MONITOREO DE SERVICIOS CONVERGENTES EN ENTORNOS JAIN SLEE

Universidad
del Cauca

DIEGO FELIPE ADRADA GÓMEZ
ESTEBAN DAVID EDGARDO SALAZAR LÓPEZ

Monografía presentada para optar al título de Ingeniero en Electrónica y Telecomunicaciones

Director: PhD. Ing. Juan Carlos Corrales Muñoz

Asesor: Msc(c). Ing. Julián Andrés Rojas Meléndez

Universidad del Cauca
FACULTAD DE INGENIERÍA ELECTRÓNICA Y TELECOMUNICACIONES
GRUPO DE INGENIERÍA TELEMÁTICA
POPAYÁN, ENERO DE 2014

TABLA DE CONTENIDO

Pág.

1. INTRODUCCIÓN 1

1.1 Contexto General 1

1.2 Escenario de Motivación 2

1.3 Definición del Problema 2

1.4 Objetivos 4

1.4.1 Objetivo General 4

1.4.2 Objetivos Específicos 4

1.5 Alcance 4

1.6 Contribuciones 5

1.7 Contenido de la Monografía 6

2. ESTADO ACTUAL DEL CONOCIMIENTO 7

2.1 Base Conceptual 7

2.1.1 Servicios Convergentes 7

2.1.2 Especificación JAIN SLEE 8

2.1.2.1 Conceptos Fundamentales de JAIN SLEE 9

2.1.2.2 JAIN SLEE *Facilities* 10

2.1.3 Calidad de Servicio (QoS) 10

2.1.4 Monitoreo de Servicios 11

2.2 Trabajos Relacionados 11

2.2.1 Modelos para el monitoreo de servicios 11

2.2.2 Enfoque Orientado a Aspectos para el Monitoreo Dinámico de un Entorno de Ejecución de Lógicas de Servicio (SLEE) 12

2.2.3 Monitoreo en Tiempo de Ejecución de Servicios Web Implementados en BPEL..... 13

2.2.4 Monitoreo Automático de Acuerdos de Nivel de Servicio de Servicios Web 13

2.2.5 Monitoreo del Entorno de Red Soportado en la Adaptación de Servicios Compuestos 14

2.3 Brechas del Conocimiento 14

3. DEFINICIÓN DE MÉTRICAS Y REGLAS PARA EL MONITOREO DE SERVICIOS CONVERGENTES 17

3.1 Evaluación de herramientas para el monitoreo de servicios 17

3.1.1 Criterios para la evaluación de herramientas 18

3.1.1.1 Parámetros de Calidad de Servicio (QoS) 18

3.1.1.2 Tipo de servicios monitoreados 20

3.1.1.3 Tipo de implementación 21

3.1.2 Evaluación de herramientas 21

3.2 Métricas y Reglas propuestas 24

4. DEFINICIÓN DE MECANISMO DE CONTROL PARA EL MONITOREO DE SERVICIOS CONVERGENTES 27

4.1 Servicios Convergentes 27

4.2 Mecanismo de monitoreo 31

4.2.1 Módulo de Monitoreo 32

4.2.1.1 Detección de nuevos servicios convergentes 33

4.2.1.2 Módulo Code Injector 33

4.2.1.2.1 Instrumentación de código en tiempo de ejecución 33

4.2.1.2.2 Detección de puntos de control e inserción de código 34

4.2.1.3 Módulo Service Monitor 36

4.2.1.3.1 Monitoreo de servicios atómicos 36

4.2.1.3.2 Persistencia de Datos 37

4.2.1.4 Módulo QoS Monitor 38

4.2.1.4.1 Acceso a datos 39

4.2.1.4.2 Cálculo de métricas y reglas de QoS 39

4.2.1.5 Módulo Alarm Monitor 40

4.2.1.5.1 Detección de Alarmas y conexión con el Módulo de Reconfiguración 40

4.2.2 Módulo Monitoring Data 41

4.2.2.1 Estructura de la base de datos 43

4.2.3 Módulo Monitoring View 45

4.2.3.1 Estructura general del Módulo Monitoring View 45

4.2.3.1.1 *Web User Interface* 45

4.2.3.1.2 *Server Code Execution* 46

5. IMPLEMENTACIÓN Y RESULTADOS 48

5.1 Descripción del Prototipo 48

5.1.1 Modelo de Casos de Uso del Sistema 48

5.1.2 Descripción de la Arquitectura de Referencia 51

5.1.2.1 Soporte 52

5.1.2.1.1 Software de implementación de la capa de soporte 52

5.1.2.2 Lógica de Negocio 52

5.1.2.3 Lógica de Presentación 52

5.1.3 Diagrama de Despliegue 53

5.2 Pruebas del Prototipo 54

5.2.1 Evaluación Funcional 54

5.2.1.1 Servicio *LinkedIn Job Notificator* 54

5.2.1.2 Enfoque Reactivo del Mecanismo de Monitoreo 55

5.2.1.3 Enfoque Proactivo del Mecanismo de Monitoreo 60

5.2.1.4 Módulo Alarm Monitor y conexión con el Módulo de Reconfiguración 65

5.2.1.5 Módulo de Monitoring View 67
5.2.2 Evaluación de Rendimiento del Sistema 68
5.2.2.1 Enfoque Reactivo 68
5.2.2.2 Enfoque Proactivo 69

6. CONCLUSIONES, CONTRIBUCIONES Y TRABAJO FUTURO 71
6.1 Conclusiones 71
6.2 Contribuciones 72
6.3 Trabajo Futuro 73

REFERENCIAS 74

INDICE DE FIGURAS

Pág.

Figura 1-1. Arquitectura en Capas de JAIN SLEE *9*
Figura 2-1. Servicio Web Compuesto *20*
Figura 4-1. Vista General de la plataforma TelComp2.0 ..... *28*
Figura 4-2. Estructura general de un servicio convergente *29*
Figura 4-3. Flujo de ejecución servicio convergente básico *30*
Figura 4-4. Estructura de un servicio convergente básico *30*
Figura 4-5. Vista General del Mecanismo de Monitoreo *32*
Figura 4-6. Esquema general de un documento para el modelo de datos incrustados *43*
Figura 4-7. Documento asociado al mecanismo de monitoreo *44*
Figura 4-8. Estructura general de la base de datos implementada *44*
Figura 4-9. Estructura general del módulo Monitoring View *45*
Figura 5-1. Diagrama de casos de uso del sistema *50*
Figura 5-2. Arquitectura de Referencia del sistema *51*
Figura 5-3. Diagrama de Despliegue del sistema *53*
Figura 5-4. Flujo de ejecución del servicio LinkedIn Job Notificator Service *55*
Figura 5-5. Consola de Administración Mobicents JSLEE, despliegue servicio atómico *56*
Figura 5-6. Consola de Administración Mobicents JSLEE, despliegue servicio convergente *57*
Figura 5-7. Clase SBB Orquestadora original *58*
Figura 5-8. Clase SBB Orquestadora instrumentada por el mecanismo de monitoreo *58*
Figura 5-9. Alarma activada en la consola de gestión de Mobicents JSLEE *59*
Figura 5-10. Envío de peticiones a las WSDL de servicios atómicos *61*
Figura 5-11. Recepción de respuesta de peticiones a las WSDL de servicios atómicos *61*
Figura 5-12. Base de datos en MongoDB con valores de QoS almacenados *62*
Figura 5-13. Calculo de parámetros de QoS por parte del módulo QoS Monitor *63*
Figura 5-14. Valores de QoS para el servicio convergente *63*
Figura 5-15. Calculo de parámetros de QoS para el escenario no deseado 1 ..... *64*
Figura 5-16. Valores de QoS para el escenario no deseado 1 ..... *64*
Figura 5-17. Alarma activada en la consola de gestión de Mobicents JSLEE *65*
Figura 5-18. Envío de evento por parte del módulo Alarm Monitor *66*
Figura 5-19. Recepción del evento por parte del Módulo de Reconfiguración *66*
Figura 5-20. Gráfica de valores en tiempo real de disponibilidad y tiempo de respuesta *67*
Figura 5-21. Uso de memoria con el enfoque reactivo del mecanismo *68*
Figura 5-22. Uso de memoria con el enfoque proactivo del mecanismo *69*

