UNIVERSIDAD DISTRITAL
FRANCISCO JOSÉ DE CALDAS

Tecnura

http://revistas.udistrital.edu.co/ojs/index.php/Tecnura/issue/view/650

DOI: http://dx.doi.org/10.14483/udistrital.jour.tecnura.2015.4.a01

INVESTIGACIÓN

Automatic orchestration of converged services on JSLEE environment

Orquestación automática de servicios convergentes en entornos JSLEE

Jesus David Ramirez[1], Juan Carlos Corrales[2]

Fecha de recepción: 8 de septiembre de 2014 **Fecha de aceptación:** 24 de agosto de 2015

Como citar: Ramirez, J. D., & Corrales, J. C. (2015). Automatic orchestration of converged services on JSLEE environment. Revista Tecnura, 19(46), 15-26. doi:10.14483/udistrital.jour.tecnura.2015.4.a01

Abstract

A widely adopted solution in order to obtain a low Time to Market by a segment of Telecommunication operators is the use of the concept of service composition because their philosophy is to reuse software components previously implemented. The composition has two phases, the synthesis and orchestration, understanding the second one as a challenge to converged services because it requires extensive technical knowledge and experience. This article proposes a mechanism based on graphs and Petri Nets to automate the orchestration of services in converged JSLEE environments, which operates at design time in order to not affect performance in the execution of composite services. The results demonstrate that the mechanism can transform an abstract process with 20 component services in an executable JSLEE service, without exceeding 500 ms.

Keywords: Automatic Orchestration, JAIN SLEE, Service Composition.

Resumen

Una solución ampliamente adoptada para obtener un bajo Time to Market por parte de los operadores de Telecomunicaciones, es la utilización del concepto de composición de servicios ya que su filosofía es reutilizar componentes software previamente implementados. La composición tiene dos fases, la síntesis y la orquestación, siendo la segunda un reto en los servicios convergentes debido a que requiere amplio conocimiento técnico y experiencia. En este artículo se propone un mecanismo basado en Grafos y redes de Petri para automatizar la orquestación de servicios sobre entornos convergentes JSLEE, el cual funciona en tiempo de diseño con el propósito de no afectar el rendimiento en la ejecución del servicio compuesto. Los resultados demuestran que el mecanismo puede transformar un proceso abstracto con 20 servicios componentes, en un servicio ejecutable JSLEE, sin superar los 500 ms.

Palabras clave: Composición de servicios, JAIN SLEE, Orquestación Automática.

[1] Electronic and Telecommunications Engineer, master's student in Telematics Engineering, University of Cauca. Popayán, Colombia. Contact: jdramirez@unicauca.edu.co

[2] Electronic and Telecommunication Engineer, master's student of Telematics Engineering, PhD in Computer Science. Professor at the University of Cauca. Leader of the Telematics Engineering Group. Popayán, Colombia. Contact: jcorral@unicauca.edu.co

INTRODUCTION

Currently, the advances in Information and Communication Technology (ICT) have generated among users a growing demand for new and more personalized services with greater functionality (Martinez, et al., 2009). To meet this demand, telecommunication operators (hereafter called "operators") have followed new trends that incorporate Web technologies to their business models. One such trend is known as Telco 2.0 (Yoon, 2007), which aims to integrate the concepts, technologies and services from the Web domain with traditional services of telecommunications domain (Telco services). Thus, operators may expand their portfolio of services offering converged services, which are defined as the coordination of a set of Web and Telco services that are implemented using different types of networks and protocols (ITU-T, 2006) (Chudnovskyy, Weinhold, Gebhardt, & Gaedke, 2011).

However, there are two factors that operators should consider to develop converged services. First, the high-performance that converged services (for example Telco services) require to be provided in real time and with a high degree of availability (Johnson, Kogan, Levy, Saheban, & Tarapore, 2004); and second, low Time to Market, in other words, the time it takes for a service to be developed, since it is planned until it is launched and offered for sale (Haran, 2011).

Accordingly, one of the possible solutions to maintain high performance is the acquisition of Service Delivery Platforms (SDP) that allows the creation of new converged services with quality that an operator requires. These platforms are based on technologies such as SIP Servlets, JAIN SLEE (*Java APIs for Integrated Networks Service Logic Execution Environment*), SCIM (Manager of interaction between service capabilities), among others (Bond, Cheung, & Levenshteyn, 2009). Although, currently the specification JAIN SLEE (hereafter called "JSLEE") (JCP, 2008) is the most commonly used for this purpose. JSLEE proposes a high performance environment characterized by high availability, low latency, which is event-driven, component-based, asynchronous iterations, and also allows abstraction over multiple network protocols like HTTP, SIP, SS7, among others.

