Web and Telco Service Integration: A Dynamic and Adaptable Approach

Julián Rojas, Leandro Ordóñez-Ante, Juan Carlos Corrales
Telematics Engineering Group
University of Cauca
Popayán, Colombia

***Abstract*—The current evolution of the Web, known as Web 2.0 and characterized by providing a diverse global service ecosystem, has marked a change in the role played by telecom operators. In order to maintain high competitive market dynamism and generate new revenue sources, many operators seek to leverage the wide variety of existing Web services and integrate them with its infrastructure capabilities. Such integration leads to various challenges from a technological perspective, where the heterogeneity on networks and the need for highly qualified personnel for the development of these services are highlighted. This paper is propose the definition of a mechanism for integrating both Web and Telco services which facilitates and speeds the development of new services, considering the dynamic conditions of its execution.**

Keywords*—Web Services; Telco Services; JAIN SLEE; Integration; Adaptation

I. Introduction

Currently there is a trend in the telecommunications industry that has created a scenario in which a new model known as Telco 2.0 [1] has been defined. This model relates the concepts, services and Web 2.0 technologies with traditional telecommunications features (Telco services), allowing operators to expand their service portfolio and have a greater impact on the market by reaching end-users with more complex and personalized services. These new kind of services are known as converged services due to its integration of functionalities form the telecommunications domain (voice, video and data) with Information Technology (IT) services from the Web domain.

Such integration demands complex and robust platforms that support the interoperability of different technologies and communication protocols, characteristic of services from both Telco and Web domains. Different approaches have been proposed such as the SIP Servlets Specification [2], the Ericsson Converged Service Studio [3], Alcatel-Lucent uReach CSF (Converged Services Framework) [4], among others. An alternative that stands out is the JAIN SLEE Specification [5], which proposes a standard and robust environment for the creation and execution of converged services, meeting the rigorous performance requirements typical of Telco services (high availability, low latency, asynchronous behavior, etc.) and allowing its integration with Web technologies.

Generally Telco services belonging to an operator reside on platforms located within its network, where they are managed and executed. On the other hand, Web services tend to be distributed applications that belong to 3rd party providers, therefore integrating them to the operator infrastructure, requires the development of modules that represent them and manage the data interaction present during their invocation. This implies an extra effort on developing terms considering that, for each Web service that wants to be integrated, a module must be designed and built.

Another issue related to the integration of Web and Telco services over an operator network is that due to the distributed nature of Web services, when failures occur on runtime, the operator usually does not have access to the platforms where Web services reside to address and correct them. Such failures become a major issue and could compromise the correct operation of converged services that relay on those Web services. This issue poses the need for operators to implement contingency mechanisms that adapt to Web service invocation failures at runtime, in order to provide high quality converged services.

This paper introduces the definition of a mechanism for integrating Web and Telco services over a converged platform, specifically JAIN SLEE. The proposed mechanism facilitates the inclusion of Web services into telecommunications networks by defining an interaction model between JAIN SLEE and Web environments that allows integrating different Web services without having to develop additional modules for each Web service that wants to be integrated. The interaction model also considers the dynamic conditions of Web services invocation by detecting possible failures at runtime and adapting to alternative ones, defined by the converged service designer at creation time. The rest of the paper is arranged as follows. Next section presents a conceptual base of the different technologies related with this work. Section 3 describes different proposals that address the integration of Web and Telco services. On section 4 is presented a detailed description of the proposed integration mechanism. Section 5 presents the fault handling and adaptation functions included in the mechanisms. Section 6 presents a case study through which the proposed mechanism is evaluated. Finally, on section 7 are presented the conclusions and future work.

II. Background

The integration of Web and Telco services contemplates the use of diverse technologies that comprise both domains. Telecommunications protocols such as SIP, SS7 stack and

SMPP, and Web frameworks like WSDL/SOAP services and Apache Axis2 are among the technologies that must be taken into consideration. Is also important the platform that supports the integration of these services, which determines how converged services are built and executed. As stated before, the selected platform for developing this proposal is the JAIN SLEE specification. Next is presented a description of the main technologies and concepts that comprises this proposal.