INDICE DE TABLAS

Pág

Tabla 2-1. Brechas de los trabajos relacionados *9*
Tabla 3-1. Presencia de los parámetros en cada herramienta *21*
Tabla 3-2. Valor de N_v para cada parámetro de QoS *22*
Tabla 3-3. Peso de cada parámetro de QoS *22*
Tabla 3-4. Valor de PT_{par} para cada herramienta analizada *22*
Tabla 3-5. Valor de T_{sm} y T_{imp} para cada herramienta analizada *23*
Tabla 3-6. Valor de calificación final para cada herramienta analizada *23*
Tabla 3-7. Herramientas ordenadas de acuerdo a la calificación *24*
Tabla 3-8. Métricas para medir disponibilidad y tiempo de respuesta *25*
Tabla 4-1. Códigos de Estado de una respuesta HTTP *37*
Tabla 4-2. Datos almacenados en la base de datos implementada *44*
Tabla 5-1. Descripción caso de uso: Detectar Servicio Convergente *48*
Tabla 5-2. Descripción caso de uso: Instrumentar Código *49*
Tabla 5-3. Descripción caso de uso: Recolectar valores de QoS *49*
Tabla 5-4. Descripción caso de uso: Calcular métricas de QoS *49*
Tabla 5-5. Descripción caso de uso: Generar Alarmas *50*
Tabla 5-6. Descripción caso de uso: Informar Fallos *50*
Tabla 5-7. Especificaciones del sistema *52*
Tabla 5-8. Pruebas enfoque reactivo del mecanismo de monitoreo *55*
Tabla 5-9. Pruebas enfoque proactivo del mecanismo de monitoreo *60*
Tabla 5-10. Pruebas módulo Alarm Monitor *66*
Tabla 5-11. Pruebas módulo Monitoring View *67*
Tabla 5-12. Especificaciones técnicas del Equipo empleado para las pruebas *68*

Capítulo 1

1. Introducción

1.1 Contexto General

La Web ha evolucionado desde la consulta de información por medio de simples páginas, hasta la extracción personalizada de la misma a través de aplicaciones y servicios web. En la actualidad, es posible encontrar una amplia oferta de recursos y servicios, entre los que se encuentran comunidades virtuales, servicios de redes sociales, blogs, wikis y mashups. Este nuevo paradigma de la Web denominado Web 2.0 ha conducido a un cambio radical en cómo las personas y las empresas perciben la información, a tal punto que ha modificado la manera en que interactúan con esta. En este contexto, la industria de las telecomunicaciones ha generado un nuevo modelo conocido como Telco 2.0 [1], el cual relaciona los conceptos, servicios y tecnologías de la Web 2.0 con los servicios tradicionales de telecomunicaciones, permitiendo que los operadores amplíen su portafolio de servicios y que los usuarios puedan relacionarse con estos de manera más directa y confiable. Este nuevo tipo de servicios son conocidos como servicios convergentes dado que integran las funcionalidades tradicionales del dominio de las telecomunicaciones (voz, video y datos) con servicios de información pertenecientes al dominio de la Web, generando de esta forma nuevos servicios con funcionalidades más complejas y orientadas a satisfacer las necesidades de usuarios cada vez más exigentes.

Para el desarrollo y ejecución de nuevos servicios convergentes es necesario contar con plataformas robustas, que permitan llevar a cabo la integración de las diferentes tecnologías y protocolos de comunicación característicos de los servicios, tanto del dominio de las telecomunicaciones como del dominio de la Web. Bajo este propósito, existen tecnologías como la especificación SIP Servlets, Converged Service Studio de Ericcson [2], la plataforma uReach CSF (*Converged Services Framework*) de Alcatel-Lucent [3], entre otros. Sin embargo, una alternativa que sobresale es la especificación JAIN SLEE [4] [5] la cual plantea un entorno robusto y estándar para el desarrollo y ejecución de servicios convergentes, satisfaciendo rigurosos requerimientos de los servicios de telecomunicaciones (alta disponibilidad, baja latencia, asincronía, etc.) y permitiendo la interoperabilidad de diferentes tecnologías y protocolos de comunicación.

Esta creciente complejidad de los requerimientos en el desarrollo y despliegue de servicios convergentes, revela un reto importante en el manejo y apropiada gestión que se debe dar a los sistemas computacionales que soportan dichos procesos, dando origen a nuevos modelos para su adecuada gestión, basados en la computación autónoma [6], donde se busca automatizar las funciones del sistema de tal forma que éste sea capaz de adaptarse automáticamente a situaciones que condicionen su correcto funcionamiento. Dentro de este tipo de funciones se encuentra el monitoreo de los componentes y servicios que hacen parte de dicho sistema. A través del monitoreo es posible observar y registrar de forma detallada su comportamiento de tal forma que se pueda analizar sí un servicio está funcionando de manera adecuada o sí presenta fallas.

1.2 Escenario de Motivación

Actualmente el desarrollo de servicios convergentes y las diferentes áreas que comprenden este concepto (creación, composición y ejecución) reciben especial atención por parte de la academia y la industria, representando un foco de investigación importante. Específicamente, dentro del área de la ejecución de servicios convergentes, los esfuerzos se encuentran orientados hacia el desarrollo de plataformas que soporten y cumplan con los diferentes requerimientos de este tipo de servicios, al tiempo que garantizan su correcto funcionamiento.

Uno de los principales retos en el área de la ejecución de servicios convergentes es realizar un monitoreo efectivo y preciso de dichos servicios, el cual permita detectar los posibles fallos que se puedan presentar en tiempo de ejecución, de tal forma que se puedan tomar las medidas pertinentes para garantizar el cumplimiento de los parámetros de Calidad de Servicio (QoS) establecidos entre cliente y proveedor. De esta forma, existen propuestas y aproximaciones que abordan la problemática del monitoreo de los servicios a través de mecanismos y técnicas de monitoreo de servicios compuestos, sin embargo, estos trabajos presentan una visión centrada en los servicios pertenecientes al dominio de la web, dejando de lado su integración con las funcionalidades del dominio de las telecomunicaciones y por lo tanto su aplicación a los nuevos servicios convergentes.

1.3 Definición del Problema

En la actualidad se pueden identificar claramente tres enfoques cuando se habla de servicios, el primero es el de la Web, con los nuevos servicios sociales (Facebook, Linkedin, Twitter, etc); el segundo desde los operadores de telecomunicaciones, con servicios tradicionales (buzón de mensajes, llamada, SMS, etc); y el tercero con servicios convergentes, los cuales se entienden como la coordinación de un conjunto de servicios de diferentes proveedores, tales que a la vista del usuario final sean un solo servicio [7].

De esta forma, el paradigma Telco 2.0 permite la existencia de una amplia gama de servicios convergentes, orientados a satisfacer los requerimientos de usuarios cada vez más exigentes. Debido a esto, los proveedores de servicios de telecomunicaciones (Telcos) buscan mejorar su competitividad para crecer y mantenerse en la industria, razón por la cual estos se encuentran en la búsqueda de nuevos mecanismos que permitan una adecuada gestión de los sistemas que soportan la ejecución de dichos servicios. Por lo tanto, una función que toma gran importancia es el monitoreo, ya que a través de este es posible detectar fallos en los servicios para posteriormente tomar las medidas necesarias que permitan corregirlos, y de esta forma evitar que el usuario final perciba un detrimento en la calidad del servicio, lo cual trae como consecuencia una mala percepción del operador.

Como resultado de lo anterior, se hace necesario tener en cuenta la forma en que se aborda el monitoreo para cada tipo de servicios, la cual se explica a continuación:

- ✓ En el mundo de la Web, se han realizado varios trabajos que abordan la construcción de modelos, métodos y sistemas de monitoreo que buscan garantizar la disponibilidad y accesibilidad de los servicios, con el fin de cumplir con determinados parámetros de Calidad de Servicio (QoS) o con requerimientos establecidos entre clientes y proveedores mediante los Acuerdos de Nivel de Servicio (SLA) [8]. Sin embargo, la diversidad en el tipo de implementación realizada y los parámetros que se tienen en

cuenta para realizar las tareas de monitoreo, hacen que no se cuente con un mecanismo estándar que defina el proceso de monitoreo de servicios web.

