17]). 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.

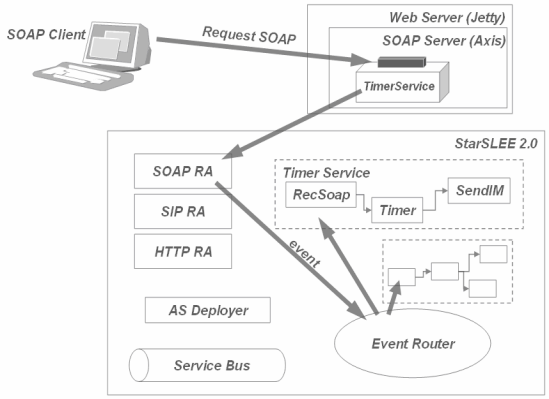

Figure 4. Extended JAIN-SLEE architecture

Once defined the new extended architecture (Figure 4), 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, meanwhile 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.

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 red bordered SBB 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 5. 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 which 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 receives 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 6 there are three service heads (ReceiveIM_0, ReceiveIM_1, and ReceiveIM_2) and two service tails (SendIM_0, and Echo_0).

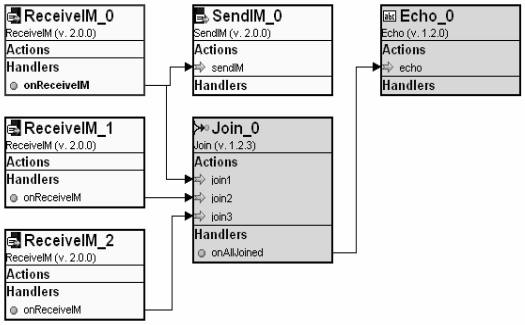

Figure 6. Service heads and service tails

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 7 the original service of figure 6 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.

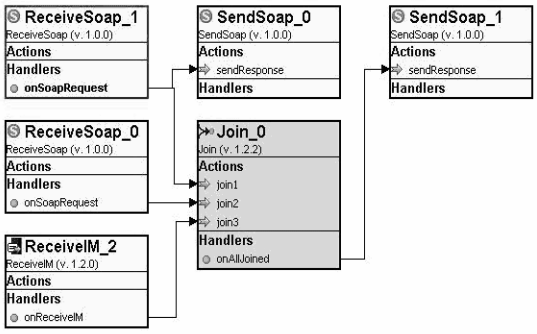

Figure 7. New Service heads and tails

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.

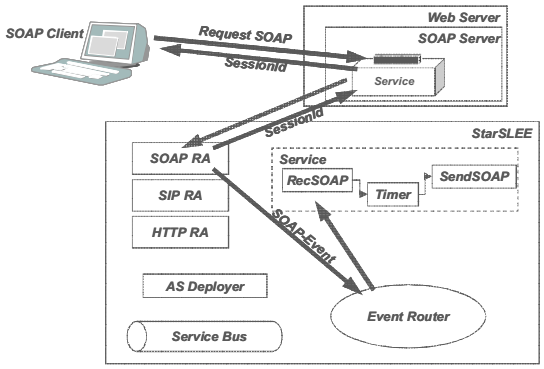

Figure 8. Example scenario

Figure 8 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.

8. Service discovery

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

The IT domain still lacks in discovery standards and solutions. Nevertheless 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, UDDI is a group of specifications that lets Web service providers publish information about their Web services 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.

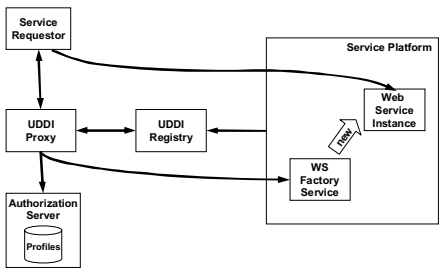

Figure 9. UDDI proxy architecture

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.

9. Conclusions and Future Work

There is an increasing interest in introducing Web Service technology in telecom service platforms, but to get to a successful applicability to this 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 limitations in the usage of the network capabilities in term of provided features.

Meanwhile emerging event based asynchronous engines (such as JAIN-SLEE) are designed for telecom environment but capable of integrating also 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.

Aside the SOAP solution described in this work, we are currently working on an enhanced version of the SOAP resource adaptor in accordance with Web Service Notification [18] family of specifications, which 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.

Web Services can act asynchronously as long as they make their own state persistent. This was reached referring to the Web Service Resource Framework (WSRF) family of specifications [17], which defines a generic and open framework for modeling and accessing stateful resources using Web services.

In the asynchronous scenario, a SOAP server redirects inbound messages to the SOAP-RA, which creates a SOAP event to be dispatched by means of the event router of the JAIN-SLEE container. Then, whenever a service needs to contact back the client it triggers an event to the SOAP RA which calls back the client.

Acknowledgments

The authors want to thank Roberto Antonini, GianPiero Fici, Anna Picarella, and Alessia Salmeri 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).

10. References

[1] C.A. Licciardi, and P. Falcarin, “Analysis of NGN service creation technologies”, Annual Review of Communications vol. 56, IEC, 2003, pp 537-551.

[2] C.A. Licciardi, and P. Falcarin, “Next Generation Networks: The services offering standpoint”. In Comprehensive Report on IP services, Special Issue of the IEC, 2002.

[3] Glitho, R.H., Khendek, F., De Marco, A., “Creating Value Added Services in Internet Telephony: An Overview and a Case Study on a High-Level Service Creation Environment”. In IEEE Transactions on Systems, Man, and Cybernetics - Part C: Applications and Review, Vol. 33, n. 4, November 2003.

[4] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, E. Schooler, SIP: Session Initiation Protocol, RFC 3261, June 2002.

[5] A. Baravaglio, C.A. Licciardi, C. Venezia. “Web Service Applicability in Telecommunications Service Platforms”. In Proc. of the International Conference on Next Generation Web Services Practices, Seoul, Korea, August 2005

[6] XML (eXtensible Mark-up Language) specification. On-line at http://www.w3.org/XML/

[7] JAIN SLEE (JSLEE) v1.1, Java Specification Request (JSR) 240, 2005, http://www.jcp.org/jsr/detail/240.jsp

[8] G. Valetto, L.W. Goix, G. Delaire, “Towards Service Awareness and Autonomic Features in a SIP-enabled Network”. In Proc. of the Workshop on Autonomic Communication (WAC2005), Athens, Greece, October 2005.

[9] “Introduction to UDDI: Important Features and Functional Concepts”, OASIS, October 2004

[10] J. Chung, K. Lin, R. Mathieu, “Web Services Computing: Advancing Software Interoperability”, IEEE Computer, October 2003, pp 35-37.

[11] The Parlay Group: Parlay X Working Group, “Parlay X Web Services White Paper”, The Parlay Group, 2002.

[12] UDDI Spec Technical Committee, “UDDI Version 3.0.2”, OASIS, 2004

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

[14] Open Mobile Alliance (OMA) specifications. On-line at http://www.openmobilealliance.org

[15] SOAP (Simple Object Access Protocol) specifications. On-line at http://www.w3.org/TR/soap/

[16] WSDL (Web Service Definition Language) specification. On-line at http://www.w3.org/TR/wsdl

[17] Web Service Resource Framework specifications. On-line at http://www.oasis-open.org/committees/wsrf

[18] Web Service Notification specifications. On-line at http://www.oasis-open.org/committees/wsn