A. Web Services

According to the W3C (World Wide Web Consortium) definition, a Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards [6]. Following such definition, Web services can be characterized as distributed software components which can be described, published, discovered and invoked with standard protocols. Web services communicate using XML and Web protocols, working internally and across the Internet. They support heterogeneous interoperability and use SOAP for service calls and WSDL for service descriptions [7].

B. Apache Axis2

The Apache Axis2 project is a Java-based implementation of both the client and server sides of the Web services equation. Designed to take advantage of the lessons learned from Apache Axis 1.0, Apache Axis2 provides a complete object model and a modular architecture that makes it easy to add functionality and support for new Web services-related specifications and recommendations [8].

Among the functionalities provided by Apache Axis2 is the creation of implementation classes for both the server and client using WSDL documents. This is a significant advantage for dynamic invocation of Web Services considering that Web service clients usually are hard coded implementations that cannot be modified at runtime and are designed to deal with the invocation and data interaction of a single Web service. Therefore, Apache Axis2 is able to infer data requirements and interactions of Web services from its WSDL description documents and dynamically implement the necessary classes for its invocation at runtime.

C. JAIN SLEE Specification

JAIN SLEE aims at defining a new kind of application server designed for hosting carrier-grade Telco services. In particular, a JSLEE container is designed for hosting communication applications while typical application servers have been designed for enterprise applications and they usually do not consider high-availability and performance concerns. JSLEE containers relay on an event based model, with asynchronous interactions among components [9]. The atomic element define by JSLEE is the SBB (Service Building Block).

An SBB is a software component that sends and receives events, and performs computations based on the receipt of such events and its current state. Events are used to represent occurrences of importance that may occur at arbitrary points of time. An event may asynchronously originate from different sources such as communications protocol stacks, network elements or from application components within the SLEE. The SLEE deals with those events through elements called resource Adaptors which adapt the particular interfaces of an external resource into the interfaces and requirements of the SLEE [10].

D. TelComp 2.0 Project

The present work is framed within the Project TelComp 2.0: *Retrieval and Composition of Complex Components for the Creation of Telco 2.0 Services* [11], funded by COLCIENCIAS and developed by the Telematics Engineering Group of the University of Cauca. The TelComp 2.0 project proposes the generation of a platform aimed to support the process of creation, composition and execution of new converged services, providing developers with tools that allow them to articulate atomic services (Web/Telco) over a unified environment for defining new value-added functionalities. TelComp 2.0 execution environment is based on the JAIN SLEE specification which must support the execution of Telco services and its integration with Web services. Therefore, the present proposal represents a major contribution to this project by defining a dynamic and adaptable mechanism for integrating Web and Telco services that facilitates and allows automating the creation of converged services.

III. Related Works

There are several proposals that address the integration of Web and Telco services, defining interaction models that mainly focus on representing and exporting Telco service functionalities as Web services so they can be composed in business processes platforms such as BPEL (Business Process Execution Language) or ESB (Enterprise Service Bus).

One of these proposals is the Parlay X specification which details a set of simple web services that can be used as building blocks for telecom applications. The key design point for all of these web services was simplicity. Parlay-X combines sets of communications functions into useful but non application specific building blocks. The capabilities are restricted to those which can be performed with a single SOAP message exchange, since this simplifies use for the non-professional programmer. The Parlay-X functions are documented in a self-documenting XML interface [12]. Following this line of development, in [7] is proposed a mechanism to encapsulate Telco functionalities running on JAIN SLEE platforms as WSDL/SOAP Web services, for them to be integrated with other Web services through an ESB. However, such encapsulation enforces a synchronous behavior of Telco services and does not allow managing communication sessions during the execution of converged services. Another approach is the one presented in [13] where a set of design patterns to represent Telco services behavior are proposed. The patterns are implemented using Parlay X interfaces and composite applications are built over a BPEL engine. This work address the complexity of managing Telco service transactions on a synchronous environment but this requires the extension of the BPEL standard by adding additional parameters that consider the asynchronous nature of Telco services, significantly increasing the complexity of implementing such system.