- ✓ En el dominio de las telecomunicaciones dada la naturaleza de los servicios existentes, los cuales son desarrollados con altos niveles de disponibilidad, las tareas de monitoreo se enfocan principalmente en analizar el tráfico de paquetes y la seguridad en la red, en su mayoría haciendo uso de herramientas y software de carácter propietario [9].

- ✓ En el entorno convergente, plataformas basadas en la especificación JAIN SLEE que permiten la ejecución de servicios convergentes como Mobicents [10], cuentan con su propia consola de gestión, la cual brinda herramientas a un usuario administrador, para interactuar con el SLEE, permitiendo realizar las funciones comunes de gestión como el despliegue e instalación de Unidades Desplegables[1], configuración de componentes, recursos y servicios. Sin embargo, las tareas de monitoreo se limitan simplemente a observar e inspeccionar las actividades de JAIN SLEE.

Teniendo en cuenta lo mencionado anteriormente en los tres enfoques, se puede resumir lo siguiente: las plataformas basadas en la especificación JAIN SLEE permiten la ejecución de servicios convergentes, los cuales no cuentan con el mismo grado de disponibilidad de los servicios de telecomunicaciones. Por lo tanto, es de vital importancia contar con un mecanismo de monitoreo para este tipo de servicios, el cual permita detectar los posibles fallos que puedan presentarse en tiempo de ejecución, de tal forma que se puedan tomar las medidas necesarias para garantizar un correcto funcionamiento de los mismos. Actualmente, la especificación JAIN SLEE no cuenta con este tipo de mecanismos que permitan realizar un monitoreo adecuado sobre los servicios. Adicionalmente, un inconveniente importante es que no se han definido cuáles son los parámetros que se deben tener en cuenta para realizar dicho monitoreo.

Teniendo en cuenta el problema planteado anteriormente, surge la siguiente pregunta de investigación:

¿Cómo proporcionar un mecanismo eficiente de monitoreo de servicios convergentes en entornos JAIN SLEE?

[1] Unidad Desplegable: Es un archivo .jar que puede ser instalado en el SLEE y contiene la información necesaria para el funcionamiento de un servicio (SBBs, eventos, librerías, etc).

1.4 Objetivos

1.4.1 Objetivo General

Definir un mecanismo para el monitoreo de servicios convergentes en entornos JAIN SLEE.

1.4.2 Objetivos Específicos

- ✓ Definir un conjunto de métricas y reglas para el monitoreo de servicios convergentes, de acuerdo a parámetros no funcionales de QoS a nivel de servicio.

- ✓ Definir un mecanismo que seleccione los puntos de control adecuados para realizar el monitoreo de servicios convergentes en tiempo de ejecución.

- ✓ Evaluar experimentalmente el monitoreo de servicios convergentes en entornos JAIN SLEE basado en QoS a través de un prototipo.

1.5 Alcance

El principal objetivo de este trabajo es definir un mecanismo para el monitoreo de servicios convergentes el cual tiene en cuenta tanto el dominio web como el de las telecomunicaciones, considerando dos enfoques del monitoreo de servicios: reactivo y proactivo. El primero de ellos permite ejecutar un control de fallos durante la ejecución de los servicios convergentes teniendo en cuentos los puntos críticos de dicho proceso, el segundo brinda la posibilidad de observar ciertos parámetros no funcionales de calidad de servicio definidos para los servicios convergentes, y emitir alarmas ante una violación de los valores establecidos para estos. Finalmente el mecanismo de monitoreo propuesto es validado a través del desarrollo de un prototipo.

Adicionalmente, para el despliegue de servicios convergentes, existen diversas plataformas, algunas de ellas como Rihno [11], son de carácter propietario y requieren licencias de pago para su uso, sin embargo, Mobicents [10] cuenta con una licencia de carácter abierto, por lo cual fue la plataforma escogida para el despliegue y desarrollo del prototipo experimental del trabajo de grado.

Finalmente, para un caso de estudio de un servicio convergente, se desarrollan una serie de pruebas tanto funcionales como de rendimiento, las cuales permitieron demostrar la aplicabilidad del mecanismo propuesto en este tipo de servicios.

1.6 Contribuciones

- ✓ Este trabajo es un aporte al proyecto de Maestría en Ingeniería Telemática de la Universidad del Cauca, titulado *Reconfiguración Automática de Servicios Convergentes en Entornos JAIN SLEE,* desarrollado por el Mag. (c) Julián Andrés Rojas Meléndez, el cual se encuentra enmarcado dentro del proyecto *Telcomp2.0: Recuperación y Composición de Componentes Complejos para la Creación de Servicios Telco 2.0* [12], llevado a cabo por el Grupo de Ingeniería Telemática de la Universidad del Cauca y financiado por COLCIENCIAS [13]. Proyecto Código: 1103-521-28338, Contrato RC 458-2011.

- ✓ Un análisis y evaluación de las diferentes herramientas vigentes para el monitoreo de servicios web.

- ✓ Un conjunto de métricas y reglas para el monitoreo de servicios convergentes de acuerdo a parámetros no funcionales de QoS asociados a servicios convergentes.

- ✓ Un mecanismo para el monitoreo de servicios convergentes, el cual considera tanto el dominio de la web como el de las telecomunicaciones y está basado en software de libre distribución.

- ✓ Un prototipo experimental, donde se muestra el proceso de monitoreo de servicios convergentes en un entorno JAIN SLEE.

- ✓ Se destaca la producción del artículo titulado: "*Automatic Code Instrumentation for Converged Service Monitoring and Fault Detection*" aceptado en la conferencia "*The 28th IEEE International Conference on Advanced Information Networking and Applications (AINA-2014)*", la cual se celebrará en Victoria, Canada, los días 13-16 de Mayo de 2014. Este artículo se presentará dentro del Workshop "*2nd International Workshop on Network Management and Monitoring (NetMM 2014*)". Los artículos aceptados en esta conferencia son indexados dentro de la base de datos de IEEE Computer Society, IEEE Xplore y DBLP.

- ✓ Se destaca la producción de artículo titulado: "*Automatic Monitoring of Converged Services on a Telco2.0 Environment based on QoS Constraints*" enviado a la revista Ingeniería y Universidad de la Pontifica Universidad Javeriana, indexada en Categoría A2 de Publindex - Colciencias, el cual se encuentra en este momento en periodo de revisión.

1.7 Contenido de la Monografía

El presente trabajo de grado contiene los siguientes capítulos:

Capítulo 2. Estado actual del conocimiento

En este capítulo se abordan las definiciones formales de conceptos y tecnologías claves en las que se fundamenta el monitoreo de servicios convergentes, además del estado actual del conocimiento relacionado con trabajos que se han adelantado sobre las diferentes áreas que abarca este proyecto.

Capítulo 3: Definición de métricas y reglas para el monitoreo de servicios convergentes

En este capítulo se realiza una evaluación de las diferentes herramientas para el monitoreo de servicios web, a partir de la cual se definen un conjunto de métricas y reglas que se pueden utilizar para el monitoreo de servicios convergentes, basadas en parámetros no funcionales de QoS.

Capítulo 4: Definición de mecanismo de control para el monitoreo de servicios convergentes

Este capítulo refleja la contribución más importante del presente trabajo de grado. Es aquí donde se define el mecanismo para el monitoreo de servicios convergentes en un entorno JAIN SLEE. Bajo un enfoque de monitoreo reactivo, se parte de la definición de los servicios convergentes y la detección de los puntos críticos donde es necesario realizar un control en tiempo de ejecución. Así mismo, se muestra también un enfoque proactivo para realizar el monitoreo, de acuerdo a las reglas de QoS definidas en el capítulo 3.

Capítulo 5: Implementación y Resultados

En este capítulo se aborda la descripción detallada del trabajo realizado alrededor del desarrollo de un sistema software, el cual implementa la propuesta del proyecto. En la descripción se abarca la arquitectura definida para soportar la construcción del prototipo, los modelos de caso de uso del sistema, detalles de implementación y las pruebas realizadas para la evaluación del mismo, junto con los resultados obtenidos.

Capítulo 6: Conclusiones, Contribuciones y Trabajo Futuro

Finalmente, se presentan las conclusiones a las cuales se llegó en el desarrollo del presente trabajo de grado, las principales contribuciones de la ejecución del proyecto y se plantean diferentes ideas propuestas para la realización de trabajos futuros.