Moreover, a widely adopted solution for obtaining a low Time to Market, is service composition (Dinsing, Eriksson, Fikouras, Gronowski, & Levenshteyn, 2007), which is defined as: the coordination of multiple services in order to ensure their interaction for meeting a common goal (Goncalves da Silva, 2011). The composition can reduce the time to market of an operator or a service provider on the Internet because its philosophy is based on the reuse of software components previously implemented, and the definition of the interactions among them (Dan, Johnson, & Carrato, 2008).

To properly implement service composition, it is necessary to consider its structure and phases. The structure contains the following main elements: component services, control flow and data flow. The control flow defines the order in which the component services are performed, considering invocations, divisions, convergence and conditional in the flow. The data flow defines the way the data is transferred among the component services, including transformations, persistent variables and external data (Trcka, Aalst, & Sidorova, 2008). Moreover, the phases of the composition are two, Synthesis and Orchestration (Küster, Stern, & König-Ries, 2005) (Berardi, Giacomo, Sapienza, & Bozen, 2005).

In the synthesis, it is generated a plan that combines the functionality of multiple component services in order to define the behavior of composite service (Küster, Stern, & König-Ries, 2005). The result of the synthesis can be described as an abstract process, because it only defines the partial flow control and does not define the data flow; therefore, it is not an executable service. To implement the synthesis, it is possible to use automated techniques usually employed in the Web domain such as artificial intelligence planners or semantic-based tools, such as the authors proposed in (Küster, Stern, & König-Ries, 2005) (Berardi,

Giacomo, Sapienza, & Bozen, 2005) (Rao & Su, 2005). This is possible because the abstract processes are not executable and therefore they are independent of the environment where the service is performed.

Furthermore, the orchestration refers to the process of defining in detail the data flow and control flow among the composite services using a standard language that allows the generation of an executable service (Goncalves da Silva, 2011) (Berardi, Giacomo, Sapienza, & Bozen, 2005). To perform the orchestration of converged services, it is important to consider the direct dependence between the service and the environment in which it is executed, which is why this phase has a high level of complexity, since for environments such as JSLEE, it is required technical level knowledge on different protocols, languages and format of the service logic.

In this paper, we propose a mechanism that automates the orchestration of services in converged JSLEE environments, which operates at design time to reduce the time consumption and increase the performance in the service execution. This mechanism facilitates the orchestration because it automatically generates an executable service in JSLEE environments from an abstract process described in graphs.

RELATED WORKS

Given the complexity of the orchestration of converged services, some works like (Zhu, Zhang, Cheng, Wu, & Chen, 2011) (Femminella, Maccherani, & Reali, 2011) seek to facilitate the orchestration adapting techniques from the Web domain, in which recognized languages such as BPEL (Business Process Execution Language) or JPDL (Java Process Description Language) are used to define the orchestration, integrating BPEL or JBPM (Java Business Process Manager) engines to converged environment in order to execute the logic of composite service. However, this involves developing an intermediate module that adapts the BPEL o JPDL orchestration to the language and format of the converged environment; in addition, this adaptation is performed when the service is executed, so the performance is not considered in the execution (Drewniok, Maresca, Rego, Sienel, & Stecca, 2009).

In (Lehmann, et al., 2009) the authors also propose convergent services orchestration using BPEL, but unlike previous work, the adaptation module is used at design time, so that it does not affect performance; however, the orchestration must be performed by means of manual techniques, which requires advanced knowledge on different protocols and asynchronous events of the telecommunications domain, for which the BPEL language is not very suitable (Bond, Cheung, & Levenshteyn, 2009). Based on the previous state of the art, it is possible to state that the way to orchestrate services in converged environments is not yet clear.

METHODOLOGY

To define the mechanism of automatic orchestration we first select a language for describing the abstract processes and the executable service, for this task, three formal models are compared taking as selection criteria the model that best fits to abstract processes and executable service characteristics. Subsequently, we designed the architecture mechanism, where the implementation of each module let us pass through the abstract process, the orchestration, the source code, the service deployment and finally to the service execution. For testing the mechanism, we develop a prototype that allowed taking time measurements at each module and analyze the performance of the mechanism. In the following sections the design and testing of the mechanism are described in detail. In the third section, we select two formal models that may facilitate the description of the abstract processes and the executable service. In section four, we present the overall architecture of the automatic orchestration proposal, describing each of its modules. The fifth section is focused on the evaluation and analysis of

performance results of the automatic orchestration mechanism using a test tool to measure the execution time. Finally, in the fifth section, we present the conclusions and future work.