On [14] is presented a description and analysis of different approaches to formally describe Telco service functionalities for its integration on Web based execution environments. This work presents an UML based modelling approach to describe Telco services, that considers asynchronous behavior but does not address execution or integration issues. Another type of analysis is made on [15] where the advantages and disadvantages of different programming frameworks are considered for integrating Web applications on IMS (IP Multimedia Subsystem) based telecommunications architectures. This work presents a more convenient approach from operators perspective, considering that Web applications are integrated into its telecommunications infrastructure without having to design or implement Web based platforms for composing new converged services. However, the analysis made mainly focus on the programming complexity of using languages such as Java or Ruby for developing Web applications that could be integrated into IMS environments, but does not specifies how such integration should be carried out. It is also important to highlight the work presented in [16] where a model to describe and export Telco Services as Web services is defined. Such model is based on the development and inclusion of a SOAP resource adaptor to a JAIN SLEE environment which receives invocation requests for activating Telco services on the SLEE. As pointed out before, exporting Telco services as Web services allows to integrate them into Web based composition engines but does not permit to manage asynchronous transactions from the converged services execution flow perspective. Finally, the work presented in [17] defines a converged platform for executing composed services, comprising both Web and Telco domains. In this approach, integration is achieved through a composition engine which manages service invocations and data interactions in a centralized manner. However, undesired scenarios during the execution of converged services, such as possible invocation failures are not considered. Without relying on adaptable mechanisms, problems presented at runtime may cause major failures, preventing converged services to carry out its purposes.

Related works show a strong trend towards the integration of Web and Telco services, by representing and describing Telco services as Web services through initiatives such as Parlay X, so they can be included in Web based composition engines. Our approach pretends to achieve such integration by defining an interaction mechanism to include Web services into a Telco service capable environment like JAIN SLEE, where asynchronous behavior and transaction based interactions can be managed and executed. This type of integration helps to reduce the complexity for operators to develop converged services by not having to include Web based platforms within its network infrastructures and using additional interfaces for describing its Telco capabilities. We also consider the dynamic behavior of Web service invocations by adding failure detection and adaptive capabilities to the proposed interaction mechanism.

IV. Integration Mechanism

As stated before, this proposal is framed within the project TelComp 2.0. Within this project a generalized structure for building converged services has been defined, as shown on Figure 1. The execution logic of a converged service is managed by an orchestrator SBB which communicates with a set of atomic services (Web or Telco) through firing and receiving events. These events may be provided by resource adaptors or defined by each atomic service as custom events developed for specific tasks. An entity called *Event Router*, defined by the JAIN SLEE specification, is responsible for delivering events to its corresponding destinies.

Telco services implementation reside along with converged services in the JAIN SLEE environment due to the capabilities of this specification for supporting the execution of this kind of services. On the other hand, Web services implementations are distributed across the Internet, generally belonging to 3rd party providers. Therefore, invoking Web services from within the SLEE during the execution flow of converged services requires a mechanism to generate requests and receive responses while managing its data interactions. One possible approach is to implement a JAIN SLEE module for each Web service that wants to be invoked. However this is a complex and time consuming task due to the large number of existing Web Services.

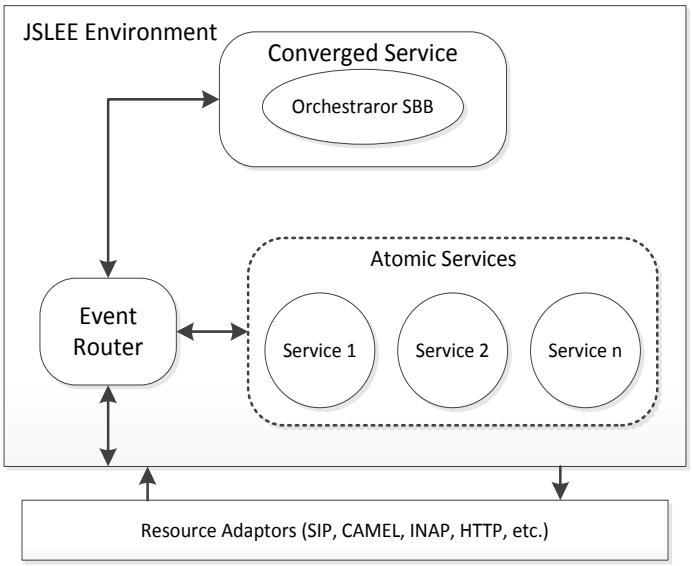

Fig. 1. Converged service general structure.