Capítulo 2

2. Estado Actual del Conocimiento

En este capítulo se presentan las bases conceptuales, definiciones formales y el estado actual de los trabajos que abarcan temas relacionados con este proyecto. Inicialmente, se describen los conceptos claves en los que se fundamenta la propuesta para el monitoreo de servicios convergentes. Posteriormente, se describen los principales trabajos relacionados, con el fin de resaltar los avances y las brechas en cada uno de ellos. Finalmente, se presenta un resumen describiendo los aportes más importantes del capítulo.

2.1 Base Conceptual

2.1.1 Servicios Convergentes

Actualmente existe una tendencia que ha generado un escenario convergente en el cual se ha definido un nuevo modelo conocido como Telco 2.0 [1], en él se relacionan conceptos, servicios y tecnologías que brinda la Web 2.0 con los servicios del dominio de las telecomunicaciones. Este nuevo paradigma brinda una posibilidad para la creación de un nuevo tipo de servicios denominados servicios convergentes.

El dominio de los servicios web cuenta con características y componentes importantes y maduros entre los que se encuentra la composición de servicios mediante el uso de estándares como WS-BPEL [14], que permite el desarrollo de aplicaciones complejas partiendo de servicios web específicos y modulares. Esto a su vez posibilita el re-uso del software y facilita la integración de los servicios en diversos entornos empresariales.

Por su parte, en el domino de las telecomunicaciones existen especificaciones como IMS (IP Multimedia Subsystem) [15] o EPS (Evolved Packet System) [16], cuya arquitectura junto a los SIP Servlets posibilita la composición de aplicaciones SIP individuales para obtener completos servicios integrados, los cuales permiten obtener beneficios en el despliegue y ejecución de los servicios de telecomunicaciones [8].

De esta forma, un servicio convergente se entiende como la coordinación de un conjunto de servicios de diferentes proveedores, tales que, a la vista del usuario final sea un solo servicio [17]. Este tipo de servicios buscan integrar tanto el dominio web como el de las telecomunicaciones, con el objetivo de que los proveedores puedan ofrecer al usuario servicios cada vez más completos, innovadores y con una mayor funcionalidad, en un corto tiempo de salida al mercado (*Time-to-Market*).

2.1.2 Especificación JAIN SLEE

La especificación JAIN SLEE (*Java APIs for Integrated Networks Service Logic Execution Environment*) [5] define un entorno estándar para la ejecución de lógica de servicios y especifica la manera en la cual, servicios de telecomunicaciones portables y de alta calidad, pueden ser construidos, gestionados y ejecutados. La especificación JAIN SLEE ha sido diseñada para soportar servicios de telecomunicaciones, cumpliendo con los requerimientos que exigen este tipo de servicios como: baja latencia, alta disponibilidad, orientación a eventos, etc [18].

JAIN SLEE provee un modelo de programación estándar que puede ser usado por la amplia comunidad de desarrolladores Java. El modelo de programación ha sido diseñado para simplificar el trabajo de aplicación del desarrollador, eliminar los errores de programación y asegurar que servicios altamente robustos puedan ser desarrollados en poco tiempo. Adicionalmente, el lenguaje Java facilita que las aplicaciones puedan ser desarrolladas una vez y luego desplegadas en cualquier entorno que implemente dicha especificación, independientemente del SLEE [18].

La especificación JAIN SLEE define una arquitectura en varias capas como se observa en la figura 1-1, la cual se compone de una capa central conocida como el modelo de componentes en donde se especifica cómo la lógica del servicio debe ser construida y empaquetada, cómo debe procesar eventos y cómo debe ser ejecutada [18].

La capa de gestión del SLEE especifica el mecanismo por el cual un administrador gestiona el SLEE (incluyendo subscripción, instalación de servicios, ciclo de vida del sistema, etc.) y permite a los desarrolladores de servicios definir los datos necesarios para un servicio en particular. Define también la gestión de especificaciones de perfil de servicios y de usuarios pertenecientes al SLEE [18].

La capa del sistema de componentes del SLEE define el enrutamiento de eventos tanto externos como eventos originados desde el SLEE y proporciona componentes comunes a todo el SLEE que pueden ser utilizados por los desarrolladores de servicios como herramientas de funcionamiento para las aplicaciones [18].

Los adaptadores de recursos son responsables de la comunicación con un recurso particular que es externo al modelo de programación de JAIN SLEE, comunicándolo con la lógica de enrutamiento de eventos en la implementación del SLEE y proporcionando al desarrollador de servicios las interfaces necesarias para hacer uso de las diferentes capacidades que proporciona una adaptador en particular [18]

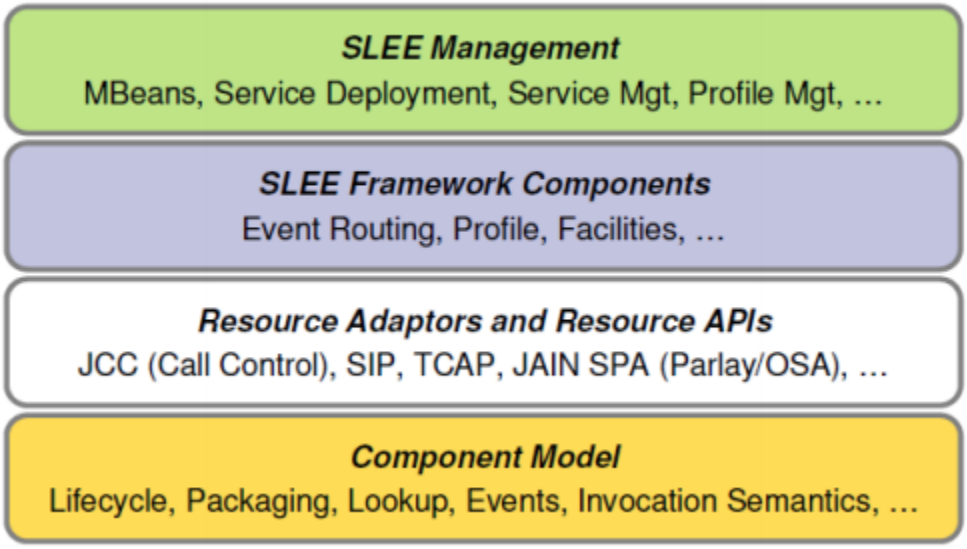

Figura 1-1. Arquitectura en Capas de JAIN SLEE (Adaptación de [18]).

2.1.2.1 Conceptos Fundamentales de JAIN SLEE

- ✓ **Perfil:** un perfil de JAIN SLEE contiene información acerca de un servicio o de un subscriptor. Los bloques de construcción de servicio que se ejecutan dentro de un entorno JAIN SLEE pueden acceder a la información contenida dentro de los perfiles como parte de su lógica de aplicación [19].

- ✓ **Bloque de Construcción de Servicio:** el elemento de reutilización definido por JAIN SLEE es el bloque de construcción de servicio (SBB). Un SBB es un componente software que envía y recibe eventos, y también ejecuta lógica computacional basada en la recepción de eventos y su estado actual. El código de programación de un SBB está compuesto de clases Java [19].

- ✓ **Evento:** un evento representa un acontecimiento que puede requerir procesamiento por parte de alguna aplicación. Un evento se puede originar de 11 diferentes fuentes, por ejemplo, una pila de protocolos externa o desde el interior del SLEE [19].

- ✓ **Recursos y Adaptadores de Recursos:** los recursos son entidades externas que interactúan con otros sistemas fuera del SLEE, tal como elementos de red (HLR, MSC, etc.), pilas de protocolos, directorios y bases de datos. Un adaptador de recursos implementa la interfaz de un recurso dentro del entorno JAIN SLEE [19].

2.1.2.2 JAIN SLEE *Facilities*

Los *Facilities* son una serie de siete componentes funcionales estándar proporcionados por la especificación JAIN SLEE, los cuales cumplen diferentes tareas: T*imer Facility, Alarm Facility, Trace Facility, Activity Context Name Facility, Profile Facility, Event Lookup Facility, Service Lookup Facility* [5]*.* Para el presente trabajo de grado, representan gran importancia el *Timer Facility* y el *Alarm Facility,* los cuales se describen de manera general a continuación:

- ✓ ***Timer Facility:*** los Bloques de Construcción de Servicio (SBB) a menudo necesitan llevar a cabo acciones periódicas o iniciar acciones que un tiempo después deben ser comprobadas para asegurarse que han sido completadas de manera exitosa. JAIN SLEE provee dicha funcionalidad mediante el *Timer Facility,* este componente gestiona una serie de contadores de tiempo, cada uno de los cuales es independiente y puede disparar una serie de eventos temporizadores [5].

