Automatic Code Instrumentation for Converged Service Monitoring and Fault Detection

Diego Adrada, Esteban Salazar, Julián Rojas, Juan Carlos Corrales
Telematics Engineering Department
University of Cauca
Popayán, Colombia
Email: {diegoadrada, edsalazar, jrojas, jcorral}@unicauca.edu.co

Abstract—The notion of converged services is based on the integration of traditional telecommunications features and Web 2.0 services. Today this concept is given special attention by the telecommunications service providers as a mechanism that allows them to expand their service portfolio and have greater market dynamism. However, these services present new challenges from its management point of view, making necessary runtime monitoring mechanisms that permit adequate fault detection and thus ensure proper operation. In this paper, we present a runtime monitoring scheme that uses automatic code instrumentation techniques to provide converged services with functionalities that facilitate its fault-handling and operational management. We also provide an implementation of the monitoring scheme through a sample converged service which demonstrates its performance and applicability to this type of services.

Keywords— converged service; JAIN SLEE; Javassist; monitoring; fault detection

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 the traditional telecommunications features, 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 type of services are known as converged services due to its integration of functionalities from the telecommunications domain with IT services from the Web domain.

For the development and execution of new converged services is necessary to have robust platforms that allow to carry out the integration of different technologies and communications protocols characteristic of services from both, the telecommunications and Web domain. With this purpose, from the academy and the industry, different approaches have been proposed such as the SIP Servlets [2] specification, the Ericsson Converged Service Studio [3], the Alcatel-Lucent uReach CSF (Converged Services Framework) [4], among others. However, an open-source alternative that stands out is the JAIN SLEE [5] [6] specification, which proposes a standard and robust environment for the implementation and execution of converged services, meeting the rigorous performance requirements proper of telecom services (high availability, low latency, asynchronous behavior, etc.) and allowing its interoperability with different Web technologies and communications protocols.

Converged services and the different areas that comprise this concept (creation, composition and execution) now receive special attention, representing a major research focus on the computation and communications domain. Specifically, within the converged service execution area, one of the main challenges is to carry out effective and accurate monitoring of such services, allowing the detection of possible faults that may arise at runtime so that appropriate measures can be taken to ensure correct fault-handling and compliance with the Quality of Service (QoS) constraints established between service providers and end-users.

Currently, different environments and platforms for the creation and development of converged services focus on providing adequate functional structure to allow correct interoperability between different services from both telecommunications and Web domain. In other words, these platforms allow the composition of these functions to achieve more complex service logic and deliver value-added services to the end-user. However, during the design, creation and composition of these converged services, functional requirements that permit runtime monitoring and therefore an appropriate fault management are not considered. Taking this into account, in this proposal we make use of techniques for automatic code instrumentation which allow providing the converged services with the necessary functions for runtime monitoring and faulting detection without having to return to the creation stage.

In this paper we propose and develop a scheme for runtime monitoring and fault detection of converged services built on a JAIN SLEE environment and based on techniques of automatic code instrumentation which considers services from both Web and telecommunications domain. The rest of this paper is arranged as follows. Section 2 presents a conceptual base of the different technologies and concepts related with this work. Section 3 describes different proposals that address service monitoring. Section 4 gives a detailed description of our proposed monitoring scheme. On section 5 we present a case study through which an evaluation of our proposal is...
carried out. We end the paper with a conclusions and future work section.

II. BACKGROUND

A. TelComp2.0

The present work is framed within the Project TelComp2.0: Retrieval and Composition of Complex Components for the Creation of Telco 2.0 Services [7] funded by COLCIENCIAS [8] and developed by the Telematics Engineering Group [9] of the University of Cauca. The TelComp2.0 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. This platform has been built to be deployed over the network infrastructure of a telecommunications operator, integrating with the existing services that it provides and allowing the operator to rapidly build new and more complex services that combine Web 2.0 functionalities, which are referenced by WSDL descriptors on a service repository. Figure 1 presents a high level overview of the TelComp 2.0 platform and its process to create new converged services.

B. JAIN SLEE

As stated before, the TelComp2.0 execution environment is based on the JAIN SLEE specification, since this technology defines a standard environment for executing service logic and specifies the manner in which portable and high quality telecommunications services can be built, operated and executed. This specification defines a component model for structuring the application logic as a set of reusable object-oriented components, with the ability to reorganize them into more complex services.

Below is a description of the fundamental concepts involving the JAIN SLEE specification:

1) Service Building Block: The reuse element defined by JAIN SLEE is the service building block (SBB). An SBB is a software component that sends and receives events and also runs computational logic based on receipt of events and their current status. The programming code of a SBB is built upon Java classes.