The proposed integration mechanism defines a generalized and unique component to perform Web service invocations and manage its data interactions. For this, two main modules are defined. The first module is called *Web Service Invocator* and is implemented as a JAIN SLEE application. This module is responsible for representing Web services within the SLEE and for communicating with other Telco services involved in the execution flow of converged services. The second module is called *Dynamic WS Client* which is implemented as a Java Web application, based on the Apache Axis2 framework. This module is responsible for dynamically creating Web services invocation clients from its WSDL description documents. Figure 2 presents a modular scheme of the integration mechanism.

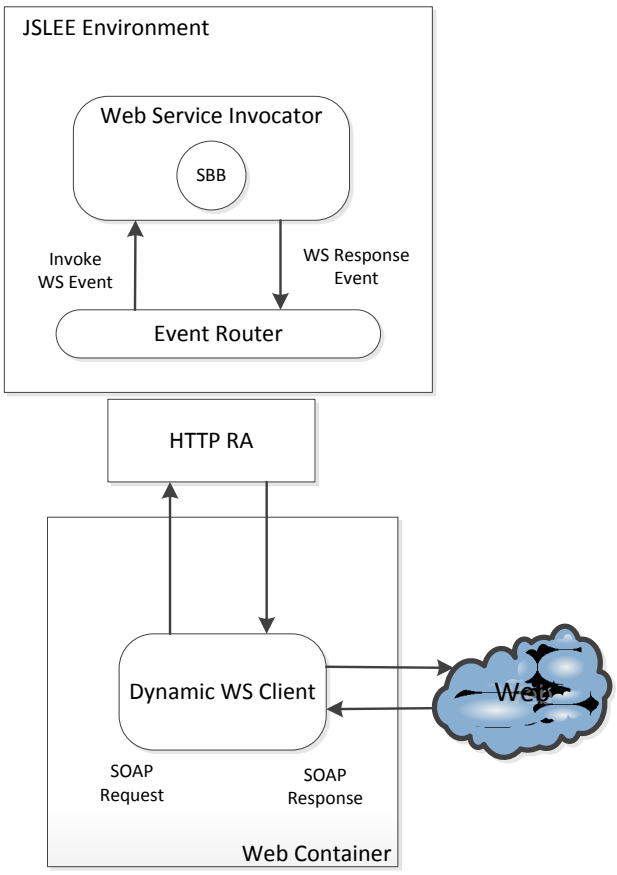

Fig. 2. Modular scheme of the integration mechasnism.

For defining a generalized module for invoking different Web services, it must be considered that despite been described in a standardized manner using WSDL documents; they present a high heterogeneity regarding its data interactions. Input and output parameters of Web services are of different types, being simple data as integers or strings, or more complex such as arrays. The number of inputs and outputs of Web services is another important factor to be considered. For example, a weather Web service may require one input (*location*) to generate a weather forecast, returning two outputs (*temperature and forecast*), while a currency convertor Web service may require three inputs (*source, target and value*) to generate the equivalency between to different currencies, returning one output (*result*). Invoking these Web services requires different methods to create the invocation requests with the corresponding input parameters and receive its responses with the corresponding output data. Below are described the *Web Service Invocation* and *Dynamic WS Client* modules and how is addressed Web service data interaction heterogeneity.

A. Web Service Invocator

As pointed out above, *Web Service Invocation* module has been implemented as a JAIN SLEE application. The interaction with this module is made through an initial event which contains the required data for a Web service invocation, and a response event which contains the invocation results, as shown on Figure 2. To deal with different number of input and output parameters with different data types, it has been defined a data managing interaction mechanism using Java HashMaps. A HashMap is a data structure used to implement associative arrays based on a Key/Value model. The initial event contains a HashMap which includes all the input parameters (name and value) of a Web service for its invocation. Same as the initial event, the response event also contains a HashMap which includes all the output parameters, associating its names and values.

Once an initial event is received by the module, it extracts all the parameters contained in the HashMap and builds a HTTP GET request addressed to the *Dynamic WS Client* module to perform the Web service invocation. Table I shows the parameters that are extracted from the HashMap and included in the HTTP request.