- ✓ ***Alarm Facility*****:** JAIN SLEE provee además un componente con fines de monitoreo conocido como *Alarm Facility*, el cual permite el establecimiento de Alarmas que son usadas por los Bloques de Construcción de Servicio y por los Adaptadores de Recursos para generar notificaciones ante un posible fallo, con el fin de que estas sean manejadas desde un cliente de gestión externo al SLEE, el cual debe registrar las notificaciones generadas por el *Alarm Facility*. Cada Alarma tiene asociados argumentos como el tipo de alarma, la fuente de origen, el nivel de la alarma y un mensaje, todos definidos en el momento en que la Alarma es establecida [5].

2.1.3 Calidad de Servicio (QoS)

La Unión Internacional de Telecomunicaciones (ITU) define la QoS como “La totalidad de las características de un servicio de telecomunicaciones que determinan su capacidad para satisfacer las necesidades explícitas e implícitas del usuario del servicio” [20]. A nivel de servicios web, el Consorcio World Wide Web (W3C) se refiere a los requerimientos de QoS como “Los aspectos de calidad de un servicio web”, estos pueden incluir el rendimiento, fiabilidad, escalabilidad, disponibilidad, seguridad, exactitud, entre otros [21].

La QoS comprende tanto la calidad de funcionamiento de la red donde opera el servicio como la calidad de funcionamiento independiente de la red. Dentro de los parámetros de funcionamiento de la red se incluyen la tasa de errores en los bits, la latencia, el ancho de banda, etc., mientras que en la calidad de funcionamiento independiente de la red se encuentran los parámetros asociados directamente al servicio a nivel de aplicación [20].

Existen además dos tipos de requerimientos de QoS: funcionales y no funcionales. Los requerimientos funcionales especifican el comportamiento o las funciones que el servicio debe ser capaz de realizar. Por otra parte, los requerimientos no funcionales especifican cómo un servicio realiza las funciones, estos últimos están principalmente asociados a la calidad y restricciones durante el tiempo de ejecución de un servicio, como por ejemplo el tiempo de respuesta, disponibilidad o precisión [22].

Es importante aclarar que el presente trabajo se centrará en el análisis de parámetros no funcionales de QoS a nivel de servicio en la capa de aplicación; durante la ejecución de servicios convergentes.

2.1.4 Monitoreo de Servicios

Cuando se habla de monitoreo de servicios, se pueden distinguir dos aspectos principales. El primero está relacionado con eventos asociados al desempeño de la red, mientras que el segundo hace referencia a eventos centrados directamente en los servicios, refiriéndose a un amplio rango de características no funcionales asociadas a estos [23].

Para el primer aspecto, la Unión Internacional de Telecomunicaciones (ITU) ha definido una serie de recomendaciones que especifican los mecanismos que se deben tener en cuenta para realizar una gestión eficiente y correcta de los sistemas de telecomunicaciones. Específicamente, la recomendación ITU-T M.3400 [24] proporciona las funciones de gestión para una Red de Gestión de Telecomunicaciones (TMN). Dentro de estas se encuentran las funciones de gestión de desempeño (*Performance Management)*, las cuales están asociadas directamente con el monitoreo, proporcionando funciones para evaluar y reportar el comportamiento y efectividad de los equipos y redes de telecomunicaciones mediante la recolección y análisis de datos, tanto de la red como de los elementos que la componen [24].

En cuanto al segundo aspecto, debido a la creciente complejidad de los servicios y de los sistemas que soportan la ejecución de estos, el monitoreo es utilizado para observar continuamente el comportamiento de dichos servicios y sistemas. Así mismo, la recolección de datos durante el tiempo de ejecución, con el fin de generar estadísticas acerca del comportamiento de los servicios, se presenta como un factor importante, debido a que en el momento de un fallo, se tiene información más precisa que permite tener un sistema de gestión y corrección de fallos más eficiente, lo cual toma gran importancia en el momento de garantizar niveles de QoS [23].

A nivel de servicios, también se realiza el monitoreo bajo otro propósitos, entre ellos se encuentra el monitoreo de los Acuerdos de Nivel de Servicio (SLA), el cual se realiza en tiempo de creación, con el fin de controlar que los acuerdos entre proveedores y usuarios del servicio, se lleven a cabo bajo determinados parámetros de QoS [23].

2.2 Trabajos Relacionados

Actualmente, un número importante de trabajos enfocan sus objetivos en el monitoreo de servicios, principalmente centrados en el dominio de la Web. Así mismo, se marca una tendencia actual hacía técnicas y políticas de monitoreo de servicios compuestos, realizadas en tiempo de ejecución y basadas en métricas de QoS (Calidad de Servicio) y aspectos relacionados con los SLA (Acuerdos de Nivel de Servicio).

2.2.1 Modelos para el monitoreo de servicios

En [22] se presenta un estudio acerca de distintos modelos de monitoreo, con el objetivo de proponer una solución para monitorear requerimientos no funcionales de QoS (Calidad de Servicio) en servicios web, mediante el uso de lenguaje natural en los SLA. Es decir, un modelo en donde los usuarios pueden especificar los parámetros de QoS con palabras como “alta disponibilidad” o “bajo tiempo de respuesta”. Dicho modelo también plantea el uso de técnicas de Inteligencia Artificial (AI), las cuales le permitan monitorear tanto

servicios simples como compuestos. Definen además, que el prototipo de prueba a implementar, estará basado en técnicas de lógica difusa y se enfocará en tres parámetros de QoS: disponibilidad, tiempo de respuesta y rendimiento. Un aporte importante de este trabajo es que se realiza una aproximación a un modelo para el monitoreo de parámetros de QoS, lo cual puede establecer una base en cuanto al desarrollo de un modelo de monitoreo de parámetros de QoS para servicios convergentes.

Por otra parte, en [25] se muestra el análisis de una serie de herramientas comerciales para el monitoreo de servicios web en términos de los SLA, partiendo del desarrollo de un modelo basado en el estándar de calidad ISO/IEC 9126-1 [26], mediante el cual se busca proveer información que pueda contribuir a la selección de una herramienta de monitoreo adecuada de acuerdo a una situación particular, analizando las características que las herramientas de monitoreo ofrecen y los atributos de calidad esenciales que debe contener una herramienta de monitoreo. Las características consideradas para la evaluación de las herramientas de monitoreo son: funcionalidad, confiabilidad, usabilidad, eficiencia, facilidad de mantenimiento y portabilidad, de estas se desprenden una serie de sub-características que a la vez contienen los atributos usados para el monitoreo de los servicios en las distintas herramientas. Finalmente se presenta una evaluación de las herramientas analizadas frente a diversos parámetros y métricas haciendo uso del *"Individual Quality Model Construction",* un método para la construcción de un modelo de calidad para herramientas de monitoreo de servicios web que está basado en el estándar ISO/IEC 9126-1. El principal aporte de este trabajo es que establece una base sólida en cuanto a los criterios para determinar los parámetros de calidad requeridos de acuerdo a un estándar de calidad, pudiendo considerarse como un punto de partida para el análisis de los parámetros de QoS a tener en cuenta durante la definición de métricas para el monitoreo de servicios convergentes.