2) Event: An event represents an occurring that may require processing by an application. An event can originate from different sources, for example, an external protocol stack or from within the SLEE.

3) Resources and Resource Adaptor: Resources are external entities that interact with other systems outside the SLEE, as network elements (HLR, MSC, etc.), protocol stacks, directories and databases. A resource adapter implements the interface of a resource within the JAIN SLEE environment.

Additionally, the JAIN SLEE framework provides a management oriented function known as the Alarm Facility, which allows the establishment of alarms that are used by the SBBs and Resource Adapters to generate notifications over meaningful events such as possible failures. These alarms are meant to be handled by an external management client through JMX (Java Management Extensions) interfaces provided by the SLEE. Each alarm has associated information such as the type of the alarm, the source, the level of the alarm and a message; all defined at the time the alarm is set.

C. Converged Service

Today it can be clearly identified three approaches when it comes to services, first is the Web approach with the new social network services (Facebook, Twitter, LinkedIn, etc.), the second is from telecommunications operators with traditional services (voice call, SMS, video, etc.) and the third are converged services which are understood as the coordination of a set of services from different vendors and sources, and are perceived by the end-user as a single service [10]. Taking this into account, TelComp 2.0 defined a general structure for converged services over a JAIN SLEE execution environment as shown in Figure 2, where the general logic structure of a converged service based on a main orchestrator SBB that communicates with a set of atomic services through firing and receiving events, which may be provided by resource adaptors or defined by each atomic service as custom events developed for specific tasks, allowing the exchange of messages and information between services.

D. Service Monitoring

The increasing complexity of the requirements in the development and deployment of converged services reveals a major challenge in the appropriate management
that should be given to them and their components. Through monitoring is possible to observe and record in detail the behavior of Web and Telco atomic services composing a converged service, so that it can be analyzed whether a service is working properly or not. This fact becomes important to the extent that a single atomic service being unavailable or presenting a significant delay on its response time may cause a major failure to the converged service, compromising QoS constraints previously defined in the agreements established between service providers and end-users. Therefore, it becomes crucial for service providers to implement monitoring mechanisms that help them to analyze and rapidly detect possible failures of the converged services at runtime.

E. Javassist

Javassist technology [11] [12] is a set of tools for developing Java-bytecode translators that allow transforming Java class files (bytecode) at runtime. Unlike other bytecode editors that are not based on reflection, Javassist API provides two levels: a source-level and a bytecode-level. The source-level API allows to modify a Java class file without specific knowledge of the Java bytecode syntax, by defining directly with Java source code the new bytecode that will be modified on the Java class file. This API takes the provided source code and compiles it on the fly to later insert it in the class file as bytecode. On the other hand, the bytecode-level API allows to directly edit the class file like any other bytecode editor. Javassist is commonly used to add new methods or functionalities to a compiled class and for reflection at runtime, as it enables Java programs to make use of meta-objects that controls the calling of the methods of a class without having to use a special compiler or virtual machine. Javassist was chosen within this work due to its easiness of use on editing compiled Java classes through its source-level API.

III. RELATED WORK

Currently, a significant number of proposals address different issues on service monitoring, mainly focused on the Web service domain. Also, there is a strong trend towards the definition of composite service monitoring policies and techniques, performed at runtime and based on QoS constraints.

Cabrera and Franch [13] present the analysis of a set of tools for Web service monitoring in terms of Service Level Agreements (SLA) and develop a model based on the quality standard ISO/IEC 25010 [14]. This model seeks to provide information that may contribute to select an adequate monitoring tool for a specific situation, through the analysis of each tool features. Haiteng et al. [15] propose a monitoring architecture that seeks to clearly separate the business logic of the Web services from the monitoring associated functionalities. For this, a Monitor Broker is introduced within the traditional web services architecture, to collect and manage information about QoS values at runtime. This Monitor Broker is implemented as a web service, which has two main components: Service Monitor, which is responsible for capturing information at the time the service is invoked, and the Result Collector, which takes the information that has been collected and stores it in a database to perform monitoring tasks.

Goel and Shyamasundar [16] design and develop a runtime monitoring architecture, where the Web service engine and the monitoring tools work concurrently, accessing the entire message exchange, without affecting the normal operation of the application being monitored. The functional properties to be monitored are defined through SLA. Li et al. [17] propose an adaptation of composite services that supports a system to monitor the network environment in which its main function is to monitor the system so that this meets the minimum execution times of a composite service established in its SLA. Additionally, this system provides information about the execution time offset caused by changes in the network environment and defines a set of rules over which a failure detection algorithm is built.