FORMAL MODELS

As mentioned in the previous section, the automatic orchestration mechanism operates from an abstract process that describes the converged service. Some of the related work, describe convergent service with languages such as abstract BPEL or JPDL (jBPM Process Definition Language), though these languages are complex to process since they are based on XML (eXtensible Markup Language) and, additionally require an IDE like Netbeans or Eclipse to describe them. Therefore, this paper is focused on using a formal model whose main objective is to represent the knowledge of the converged service in order to facilitate inference of information from the language it is described.

Below we present a comparative analysis among the most common formal models in order to select the one that best fits the characteristics of both the abstract processes (synthesis) and the resulting executable service of the orchestration mechanism.

Types of Models

***Graphs*:** formally a graph is a pair $G = (N, E)$ where N is a nonempty finite set of elements called nodes such that $N = \{n_1,\ldots, n_m\}$, and E is a set of pairs (n_i, n_k) unordered of distinct elements of N called edges, such that $E \subset N \times N$, where N and E are different and $N \cap E = \varnothing$.

From the viewpoint of composite services, graphs allow the modeling of the structure of a composite service at a high level of abstraction, representing relationships among its components such as the data flow and control flow. Although, the expressiveness of graphs is limited, it is possible to add some expressive power by annotating its nodes and edges, assigning attributes on the nodes of the graph that can register the status of execution of composite service.

***Finite State Machines (FSM)*:** are conceived as an abstract machine that can be located in any of a finite number of states at a given time, called current state, and can change from one state to another by the occurrence of an event or compliance with a specific condition, called transition. Formally, an FSM is a 5-tuple of the form $(\Sigma, S, S_0, \delta, F)$ where: Σ is a set of inputs, S is a set of states, S_0 is the initial state, δ is the transition function, and F is the set of final states.

From the point of view of the composite services, finite state machines allow the specification of the execution flow of a service; however, it is difficult to describe its control flow because, at a given point, a machine can only present a single statement limiting the performance of a composite service, which may be concurrent execution flows.

***Petri Nets (PN)*:** are a formal and graphical language for modeling systems and processes. Formally, a Petri Net is defined as a 5-tuple of the form: $PN = (P, T, A, W, M_0)$, where: $P = \{p_1, p_2, \ldots, p_m\}$ is a finite set of places or states, $T = \{t_1, t_2, \ldots, t_n\}$ is a finite set of transitions, $A \subseteq (P \times T) \cup (T \times P)$ is the set of arcs connecting places with transitions, W: $A \rightarrow \{1,2,3, \ldots\}$ is a weight function, M_0: $P \rightarrow \{0,1,2, \ldots\}$ is the initial marking having the places, which represents the state at any given time. Moreover, it holds that $P \cap T = \varnothing$ and $T \cap P = \varnothing$. The marking of the PN changes when a transition is executed and this means the markers are removed from input places and inserted into the output places of a transition (Grigori, Corrales, & Bouzeghoub, 2008).

From the point of view of the composition of services, PN allows the specification and description of both structure and behavior of this type of services. The expressiveness associated with this formal model is higher than that offered by the graph and the FSM, however, it implies high computational complexity in the methods of analysis for a system modeled with this model, which is a critical problem for processes involving the discovery,

composition and reconfiguration of composite services at execution time.

Selection of Models

In order to select the most appropriate model for both the abstract process (synthesis) and the executable service (orchestration), we have chosen the characteristics of each phase of the composition (synthesis and orchestration) as the selection criteria.

Below we present the characteristics of abstract processes (Martens, 2005).

a) *Abstract services:* each service component is a word that describes its functionality, but not a link to the service that implements it.
b) *Partial control flow*: defines the execution order, but abstract details such invocations.
c) *Abstract Control patterns*: each pattern has an element that describes it, but no structure.
d) *Data dependencies*: set data dependencies between inputs and outputs of services.
e) *No data flow:* there is not definition of variables, transformations or external data in the service.
f) *Centralized:* the control flow and data flow is implemented on a single central server.
g) *No executable*: it does not execute directly on the server.