2.2.2 Enfoque Orientado a Aspectos para el Monitoreo Dinámico de un Entorno de Ejecución de Lógicas de Servicio (SLEE)

En [27] implementan un framework de registro que es capaz de insertar código de monitoreo en una aplicación JAIN SLEE. Para lograr este propósito hacen uso de una implementación del framework AOP llamada JBoss-AOP. Este framework permite interceptar un método en el momento en que este es invocado, e insertar en él código adicional de manera transparente. Para ello, es definido un archivo de tipo XML (*eXtensible Markup Language*) en el cual se fija el número máximo de puntos de unión, los cuales definen los puntos dónde puede ocurrir una inserción de código. Este trabajo presenta una base importante debido a que define un método para la inserción de código con fines monitoreo en tiempo de ejecución sobre aplicaciones basadas en JAIN SLEE. Así mismo, AOP (*Aspect-Oriented Programming*) constituye una herramienta importante para servidores de aplicaciones tipo SLEE, dado que permite implementar en diferentes contenedores aplicaciones de monitoreo, pruebas, rastreo de niveles de servicio, recolección de estadísticas, entre otras. El aporte más sobresaliente de este trabajo es que se utiliza JBoss-AOP como una opción para introducir código en aplicaciones desplegadas en entornos JAIN SLEE, siendo una alternativa a considerar para lograr el monitoreo de servicios convergentes en tiempo de ejecución.

2.2.3 Monitoreo en Tiempo de Ejecución de Servicios Web Implementados en BPEL

En [28] se propone una arquitectura de monitoreo que busca separar de forma clara, la lógica de negocio del servicio web de las funcionalidades asociadas al monitoreo. Para ello, introducen un Agente de Monitoreo dentro de la arquitectura tradicional de los servicios web, con el fin de recoger y administrar información sobre valores de QoS en tiempo de ejecución. Dicho agente de monitoreo es implementado como un servicio web. Primero, se describe el flujo de eventos en BPEL (*Business Process Excecution Language*) y luego, mediante el uso de AOP, se realiza la interceptación de mensajes SOAP (*Simple Object Access Protocol*). La arquitectura define dos componentes principales del Agente de Monitoreo: El Monitor de Servicio, que se encarga de capturar la información en el momento en que el servicio es invocado; y el Colector de Resultados, el cual toma la información que ha sido recogida, la procesa y la almacena en una base de datos. Se definen además tres métricas de QoS: tiempo de respuesta, disponibilidad y precisión.

Para la experimentación del modelo propuesto utilizan una herramienta comercial para la ejecución de pruebas de desempeño llamada Mercury Load Runner, la cual crea escenarios del mundo real para simular el comportamiento de un sistema. De esta forma, efectúan una comparación de tiempos de ejecución de servicios web con y sin el Agente de Monitoreo implementado. Con ello logran concluir que la diferencia de tiempos no difiere en un porcentaje mayor al 1%. Este trabajo presenta un enfoque importante y realizable de monitoreo de servicios web en tiempo real, resaltando como principal aporte el método utilizado para la recolección de datos durante la ejecución de los servicios con el fin de realizar un monitoreo exhaustivo de los mismos.

2.2.4 Monitoreo Automático de Acuerdos de Nivel de Servicio de Servicios Web

En este trabajo [29], diseñan y desarrollan una arquitectura de monitoreo en tiempo de ejecución implementada en Orc[2], donde el motor del servicio web y las herramientas de monitoreo trabajan de forma concurrente accediendo a todo el intercambio de mensajes, sin afectar el funcionamiento normal de la aplicación que está siendo monitoreada. Bajo este propósito, mediante los SLA se especifican las propiedades funcionales que serán monitoreadas, donde una vez definidas, se generan unos actores claves llamados Observadores, los cuales son integrados de forma automática al sistema implementado en Orc. El método utilizado para crear dichos Observadores consiste en generar una Máquina de Mealy que se encarga de observar que el envío de mensajes y eventos sea apropiado, seguido a esto, agregan funcionalidades externas con el fin de cuantificar valores de QoS. Finalmente integran la Máquina de Mealy con el motor del servicio web.

En este trabajo se toma un caso estudio del proceso de pago de una factura en el cual intervienen un cliente y el proveedor del servicio. Primero se modela el proceso que se debe llevar a cabo para realizar la operación mediante un diagrama de flujo, posteriormente, se definen dos puntos clave donde el monitoreo es requerido por el motor del servicio web. Cada uno de esos puntos emite una alarma ante una posible violación de lo establecido en el SLA, en este caso teniendo en cuenta parámetros de tiempo de respuesta y de confiabilidad. Este trabajo define además una arquitectura síncrona, la

[2] Lenguaje de programación concurrente que proporciona acceso uniforme a los servicios computacionales.

cual trabaja de manera conjunta con el motor del servicio web, sin que este se vea afectado. El principal aporte de este trabajo es que define una arquitectura síncrona la cual funciona de manera conjunta con el motor del servicio web, realizando todo el procesamiento de información para el monitoreo, de forma silenciosa sin afectar los servicios analizados.

2.2.5 Monitoreo del Entorno de Red Soportado en la Adaptación de Servicios Compuestos

Este trabajo [30], plantea una adaptación de servicios compuestos que soporta un sistema de monitoreo del entorno de red, en el cual su principal función es monitorear el sistema de tal forma que este cumpla los tiempos de ejecución de un servicio compuesto establecidos en los SLA y que además, proporcione información acerca de la desviación de tiempos de ejecución causados por los cambios en el entorno de red. Para este fin, los autores definen tres pasos, primero realizan la recolección de información acerca del entorno, segundo perciben los cambios en dicho entorno, y por último, calculan los cambios en el servicio. Por otra parte, basados en diez posibles eventos asociados al entorno de red, proponen siete reglas de fallos, a partir de las cuales, es posible construir un algoritmo de reconocimiento de fallos. Finalmente realizan una experimentación del modelo planteado en el que ejecutan un servicio llamado Tilt2 y el sistema confirma que ante una posible falla en el entorno de red, se ve afectado el tiempo de ejecución del servicio, de esta forma tras realizar este proceso de monitoreo, el sistema envía dicha información al motor de ejecución para que se encargue de realizar una acción basada en la adaptación de servicios compuestos. El aporte más sobresaliente de este trabajo es que se enfoca en monitorear los servicios teniendo en cuenta el entorno del servicio compuesto, lo cual es un factor importante a considerar en la dinámica del monitoreo para servicios convergentes.

2.3 Brechas del Conocimiento

De acuerdo al análisis realizado sobre los trabajos existentes, la principal brecha es el dominio en el cual estos se encuentran enfocados, puesto que la gran mayoría buscan realizar el monitoreo de servicios exclusivos del dominio de la Web. De este modo, las arquitecturas y esquemas planteados no pueden ser aplicados al monitoreo de servicios convergentes, dado que tanto las plataformas de ejecución como las funcionalidades propias de cada dominio, presentan características de funcionamiento diferentes. Por otra parte, los trabajos enfocados en un entorno convergente, no han desarrollado algún prototipo funcional que permita comprobar el funcionamiento de estos.

A continuación se muestra una tabla que contiene las brechas existentes en cada uno de los trabajos descritos anteriormente.

Trabajo	Brecha del Conocimiento
[22]	Sí bien este trabajo busca proponer un modelo muy completo para el monitoreo de características no funcionales en los Servicios Web, simplemente se limitan a citar la literatura existente en la cual se pueden basar para lograr dicho objetivo. No realizan la implementación de dicho modelo. Por otra parte, este trabajo se enfoca sólo en el domino de la Web.
[25]	El trabajo está enfocado al análisis de herramientas comerciales, además, no se usaron licencias completas en todos los casos. Al no tener en cuenta herramientas académicas o bajo licencia libre, es posible que se estén ignorando modelos, parámetros y características que pueden ser útiles en la construcción de un modelo de calidad unificado que pueda servir para cualquier entorno, sea académico o comercial.
[27]	En este trabajo se define un método para la inserción de código con fines de monitoreo en tiempo de ejecución sobre aplicaciones basadas en JAIN SLEE, pero no se realiza una implementación ni se define una arquitectura o un esquema para la realización del monitoreo como tal. Por otra parte, tampoco se definen los parámetros de QoS que pueden ser monitoreados bajo el enfoque presentado.
[28]	Aunque en este trabajo se define un modelo de QoS basado en tres métricas (tiempo de respuesta, disponibilidad y precisión); en el momento de realizar la evaluación del sistema, solo se muestran resultados del tiempo de ejecución de los servicios web, dejando de lado la detección de fallos en las métricas mencionadas anteriormente. Así mismo, la herramienta que se usa para la experimentación del modelo propuesto es de carácter propietario. Por otra parte, este trabajo se encuentra centrado exclusivamente en servicios de la Web.
[29]	En este trabajo se realizan aportes importantes en el monitoreo de parámetros de QoS como el tiempo de respuesta o la confiabilidad, pero con el enfoque presentado solo es posible aplicar el monitoreo a un servicio web, dejando a un lado su aplicación al monitoreo de servicios compuestos que hacen uso de múltiples instancias. Así mismo, este trabajo está centrado únicamente en el dominio de la Web.
[30]	En este trabajo se define un algoritmo de reconocimiento de fallos, sin embargo, este se basa únicamente en eventos asociados al entorno de red, dejando de lado de lado parámetros no funcionales de QoS a nivel de aplicación asociados al monitoreo de servicios como son la disponibilidad, confiabilidad, precisión, entre otros.