TABLE I. WEB SERVICE INVOCATION PARAMETERS

Parameter	Description
serviceWSDL	This parameter reference the URL of the Web service WSDL. For example: $?serviceWSDL = http://address/wheather?wsdl$
operationName	This parameter contains the name of the specific operation that wants to be invoked form the Web service. For example: $\&operationName = getForecast$
inputs	This parameter contains the name and value of the different inputs required for the Web service for its execution. For example: $\&location = Bogota$

The HTTP response returned by the *Dynamic WS Client*, which contains the output parameters resulting from invoking a Web service, presents such information using a predefined XML structure which will be described later. *Web Service Invocator* module extracts the data contained in the HTTP response through an XML parser and generates a HashMap that presents the outputs in a Name/Value manner which is included in the response event. Table II presents an example of the output HashMap, resulting from the invocation of a weather Web Service.

TABLE II. OUTPUT HASHMAP STRUCTURE EXAMPLE

Key	Value
$temperature$	$(String)"50°F - 77°F"$
$forecast$	$(String)"Cloudy\ with\ 60\%\ chance\ of\ light\ rain"$

HashMaps represent a very convenient mechanism to present the output parameters of a Web service due to their capacity of relating different type of data such as primitive variables, or arrays.

B. Dynamic WS Client

This module receives the HTTP GET request coming from the *Web Service Invocator* module which contains all the parameters needed to invoke a Web service. Using the Apache Axis2 functions, it retrieves the WSDL of the Web service and

maps the input parameter values to create an appropriate client that generates SOAP requests to the Web service. Upon receiving a SOAP response, it takes the output parameter values and generates an XML document which organizes the information. Figure 3 presents an example of a Web service response structured through an XML document.

```
<?xml version="1.0" encoding="UTF-8"?>
<outputs>
    <output>
        <name>temperature</name>
        <value>50°F-77°F</value>
    </output>
    <output>
        <name>forecast</name>
        <value>Cloudy with 60% chance of light rain</valu
    </output>
</outputs>
```

Fig. 3. Example of response structured as an XML document.

The response XML documents are defined by a structure which contains a main tag called *outputs* that encapsulate all the output parameters of a Web service response. A specific parameter is represented by the tag *output* which contains the *name* of the parameter and its corresponding *value*. To represent parameters as arrays, its content is separated in different *output* structures which have the same *name* but possibly a different *value*. Having *output* structures with the same *name* allows the *Web Service Invocator* module to identify arrays and build them to be included in the response HashMap, preventing them to be considered as independent parameters. Once the XML document including all the output parameter has been created, the *Dynamic WS Client* module generates an HTTP 200 OK response including the XML document and addressed to the *Web Service Invocator Module*.

V. Fault Handling and Adaptation

The fault handling and adaptation functions added to the integration mechanism are aimed to detect possible failures during the invocation of Web services and to adapt the mechanism for invocating backup services which are defined at creation time, allowing converged services to carry out its purpose. Based on the service fault taxonomy defined in [18], the fault handling mechanism identifies two different types of faults:

- *Service Provider Faults:* reference possible faults that occur on platforms hosting Web services implementation. For example a Web service being unavailable won't be able to process and respond invocation requests.
- *Communication Faults:* reference possible faults that may occur due to network issues. For example sending to a Web service an invocation request over a network holding heavy traffic may result on a delayed response or no response at all.

The detection of *Service Provider Faults* is made by capturing exceptions generated in the *Dynamic WS Client* module during the invocation of Web services. Once an exception is captured, it sends a HTTP error response to the *Web Service Invocator* module. On the other hand, *Communication Faults* are detected by establishing an execution timer for every invocation request sent by the *Web Service Invocator* module. The maximum waiting time for a Web service response is configured by the converged service developer at the creation stage. The timer is set using the Timer Facility defined by the JAIN SLEE specification. Once a fault is detected, the *Web Service Invocator* module adapts itself to perform the invocation of backup Web services defined at creation time through a list provided by the converged service developer. The adaptation process is formally defined in algorithm 1.