In table 1, the comparison between formal models according to the characteristics of the abstract processes is shown. Then only the characteristics with a marked difference between models are discussed: b) it is conditioned on FSM and PN as these should describe the complete control flow, and only in some cases they can abstract elements; c) it is not supported on FSM and PN, because in these, patterns require some structure to achieve a given behavior, and only some patterns could be modeled with a single abstract element; d) it is conditioned on the models, in graphs it would require additional edges and labeling, in FSM and PN, the dependencies could be marked but only among adjacent services; e) it is not supported on FSM and PN as these should describe the complete data flow through the service; finally, g) it is not supported on FSM and PN as there are tools to simulate and implement languages for this type of models.

Table 1. Comparison of models for the synthesis

Abstract Process Features	Graphs	FSM	PN
Abstract services	2	2	2
Partial control flow	2	1	1
Abstract control patterns	2	0	0
Data dependencies	1	1	1
No data flow	2	0	0
Centralized	2	2	2
No executable	2	0	0
TOTAL	13	6	6
Supported (2), Not Supported (0) and Conditional (1)			

Source: own work.

In conclusion, the formal model that best fits the characteristics of an abstract process are graphs, since this model obtained a score of 13.

Moreover, below we present the characteristics of the executable service (orchestration), (Martens, 2005).

a) *Concrete services:* each service component has a link to the service that implements it.
b) *Total control flow*: it defines the execution order of all the elements required for execution.
c) *Concrete control patterns:* each pattern has a structure describing its implementation.
d) *Data flow*: it establishes the data passing among each service including variables and transformations.
e) *Additional logic:* it implements additional logic of service, such as persistence, timers or communication with the client application.
f) *Standard language*: it is described in some standard language that is executable by an engine.
g) *Centralized:* the control and data flow are implemented on a single central server.
h) *No executable:* it is executed directly on the server.

In table 2, a comparison is made among the models according to the characteristics of the executable service (orchestration) (Martens, 2005). Below we discuss only the characteristics that show a difference among models: b) it is conditioned on graphs because patterns as external triggers or conditions of service execution would be difficult to model, and we would have to use many labels for this information; c) it is not supported on FSM since having a single state, it would not support parallel flow patterns, either in graphs since it could only label nodes but the structure would not represent the behavior patterns; d) it is not supported on graphs because it cannot model the global variables and data transformations through the service; e) it is not supported on graphs as these only describe the relationship among services; in FSM, it is conditional because it could support only some of these functions; f) it is not supported on graphs because they do not have a standard language, in contrast to FSM there is eCharts and for PN there is PNML (Petri Net Modeling Language); finally, h) it is not supported on graphs since there are no standard tools that simulate and run directly.

Table 2. Comparison of models for the orchestration

Executable Services Features	Graphs	FSM	PN
Concrete Services	2	2	2
Total flow control	1	2	2
Concrete control patterns	0	0	2
Data flow	0	2	2
Additional logic	0	1	2
Standard Language	0	2	2
Centralized	2	2	2
Executable	0	2	2
TOTAL	5	13	16
Supported (2), Not supported (0) and Conditional (1)			

Source: own work.

In conclusion, the formal model that best fits the characteristics of executable services in converged environments is Petri Nets. However, there is a large amount of subtypes of Petri Nets, whereby we chose the Colored Petri Nets (CPN), because it defines a set of special tokens which may contain different information, implying that they may be processed according to the data content.

ARCHITECTURE FOR AUTOMATIC ORCHESTRATION OF CONVERGED SERVICES

In figure 1, we present the architecture of the implemented mechanism for automatic orchestration of converged services with all its modules. Below we describe each module in detail.

TelComp2.0 Composition Environment

In order to generate the abstract processes that feed the orchestration mechanism we use the converged service composition environment developed in the project TelComp 2.0 (Corrales, 2010) as it conforms exactly to the requirements. Specifically, the above environment is a Web application that generates abstract processes represented by graphs, and also uses a repository of Web and Telco services.

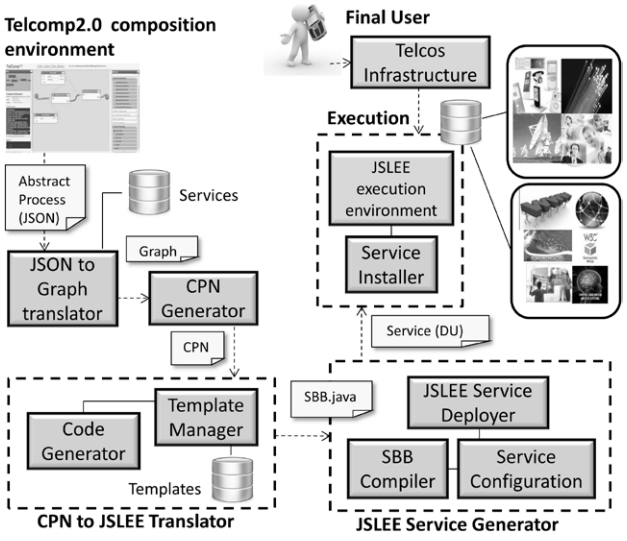

Figure 1. Architecture of Mechanism

Source: own work.