Tabla 2-1. Brechas de los trabajos relacionados.

RESUMEN

Este capítulo presentó el estado actual del conocimiento, sobre el cual se encuentra enmarcado el presente trabajo de grado, mediante la descripción de los conceptos y tecnologías relacionados con el monitoreo de servicios convergentes en entornos JAIN SLEE, tales como la Calidad de Servicio (QoS), Monitoreo de Servicios y la especificación JAIN SLEE; además se realizó un análisis de los trabajos relacionados con el monitoreo de servicios, destacando sus principales aportes; finalmente se realizó una tabla donde se enumeran las brechas existentes para cada uno de estos.

Capítulo 3

3. Definición de métricas y reglas para el monitoreo de servicios convergentes

En este capítulo se presentan los conceptos generales y el estudio realizado para la definición y proposición de un conjunto de métricas y reglas que pueden utilizarse en el monitoreo de servicios convergentes. Inicialmente se realiza un análisis de los diferentes modelos y herramientas existentes para el monitoreo de servicios en el dominio de la Web, de tal forma que sea posible realizar una evaluación que arroje resultados que permitan seleccionar un conjunto de parámetros no funcionales de calidad de servicio que se adecuen a los requerimientos de los servicios convergentes. Posteriormente, se procede a definir un conjunto de métricas de QoS para los servicios atómicos que componen a los servicios convergentes y una serie de reglas para estos últimos, tomando como base los parámetros definidos en la evaluación realizada con anterioridad.

3.1 Evaluación de herramientas para el monitoreo de servicios

Basados en la búsqueda de enfoques vigentes que proponen herramientas para el monitoreo de servicios web y teniendo en cuenta la funcionalidad y concordancia en el desarrollo de estos, se han seleccionado 10 herramientas como base para realizar la evaluación que permita determinar las métricas que se van a usar en el monitoreo de servicios convergentes, dichas herramientas se mencionan de manera general a continuación, mientras que su análisis y descripción detallada se encuentra en el Anexo B de la presente monografía.

Li et al. [30] propone un modelo para el monitoreo del entorno de red que soporta la composición de servicios compuestos, en el cual la principal función es monitorear el sistema, de tal forma que este cumpla con los mínimos establecidos para el tiempo de ejecución de un servicio compuesto, los cuales son previamente establecidos en los SLA. Moser et al. [31] presenta un sistema de monitoreo llamado *VieDAME*, enfocado en el monitoreo de procesos BPEL de acuerdo a atributos de QoS, para ello, este sistema intercepta mensajes SOAP (*Simple Object Access Protocol*) dentro de BPEL en tiempo de ejecución. Otra enfoque es presentado por Oriol et al. [32], donde se introduce un sistema denominado *SALMon,* el cual permite monitorear aplicaciones web con el fin de detectar violaciones en los SLA. Hasan et al. [22] propone un modelo para monitorear el cumplimento de requerimientos no funcionales de QoS especificados mediante lenguaje natural en los SLA. De igual manera, Haiteng et al. [28] propone una arquitectura de monitoreo de parámetros de QoS, donde se separa de manera clara la lógica del negocio de los servicios web de las funcionalidades asociadas al monitoreo. Goel et al. [29] desarrolla una arquitectura de monitoreo en tiempo de ejecución donde el motor del servicio web y las herramientas de monitoreo trabajan de forma concurrente accediendo a todo el intercambio de mensajes, sin afectar el funcionamiento normal de la aplicación que está siendo monitoreada, en este sistema las propiedades de QoS que serán monitoreadas son definidas a través de los SLA. El enfoque propuesto por Sun et al. [33]

propone un framework para el monitoreo de servicios web basado en políticas que expresan los requerimientos de los servicios que serán monitoreados, mientras que las tareas de monitoreo son implementadas usando AOP (*Aspect-Oriented Programming*). Artaiam et al. [34] por su parte, desarrolla una herramienta para el monitoreo de servicios web, en forma de extensión, que se ejecuta sobre el servidor de aplicaciones GlassFish, y además propone un modelo de QoS que incluye diferentes parámetros para mejorar las tareas de monitoreo del lado del servidor. Raimondi et al. [35] presenta una metodología para el monitoreo de especificaciones no funcionales de los servicios web, de tal forma que se detecten violaciones de estas en tiempo de ejecución. Finalmente, Halima et al. [36] propone un framework para el monitoreo y análisis de servicios web, basado en métricas de QoS mediante la interceptación de mensajes SOAP, donde implementan un algoritmo que detecta la degradación de los parámetros de QoS.

3.1.1 Criterios para la evaluación de herramientas

Para realizar el análisis de las diferentes herramientas que abordan la problemática del monitoreo de servicios, se definen tres criterios basados en recomendaciones dadas en [25], las cuáles están relacionadas con la evaluación y análisis de herramientas para el monitoreo de servicios web. Con base en lo anterior, los criterios que se tendrán en cuenta para el análisis son: parámetros de Calidad de Servicio (QoS), tipo de servicios monitoreados y el tipo de implementación, los cuales se describen a continuación:

3.1.1.1 Parámetros de Calidad de Servicio (QoS)

Los servicios convergentes involucran el dominio de la web y de las telecomunicaciones, de esta forma, tanto el estándar ISO/IEC 9126 [26] como la especificación ITU-T M.3400 [24] son dos referentes que proponen parámetros para evaluar el rendimiento y la calidad de los componentes de un sistema tanto desde la perspectiva software como desde la perspectiva de la gestión de una red de telecomunicaciones.

Para efectos de la evaluación, en primer lugar es necesario obtener una serie de parámetros de QoS que puedan ser aplicados a nivel de servicio de acuerdo a la concurrencia y relevancia que tengan los mismos en [26] y [24]. De esta forma, se escogen los siguientes ocho parámetros que pueden ser aplicables a servicios convergentes: Disponibilidad, Tiempo de Respuesta, Tiempo de Ejecución, Throughput, Precisión, Fiabilidad, Uso de Recursos y Latencia.

Ahora con base en definiciones formales, se propone una definición contextualizada en el monitoreo de servicios para cada uno de ellos:

- ✓ **Disponibilidad:** la disponibilidad es el aspecto de calidad que determina si un servicio está presente o listo para su uso inmediato. Representa la probabilidad de que un servicio esté disponible. Valores altos de disponibilidad significan que el servicio está siempre listo para su uso, mientras que valores bajos indican imprevisibilidad sobre sí el servicio estará disponible en un determinado tiempo.

- ✓ **Tiempo de Respuesta:** el tiempo de respuesta es un atributo que se refiere al tiempo transcurrido entre el envío de una petición del servicio y la terminación de la respuesta del servicio. Usualmente es medido en milisegundos.

- ✓ **Tiempo de Ejecución:** el tiempo de ejecución es definido como el tiempo transcurrido para procesar una petición por parte del proveedor del servicio.

- ✓ **Throughput:** el throughput está definido como la cantidad de peticiones que pueden ser procesadas por parte de un servicio dentro de un periodo específico de tiempo.

- ✓ **Precisión:** la precisión está definida como la tasa de error producida por un servicio, es calculada evaluando todas las invocaciones desde un punto dado.

- ✓ **Fiabilidad:** la fiabilidad es el aspecto de calidad que representa la capacidad de un servicio para mantener el servicio y la calidad de servicio. Se puede representar como la probabilidad de que la petición de un servicio sea respondida de manera correcta dentro de un periodo de tiempo esperado.

- ✓ **Uso de Recursos:** el uso de recursos representa la cantidad de recursos utilizados para el envío de una petición del servicio y la recepción de su respectiva respuesta.

- ✓ **Latencia:** la latencia hace referencia al tiempo de retraso entre el momento en que se realiza una petición del servicio y el tiempo estimado en que debería llegar la respuesta.