Algorithm 1: Adaptation Process for Web Service Invocation

INPUTS: $WSInvocationData, backupServiceList[\,], timeLimit$

OUTPUTS: $WSOutputData, responseEvent$

BEGIN

$WSOutputData := \emptyset$

$httpReponse := \emptyset$

$httpRequest = createReq(WSInvocationData)$

$timer = setTimer(timeLimit)$

$fireHTTPRequest(httpRequest)$

while $timer$ is active **do**

if $httpResponse \neq \emptyset$ **then**

$deactivateTimer(timer)$

end if

end while

if $httpResponse \neq \emptyset$ **then**

if $httpResponse = 200\ OK$ **then**

$WSOutputData = parseData(httpResponse)$

else do

for ws_i in $backupServiceList[\,]$ **do**

$WSOutputData = invokeWS(ws_i)$

if $WSOutputData \neq \emptyset$ **then**

break

end if

end for

end if

else do

for ws_i in $backupServiceList[\,]$ **do**

$WSOutputData = invokeWS(ws_i)$

if $WSOutputData \neq \emptyset$ **then**

break

end if

end for

end if

$responseEvent = createEvent(WSOutputData)$

$fireEvent(responseEvent)$

END

As stated before, *Web Service Invocator* module initiates a Web service invocation by creating an HTTP request which includes the invocation parameters ($WSInvocationData$). Immediately after sending the request to the *Dynamic WS Client* module, it sets a timer which indicates the maximum waiting time ($timeLimit$) for a response. While waiting, if a response is received the timer is deactivated and the response is analyzed. If the response corresponds to an HTTP 200 OK response, the output parameters are retrieved ($WSOutputData$) and the response event is fired. Otherwise, either if the response did not arrive within the established time or if it corresponds to an HTTP error response, backup Web services (ws_i) retrieved from the list ($backupServiceList[\,]$) provided by the converged service developer are invoked until a successful invocation is achieved. However if no successful invocation is completed an error in the execution flow of the converged service will be produced. This adaptation mechanism helps to reduce the probability of major faults occurring during the execution of converged services.

VI. Implementation and Case Study

The implementation of the *Web Service Invocation* and *Dynamic Web Service* modules was made over the Mobicents JAIN SLEE server and Apache Tomcat Web container respectively. A functional test of the integration mechanism was performed through a converged service called *Twitter Financial Message* which is composed of two Web services (Finance WS and Twitter WS) and two Telco services (Receive IM and Send IM). This service initiates its execution upon receiving a SIP instant message (Receive IM) from the user containing the code name of a NASDAQ stock from which the user wants to know its current value. With the code name is invoked a Web service (Finance WS) that returns the stock current value. Then, the stock information is sent simultaneously to the user as an instant message (Send IM) and as a private message in the Twitter account of the user through a Web service (Twitter WS). Figure 4 shows the converged service diagram and Figure 5 presents the implementation modules needed for its execution.

Through the modules developed, invoking and integrating Web services into a Telco environment requires only for converged service developers to specify data flow between component services, leaving Web service clients' implementation details to the modules.

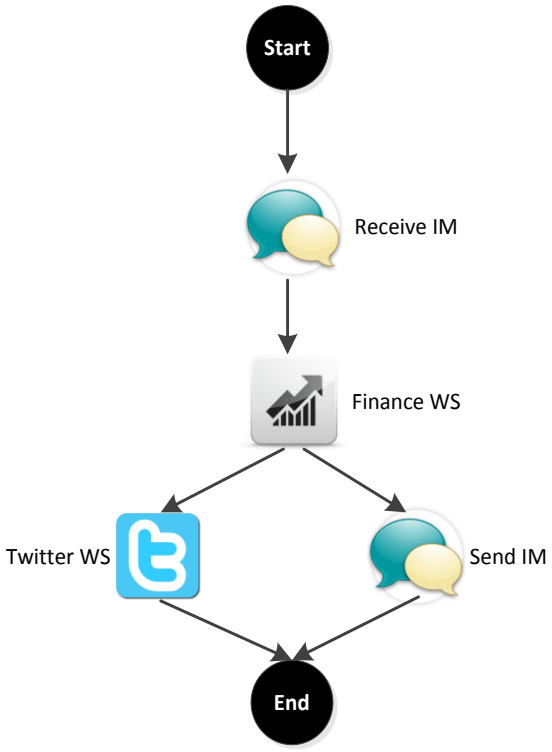

Fig. 4. Twitter Financial Message diagram.

To test the adaptation functions of the integration mechanism was set an undesired scenario where Twitter WS was made unavailable. As a backup Web service, another instance of Twitter WS was configured with a different name.