In figure 2, we present an example of a service created with the environment; the edges are indicating the control and data flow defined for the service; the nodes represent the component services. The abstract process resulting from the composition is delivered in JSON (JavaScript Object Notation) format, which structure consists of three parts: nodes described with a name and ID, the control flow edges that define the source and destination nodes through the IDs, and data dependencies among nodes using the names of the variables. For example, for figure 2, in the first part, the components User_data, GetDataTelcoService and MediaCallTelcoService have these names and IDs 1, 2 and 3, respectively, while in the second part, there is a control edge between components 2 and 3, in the third part, there are three data edges, from 1 to 2, from 1 to 3, and from 2 to 3, which are control edges and data edges at the same time; for example, the data edge from 1 to 3 relates the variables Message (output 1) and Text (input 3).

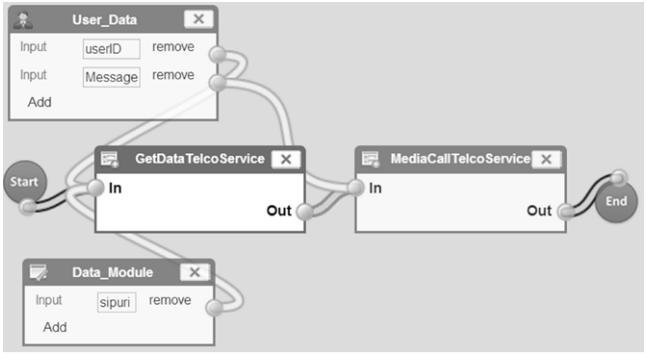

Figure 2. Example of TelComp2.0 composition environment

Source: own work.

JSON to Graph Translator

The abstract process generated with the composition environment corresponds to a graph described in JSON, however, this graph does not contain all the information necessary to start the orchestration. Therefore, this module accesses the repository of services to supplement the information of each service with the operations, input and output parameters, data types, and physical location. In order to represent all the information of the abstract process, we propose the graph model of figure 3, which defines the classes on the left side: Connector representing all patterns, Service that could be Telco or Web with its physical location, Operation indicating the tasks or actions of service; and on the right side, the classes: Control Edge linking two connectors, Data Element defining the data and its type, Association Data linking data element of two services, Data Edge gathering several data associations. It is important to highlight that the graph and its information are implemented in the Java language.

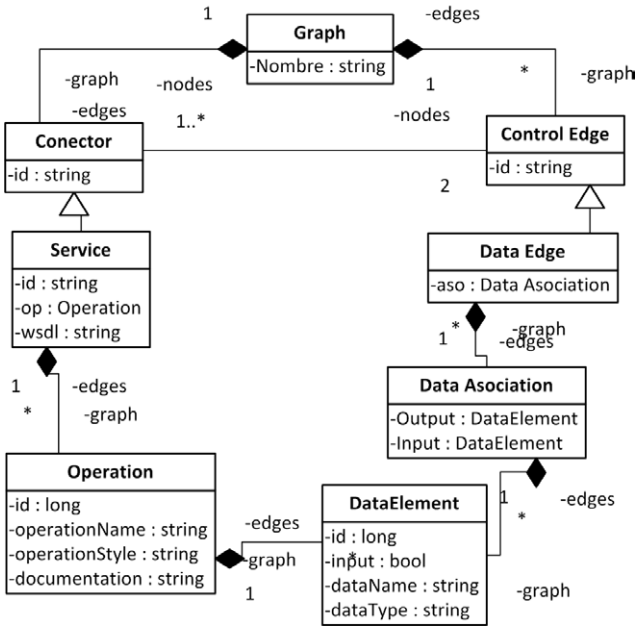

Figure 3. Graph model proposed for abstract processes

Source: own work.