Falcarin and Walter [18] implement a registration framework able to insert monitoring code on JAIN SLEE applications. For this purpose it makes use of an implementation framework called JBoss-AOP (JBoss Aspect-Oriented Programming). This framework allows intercepting a method at the time it is called, and insert additional code into it seamlessly. For this, a XML (eXtensible Markup Language) file is defined which specifies the code insertion points of the application. Although this work defines a mechanism for monitoring code insertion in JAIN SLEE applications on runtime, it does not specify a method for handling the monitoring information neither an implementation of the mechanism.

The analysis carried out on the different proposals on service monitoring, presents an important gap on the application domain they are focused on, considering the concept of converged service, since most of them center their efforts on defining monitoring mechanisms.
exclusively on Web services. Thus, architectures and posed schemes cannot be applied to the monitoring of converged services, since both execution platforms and the features of each domain, have different operating characteristics such as synchronous behavior for Telco services. Taking into account the TelComp 2.0 project as an application framework and the analysis performed on the related works, in the next section we propose a monitoring scheme that considers the converged services complexity and uses automatic code instrumentation techniques to equip such services with the necessary monitoring and fault detection functions at runtime.

IV. MONITORING SCHEME

The proposed scheme for converged service monitoring follows a reactive approach by automatically instrumenting the converged services currently deployed in the execution environment at runtime. This code insertion allows to detect failures during the execution flow of the converged service and also to obtain information regarding the failure to take the appropriate measure. This mechanism is introduced in the execution environment (Mobicents JAIN SLEE [19]) as a JAIN SLEE application and is executed in parallel with the deployed converged services. This application uses a main SBB named Code Injector which contains the instrumentation logic to access and modify the converged services. Figure 3 shows a general diagram of the SLEE with the instrumentation application deployed.

The Code Injector module is responsible for detecting the deployment of a new converged service on the execution environment. This is made possible through the ServiceStartedEvent provided by the JAIN SLEE specification and which is fired every time a new service is deployed. Once a new service deployment is detected, it validates that the deployed service is a converged service and not a Web or Telco atomic service, through an analysis of the special code structure characteristic of the converged services. After validating that the new deployment corresponds to a service that needs to be monitored, the Code Injector module uses the Javassist API and its reflection mechanisms to access the orchestrator SBB class of the converged service and identify the different methods that correspond to atomic service invocations. These methods represent a critical point on the execution flow of the converged service because they handle the response events fired after the invocation of each atomic service and proceed to invoke the next, as previously defined on that execution flow with the proper data mapping between services. If one of these invocations results on a failure due to an atomic service being unavailable or a network problem, it may compromise the correct performance of the whole converged service, making necessary for technical operators to run diagnostic tests and extensive server log analysis to determine what went wrong. Considering the above, it is convenient to have monitoring functions on such points that help to detect which atomic service has failed and also the exact location of this failure on the execution flow to facilitate the repair tasks for technical operators.

Following this approach, the Code Injector module uses the Javassist source-level API to instrument each response event handler establishing the necessary functions to monitor the behavior of atomic services and detect possible failures. The inserted code is able to determine whether an atomic service has been successfully executed or not, and in case of a failure it raises an alarm through the Alarm Facility defined by the JAIN SLEE specification containing information about the source and the location on the execution flow of the failure.

As stated on section II, these alarms could be managed through an external management client which communicates with the SLEE using JMX interfaces. In this approach we use the Mobicents JAIN SLEE Management Console to access the alarms and retrieve the failure information. It is important to note that the instrumented code on the orchestrator SBB class of converged services does not modify its normal execution flow but provides the necessary functions to perform the proper monitoring tasks.

V. IMPLEMENTATION AND CASE STUDY

The implementation of the proposed monitoring scheme is tested using a converged service named LinkedIn Job Notificator which was built using the TelComp2.0 platform. This service searches for job offers associated to the user LinkedIn profile and notifies him/her through his/her Twitter account, email and through a voice call service that takes the job offer and using a Text-to-Speech engine converts it to an audio message. Figure 4 presents the execution flow of each atomic service used in this converged service. LinkedIn Job Notificator provides a complex functionality that integrates Web 2.0 services (LinkedIn, Twitter) and telecommunications features (email, voice call). Specifically it is composed by two Web services (LinkedIn-WS, Twitter-WS), two Telco services (Email-TS, MediaCall-TS) and a data access service (DataAccess) that retrieves the user information at runtime.