The test results show that the mechanism adapts to invoke a backup Web service in an average time of 41.9 milliseconds after detecting a fault, indicating a high performance of the adaptation process as seen in Figure 6.

VII. Conclusions and Future Work

This paper presented a dynamic and adaptable approach for integrating Web and Telco services in JAIN SLEE environments. This approach defines a generalized set of modules which allow invoking Web services regardless of their functionality or required data structure. The proposed mechanism facilitates for developers to invoke Web services and integrate them into the execution flow of converged services, by only specifying the data flow required by each one without having to deal with the implementation details of Web service clients. An adaptation mechanism that detects faults occurred during the invocation of Web services is also implemented with very satisfying results. The adaptation process is carried out in a transparent manner from the user perspective, adapting to invoke backup Web services defined at the creation stage, upon the detection of failures with very low execution times. Such process helps reducing the probability of major faults during the execution of converged services.

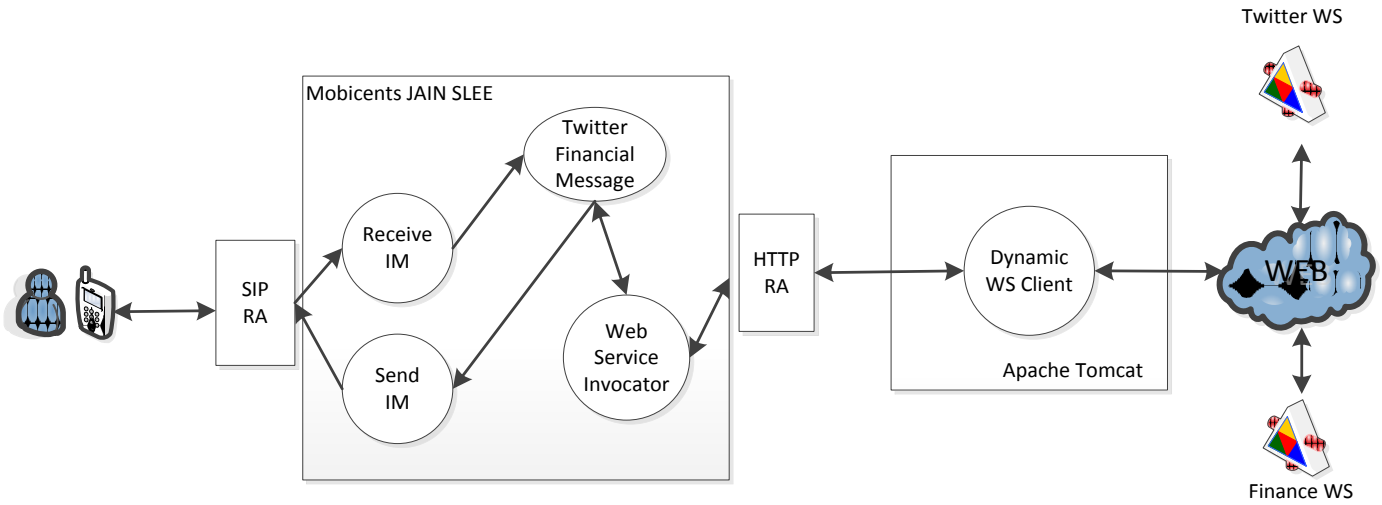

Fig. 5. Implementation of Twitter Financial Message.

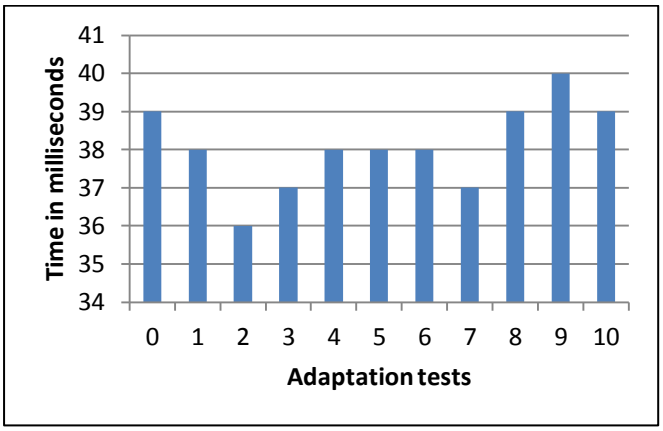

Fig. 6. Performance Adaptation Tests.