CPN Generator

The translation from graph to CPN is based on (Staines, 2011) where it is established that for each service graph node, a model in CPN is defined by

an input place (*waiting invocation*), an intermediate transition (*executing service logic*) and an output place (*sending reply*). This model has a high level of abstraction for the operation of the service because its internal behavior can be described by a number of states and transitions, which complicate the analysis of the structure of the service. Similarly, the patterns used in the graph, such as AND-SPLIT or OR-JOIN, are modeled in CPN following the structure defined in (Van der Aalst & Hofstede, 2012). It is clarified that the control flow patterns of converged services that are used in the orchestration mechanism were specifically mentioned in (Benavides, Enriquez, Ramirez, Figueroa, & Corrales, 2012); therefore, it is strongly advised that the reader reviews this work thoroughly.

Figure 4 shows an example of the conversion performed (graph on the top and CPN on the bottom), where the graph nodes are first analyzed in order to define them as service or pattern and create their respective model. Secondly, the control flow is analyzed (edges among START, Service1, Service2 and END nodes) to create the arcs of the CPN. Thirdly, the data flow is analyzed (edges among User, Service1 and Service2 nodes) to define the input data of each service through the Token of the CPN. Finally, the information acquired is organized and the CPN is created.

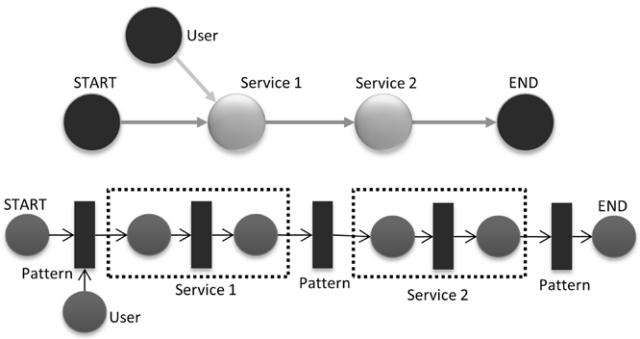

Figure 4. Equivalence among services in Petri Net graphs

Source: own work.

CPN to JSLEE Translator

This module generates a Java file (SBB) from the created CPN corresponding to the logic of orchestrated service. The main sub-module that allows this task is the code generator, which uses templates as a basic instrument; these are generic code fragments where you can set different variables such as the input parameters of a service, the names of the events, the global variables of the service, among others.

The sequences of steps that are performed within this module are the following:

- Firstly, the event information is extracted from the transitions; secondly, the control flow is extracted with the analysis of arcs; thirdly, the places are analyzed to determine the states of the program; and finally, we extract the input and output data of service from the tokens.
- The identified elements are associated with Java Templates through the Templates manager, which extracts them from the templates repository.
- The code generator sets variables and assembles the Java Templates in a specific order on a file SBB.java and its descriptor SBB.xml.

Generator of JSLEE Service

Once generated the SBB.java, the Compile module of SBB is responsible for generating the SBB.class. Subsequently, with the Service Configuration module, the necessary files of configuration are generated for the deployable unit of JSLEE services. These are described in figure 5.

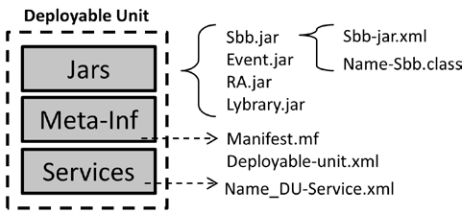

Figure 5. Deployable unit of JSLEE services

Source: own work.

- The file *"jars"* contains the .jar interacting in the operation of the service, such as: sbb, event, RA and Library. The main element is the sbb, in this *.jar* we can find the *.class* of service and the descriptor xml.
- The file *"META-INF"* containing the descriptor of DU (deployable unit.xml) and the meta-data file (MANIFEST.MF)
- The file "services" contains the descriptor of converged service.

After generating the deployable unit of convergent JSLEE service, it is automatically deployed to the server via the display module. Finally, the service is being activated awaiting execution in the JSLEE environment.

EVALUATION AND RESULTS

In this section, we present a preliminary test consisting in a time performance evaluation of a defined mechanism, which aims to measure the time used for execution. Thus, we have performed the scalability test where we increased the complexity of the submitted request (Kankanamge, 2012), increasing the number of service components in the abstract process.

Considering the advantages of Service Oriented Architecture (SOA), the mechanism of orchestration was packed as a Web service; therefore, we used the open source tool SOAPUI (SmartBEAR, 2012) for the rapid creation of advanced test on web services performance (Hussain, Wang, Toure, & Diop, 2013). The test was performed with two computer systems connected, one with SOAPUI and another with the mechanism, which operated on a Linux Ubuntu 12.04 system with core i5 processor and 4GB of RAM.