The service is triggered by the user from its mobile device. Once the service receives the initial request it identifies the user and get the LinkedIn user information
through the `DataAccess` service (step 1). Later it invokes the `LinkedIn-WS` to retrieve a new job offer (step 2). Following with the execution flow, the service invokes the `DataAccess` service to get the user’s Twitter Id and email address (step 4) and sends the job offer information through the `Twitter-WS` and `Email-TS` (step 5). Finally it invokes the `DataAccess` service to get the user’s phone number (step 7) and sends the job offer as an audio message through the `MediaCall-TS` (step 8).

Figure 4. LinkedIn Job Notificator service execution flow.

**A. Functional Evaluation**

For the functional evaluation of the monitoring scheme, we install and deploy the `Code Injector` application on the Mobicents JAIN SLEE server which will be standing by waiting for new converged service deployments. Later, the `LinkedIn Job Notificator` service is deployed immediately activating the `Code Injector` application which analyses the orchestrator SBB class of the converged service and instruments it with the necessary monitoring code. The analysis and instrumentation process itself takes 40ms to be performed on this converged service; however for the new changes on the service class to take place is necessary to make a redeploy of the service, which is performed automatically by the server, increasing the process time to 6 seconds. After this the converged service is ready to process requests and contains the monitoring functions.

Figures 5 and 6 present a fragment of the orchestrator SBB class before and after being instrumented with the monitoring code by the `Code Injector` application. In this specific case, there is a code insertion in the method that handles the response of the `LinkedIn-WS` on the second step of execution of the converged service. This code allows detecting failures and raising an alarm if so. We are able to present Figures 5 and 6 through an online tool named Show My Code [20] which allows to observe the code from a compiled Java class.

Figure 5. Original orchestrator SBB class.

Figure 6. Instrumented orchestrator SBB class.

As shown on Figure 6, between lines 141-155 of the method `onEndWSInvocatorEvent` there is a new fragment of code which establishes the functions to raise an alarm on the event of a failure. Afterwards, we establish a scenario where the `LinkedIn-WS` is unavailable to create a failure on the execution flow of the converged service. This, with the purpose of verifying that upon the failure the converged service establishes the proper alarm and it is visible on the Mobicents JAIN SLEE Management Console. Figure 7 shows that, as expected, on the management console there is an active alarm which indicates that the `LinkedIn Job Notificator` has presented a failure on the `LinkedIn-WS` and in the step 2 of the execution flow.

Figure 7. Mobicents JAIN SLEE alarms management console.
As shown above, is demonstrated the proper functionality of the monitoring scheme and the automatically instrumented fault detection functions. This greatly facilitates the management tasks associated to this converged service, allowing technical operators to easily identify malfunctioning atomic services and if possible repair them or proceed to reconfigure the converged service structure.

B. Performance Evaluation

To analyze the proposed monitoring scheme performance we use a Web tool named Jolokia [21], which allows to access remotely to Java MBeans defined by the JMX specification, through a HTTP/JSON middleware. With this, is possible to access the management interfaces of the JAIN SLEE server and retrieve behavioral information of the system. Specifically, for this analysis we access the MBean that provides memory use information of the server. First, is measured the memory use of the server with the basic atomic services and the LinkedIn Job Notificator service deployed for an interval of 15 seconds. Then we measure the memory use of the execution environment adding the Code Injector application during the same time interval. Figure 8 presents the results of those measurements.

The increase in memory usage of the execution environment with the monitoring application being executed has an average of 6.2 Megabytes for the time interval measured. Therefore, it is possible to state that once the monitoring application is included in the execution environment, this will have a minimum impact on the memory use of the system.

VI. CONCLUSIONS AND FUTURE WORK

This paper proposed a new initiative to address the problem of monitoring converged services taking into account both Web and telecommunications domain and based on the TelComp 2.0 approach. This paper, also introduces the use of code instrumentation techniques to automatically detect and analyze the structure of converged services, and provide them with monitoring and fault detection functions without modifying the service business logic. This approach enables the provisioning of previously built and hard coded converged services with the necessary monitoring and fault detection capabilities that help system managers and technical operators to easily identify and properly correct service failures. Runtime code instrumentation techniques represent an important tool on the converged service area because its application is not limited to monitoring functions and could be extended to implement automatic reconfiguration and adaptation processes. As future work, we are planning to extend this work adding reconfiguration mechanisms which allow to automatically handle and correct the possible failures detected by the monitoring scheme.

ACKNOWLEDGMENT

The authors would like to thank University of Cauca and TelComp 2.0 project (Code: 1103-521-28338 CT458-2011) for supporting this work and the MSc. student Julián Andrés Rojas.

REFERENCES