This work comprises an important contribution to the TelComp 2.0 project, specifically on the automation of converged service creation and composition. As future work, is proposed to automate the process of retrieving backup services based on functional and semantic analysis, without needing to be specified by the developer at creation time. Other approach that could be addressed is to include REST services into the integration mechanism, considering the large proliferation of this type of services. Another future approach is the design and development of a monitoring mechanism which enables to be aware of the current status of all services and detect undesired behaviors that may compromise the proper operation of converged services.

VIII. ACKNOWLEDGEMENTS

The authors would like to thank University of Cauca and TelComp2.0 project (Code: 1103-521-28338 CT458-2011) for supporting and financing this work and the MSc. students Julián Andrés Rojas and Leandro Ordoñez.

REFERENCES

[1] Jong-Lok Yoon, "Telco 2.0: a new role and business model," *Commun. Mag. IEEE*, vol. 45, no. 1, pp. 10–12, 2007.

[2] JSRs: Java Specification Requests, "JSR 289: SIP Servlet v1.1." 2008.

[3] Ericsson, "Ericsson Converged Service Studio." 2013.

[4] uReach Technologies, "Converged Services Framework." 2013.

[5] JSRs: Java Specification Requests, "JSR 240: JAIN SLEE (JSLEE) v1.1." 2008.

[6] D. Booth, H. Haas, F. McCabe, E. Newcomer, M. Champion, C. Ferris, and D. Orchard, "Web Services Architecture, W3C Working Group Note 11." W3C Technical Reports and Publications, 2004.

[7] K. Rezabeigi, A. Vafei, and N. Movahhedinia, "A Web Services based Architecture for NGN Services Delivery," *World Acad. Sci. Egineering Technol.*, vol. 43, 2008.

[8] Apache Software Foundation, "Apache Axis2 User's Guide." 2012.

[9] P. Falcarin and C. Venezia, "Communication Web Services and JAIN-SLEE Integration Challenges," *Int. J. Web Serv. Res.*, vol. 5, no. 4, p. 5978, 2008.

[10] P. Falcarin and L. Walter, "An Aspect-Oriented Approach for Dynamic Monitoring of a Service Logic Excecution Environment," *IEC Annu. Rev. Commun.*, vol. 59, pp. 237–242, 2006.

[11] Grupo de Ingeniería Telemática, "TelComp2.0 Project Website," 2013. [Online]. Available: http://190.90.112.7:8080/TelComp-SCE/.

[12] L. Zygmunt, "Parlay/OSA - a New Way to Create Wireless Services." The Parlay Group, 2003.

[13] P. Baglietto, M. Maresca, M. Stecca, A. Manzalini, R. Minerva, and C. Moiso, "Analysis of design patterns for composite telco services," in *Intelligence in Next Generation Networks (ICIN), 2010 14th International Conference on*, 2010, pp. 1–6.

[14] E. Bertin and N. Crespi, "Describing Next Generation Communication Services: A Usage Perspective," *Lect. Notes Comput. Sci.*, vol. 5377, pp. 86–97, 2008.

[15] A. Hasanović, N. Suljanović, A. Mujčić, and R. Sernec, "Dynamic Languages Integration Path for Telecom Applications," in *Digital Telecommunications, 2009. ICDT '09. Fourth International Conference on*, 2009, pp. 133–137.

[16] C. Venezia and P. Falcarin, "Communication Web Services Composition and Integration," in *Web Services, 2006. ICWS '06. International Conference on*, 2006, pp. 523–530.

[17] J. Niemoller, E. Freiter, K. Vandikas, R. Quinet, R. Levenshteyn, and I. Fikouras, "Composition in Converged Service Networks: Requirements and Solutions," in *International Workshop on Business Systems Management and Engineering*, 2010.

[18] I.-Y. Chen, G.-K. Ni, C.-H. Kuo, and C.-Y. Lin, "A BPEL-Based Fault-Handling Architecture for Telecom Operation Support Systems.," *JACIII*, vol. 14, no. 5, pp. 523–530, 2010.