The test was executed under the following parameters: a sequence that represents a single abstract process; seven executions of the same process to calculate the geometric mean and get a more precise value; three seconds among executions to avoid overlaps among queries; and n (number of component services) varying in pairs to twenty.

As expressed in figure 6, the total time of automatic orchestration mechanism has a linear growth, but does not exceed 500 ms for process less than 20 services, demonstrating an efficient use of computational resources of the mechanism. This time is mainly performed by the translation module from JSON to graphs, because in this module, we access to the service repository via Web services, which requires a larger amount of computational and network resources.

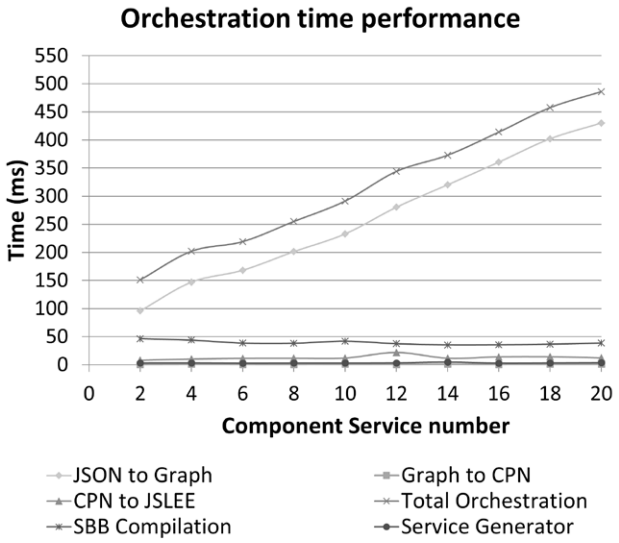

Figure 6. Time performance of mechanism.

Source: own work.

However, this could be improved by creating a direct connection to the repository database without using web services. Moreover, other modules show excellent performance as they keep almost constant in the face of the increased complexity.

CONCLUSIONS AND FUTURE WORK

In this paper, we proposed a mechanism to automate the service orchestration in converged JSLEE environments. This mechanism is based on graphs and CPN to formally represent the synthesis and orchestration respectively, this greatly facilitated the modeling of each phase and the implementation

of algorithms for analysis and translation from one phase to another.

In comparison with the first related works that adapt the orchestration to JSLEE at execution time, this mechanism operates at design time, therefore it does not affect performance in the execution of the service. The evaluation of performance showed that although the processing time increases linearly with the number of component services, for a value of 20 services, it does not surpass the 500 ms, in this respect, the mechanism even at design time is quite efficient.

Moreover, compared to the last related work that also operates at design time, this mechanism provides benefits to operators, because it allows creating the abstract process with a Web composition environment, avoiding the handling of BPEL designers. Furthermore, this mechanism is based on execution flow patterns that are specific for Telco services, instead of only Web domain. Therefore this mechanism allows generating convergent services in a very short time and in an easy way from an abstract process, sparing them the difficult task of orchestrating services on JSLEE environments, which involves extensive technical knowledge and experience.

Future work could integrate this mechanism with proposals aimed to automate the synthesis phase, so that all would be automatic composition, allowing, for example, by means of a request in natural language, a converged service environment is generated in JSLEE environments.

ACKNOWLEDGEMENTS

The authors would like to express their gratitude to the Young Researchers and Innovators Program of Colciencias 2011 and to the University of Cauca for funding this research (project code 3458). We also thank the project TelComp2.0 (Project Code 1103-521-28338 CT458-2011) to facilitate the use of the graphical composition environment and the services repository used on this project.

REFERENCES

Benavides, A., Enriquez, G., Ramirez, J. D., Figueroa, C., & Corrales, J. C. (2012). Control-Flow Patterns in Converged Services. *The Fourth International Conferences on Advanced Service Computing*, (pp. 37-42). Nice.

Berardi, D., Giacomo, G., Sapienza, L., & Bozen, B. (2005). Automatic Composition of Process-based Web Services : a Challenge. *Analysis, 531*, 1-3.

Bond, G., Cheung, E., & Levenshteyn, R. (2009). Unified Telecom and Web Services Composition : Problem Definition and Future Directions. *IPTCOMM'09* (pág. 13). ACM.

Chudnovskyy, O., Weinhold, F., Gebhardt, H., & Gaedke, M. (2011). Integration of Telco Services into Enterprise Mashup Applications. *Current Trends in Web Engineering* (págs. 37-48). Springer Berlin Heidelberg.

Corrales, J. C. (2010). *Proyecto Telcomp2.0 (Financiado mediante contrato RC 458-2011 celebrado entre la Fiduciaria Bogotá, la Universidad del Cauca y COLCIENCIAS)*. Popayan.

Dan, A., Johnson, R., & Carrato, T. (2008). SOA service reuse by design. *Proceedings of the 2nd international workshop on Systems development in SOA environments SDSOA 08*, (pp. 25-28).

Dinsing, T., Eriksson, G., Fikouras, I., Gronowski, K., & Levenshteyn, R. (2007). Service composition in IMS using Java EE SIP servlet containers. *Ericsson Review, 3*, 92-96.

Drewniok, M., Maresca, M., Rego, S., Sienel, J., & Stecca, M. (2009). Experiments and performance evaluation of Event Driven Mashups. *IEEE Symposium on Computers and Communications* (pp. 19-22). Sousse: IEEE.

Femminella, M., Maccherani, E., & Reali, G. (2011). Workflow Engine Integration in JSLEE AS. *IEEE Communications Letters, 15*(12), 1405-1407.

Goncalves da Silva, M. E. (2011). User-centric Service Composition–Towards Personalised Service Composition and Delivery. *PhD Thesis*, 233. Unversity of Twente.

Grigori, D., Corrales, J. C., & Bouzeghoub, M. (2008). Behavioral matchmaking for service retrieval: Application to conversation protocols. *Information Systems Journal, 33*(7-8), 681-698.

Haran, H. (2011). *Time to Market Research: Highlights and Key Findings*. AMDOCS-Customer Experience Systems Innovation.

Hussain, S., Wang, Z., Toure, I., & Diop, A. (2013). *Web Service Testing Tools: A Comparative Study*. arXiv preprint arXiv:1306.4063.

ITU-T. (2006). *ITU-T Recommendation Y.2013–Converged services framework funtional requirements and architecture*. TELECOMMUNICATION STANDARDIZATION SECTOR OF ITU.

JCP. (2008). *JAIN SLEE v 1.1*. Retrieved Decembre 5, 2012, from JSR 240: http://jcp.org/en/jsr/detail?id=240

Johnson, C., Kogan, Y., Levy, Y., Saheban, F., & Tarapore, P. (2004). VoIP reliability: a service provider's perspective. *IEEE Communications Magazine*, 48-54.

Kankanamge, C. (2012). *Web Services Testing with SoapUI*. Packt Publishing Ltd.

Küster, U., Stern, M., & König-Ries, B. (2005). A classification of issues and approaches in automatic service composition. *Intl Workshop WESC, 5*, 25-35.

Lehmann, A., Eichelmann, T., Trick, U., Lasch, R., Ricks, B., & Toenjes, R. (2009). TeamCom: A Service Creation Platform for Next Generation Networks. *Fourth International Conference on Internet and Web Applications and Services* (pp. 12-17). Venice/Mestre: IEEE.

Martens, A. (2005). Consistency between executable and abstract processes. *IEEE International Conference on eTechnology eCommerce and eService* (pp. 60-67). IEEE.

Martinez, A., Zorita, C. B., Martin, A. M., Morchon, C. G., Calavia, L., Perez, J. A., & Caetano, J. (2009). TelecomI+D03: New Business Models: User Generated Services. *IEEE Latin America Transactions, 7*(3), 395-399.

Rao, J., & Su, X. (2005). A Survey of Automated Web Service Composition Methods. *Semantic Web Services and Web Process Composition, 33*(2), 43-54.

SmartBEAR. (2012). *SOAPUI*. Recuperado el 5 de December de 2012, de http://www.soapui.org

Staines, A. S. (2011). Rewriting Petri Nets as Directed Graphs. *International Journal of Computers, 5*(2), 289-297.

Trcka, N., Aalst, W., & Sidorova, N. (2008). *Analyzing Control-Flow and Data-Flow in Workflow Processes in a Unified Way*. Computer Science Report No. 08-31, Technische Universiteit Eindhoven, Eindhoven.

Van der Aalst, W., & Hofstede, A. (2012). *Workflow Patterns*. Recuperado el 5 de December de 2012, de http://www.workflowpatterns.com/documentation/

Yoon, J.-L. (2007). Telco 2.0: a new role and business model. *IEEE Communications Magazine, 45*(1), 10-12.

Zhu, D., Zhang, Y., Cheng, B., Wu, B., & Chen, J. (2011). HSCEE : A Highly Flexible Environment for Hybrid Service Creation and Execution in Converged Networks. *Journal of Convergence Information Technology, 6*(3), 264-276.