Una vez establecidos los parámetros a evaluar, se debe determinar un peso para cada uno de ellos, el cual estará definido a partir de ahora como P_{par}.

Para calcular el valor de P_{par} se tienen en cuenta dos criterios:

- ✓ El número de veces que dicho parámetro está presente en los modelos analizados, definido a partir de ahora como N_v, el cual tendrá un valor de 1 para un parámetro presente en las 10 herramientas analizadas y un valor de 0 para un parámetro que no es tenido en cuenta en ningún enfoque. La función que describe a N_v es la siguiente:

$$Nv = \frac{\#\ veces\ presente}{\#\ modelos} \quad (1)$$

- ✓ La conveniencia de aplicar un determinado parámetro en el monitoreo de servicios convergentes, definido a partir de ahora como C_{sc}, el cual tendrá un valor de 1 para un parámetro cuyo monitoreo en servicios convergentes sea altamente conveniente y 0 para un parámetros que no presente relevancia en el monitoreo de servicios convergentes. El valor de C_{sc} será otorgado de manera arbitraria de acuerdo a lo mencionado anteriormente.

  Luego, el peso de cada parámetro P_{par} será el promedio de los valores de N_v y C_{sc}, así:

$$P_{par} = \frac{Nv + Csc}{2} \quad (2)$$

3.1.1.2 Tipo de servicios monitoreados

El tipo de servicios monitoreados hace referencia a sí la herramienta que está siendo evaluada puede ser aplicable a servicios web compuestos o sólo a servicios web tradicionales. A continuación se describen estos dos tipos de servicios:

- ✓ **Servicio Web:** es un sistema software con interfaces y conexiones públicas definidas y descritas en XML. Puede ser descubierto por otros servicios a través de un Identificador de Recurso Uniforme (URI, Uniform Resource Identifier) e interactuar entre ellos de una manera prescrita por su definición y usando mensajes basados en XML transmitidos por protocolos de Internet [37].

- ✓ **Servicio Web Compuesto:** es un servicio web cuyas operaciones son interfaces a operaciones proporcionadas por otros servicios web conocidos como Servicios Web Atómicos, de tal manera que cada uno de sus componentes pueden ser conectados entre sí para construir nuevos servicios. Se debe resaltar que cada Servicio Web Atómico puede proporcionar solo un subconjunto de las operaciones requeridas por el Servicio Web Compuesto.

  En general, cuando una aplicación cliente invoca operaciones de un servicio web compuesto, éste no ejecuta esas operaciones por sí mismo, en cambio, se comporta como un mediador que delega las operaciones a algunos servicios web atómicos. La Figura 3-1 muestra un escenario de ejemplo de invocación de operaciones entre la aplicación cliente, el servicio web compuesto y los servicios web atómicos. Cuando la operación A en el servicio Compuesto es invocada por la aplicación cliente, tanto WS1.A o WS3.A pueden ser seleccionados para la ejecución.

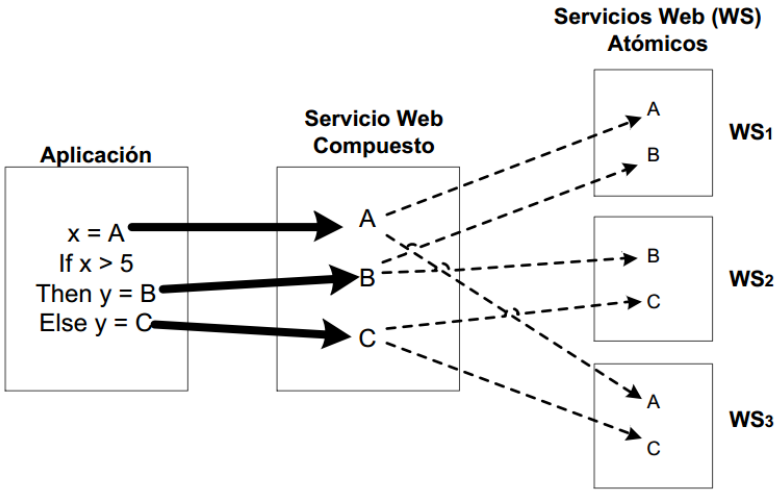

Figura 3-1. Servicio Web Compuesto

El criterio del tipo de servicios monitoreados estará definido como T_{sm} y tendrá 3 valores posibles: 0 sí el modelo no especifica de forma clara qué tipo de servicios se monitorea, 0.5 sí la herramienta se limita al monitoreo de servicios web simples y 1 sí soporta el monitoreo de servicios web compuestos.

3.1.1.3 Tipo de implementación

El criterio del tipo de implementación de las herramientas analizadas revela sí el enfoque propuesto se desarrolla y se ejecuta en un entorno de prueba. Estará definido a partir de ahora como T_{imp} y al igual que el criterio anterior, este tendrá 3 valores posibles: 0 sí el modelo propuesto se limita a plantear una posible herramienta y no es implementado ni ejecutado en un caso de estudio, 0.5 sí la herramienta se implementa pero no realiza una experimentación del prototipo y 1 sí se implementa y además se ejecuta en un caso de estudio donde se comprueba su correcto funcionamiento.

3.1.2 Evaluación de herramientas

Con base en los 3 criterios definidos en la sección anterior (P_{par}, T_{sm}, T_{imp}) se procede a evaluar cada herramienta analizada, otorgando un porcentaje de la evaluación total a cada parámetro, teniendo en cuenta la relevancia de estos en nuestro trabajo, de la siguiente manera:

Para los propósitos del presente trabajo los parámetros de QoS tenidos en cuenta por cada modelo es el criterio más relevante, de esta forma tendrá un porcentaje del 60% en la evaluación final de cada modelo, mientras que los criterios del tipo de servicios monitoreados y el tipo de implementación tendrán un porcentaje del 20% cada uno.

Cada herramienta tendrá una calificación con un valor comprendido entre 0 y 1, la cual será 1 para un enfoque que mejor se adapta al monitoreo de servicios convergentes y 0 un enfoque que definitivamente no es relevante en el monitoreo de servicios convergentes.

Inicialmente es necesario hallar el valor del peso de cada parámetro P_{par}, para ello primero se debe conocer el valor de N_v de cada uno, la tabla 3-1 muestra el número de veces que cada parámetro está presente en los enfoques analizados.

Enfoque	**Parámetros**							
	D	**TR**	**TE**	**T**	**P**	**F**	**UR**	**L**
1. Li et al. [30]	X	X		X			X	X
2. Moser et al. [31]	X	X			X			
3. Oriol et al. [32]	X	X	X		X			
4. Hasan et al. [22]	X	X		X				
5. Haiteng et al. [28]	X	X			X			
6. Goel et al. [29]	X	X				X		
7. Sun et al. [33]		X				X		
8. Artaiam et al. [34]	X					X		
9. Raimondi et al. [35]				X		X		X
10. Halima et al. [36]	X	X	X	X				
Total	**9**	**9**	**2**	**5**	**4**	**5**	**2**	**2**

Tabla 3-1. Presencia de los parámetros en cada herramienta.

Donde D: Disponibilidad, TR: Tiempo de Respuesta, TE: Tiempo de Ejecución, T: Throughput, P: Precisión, F: Fiabilidad, UR: Uso de Recursos, L: Latencia.

De esta forma, en la Tabla 3-2 se muestra el valor de N_v obtenido para los ocho parámetros mediante la fórmula (1), además del valor de C_{sc} establecido para cada uno de ellos.

Atributo	N_v	C_{sc}
Disponibilidad	0.90	0.90
Tiempo de Respuesta	0.90	0.90
Tiempo de Ejecución	0.20	0.45
Throughput	0.50	0.45
Precisión	0.40	0.70
Fiabilidad	0.50	0.70
Uso de Recursos	0.20	0.45
Latencia	0.20	0.25

Tabla 3-2. Valor de N_v para cada parámetro de QoS.

Posteriormente, mediante la fórmula (2) se calcula el valor correspondiente al peso de cada parámetro, los resultados se muestran en la tabla 3-3.

Parámetro	P_{par}
Disponibilidad	0.90
Tiempo de Respuesta	0.90
Tiempo de Ejecución	0.32
Throughput	0.47
Precisión	0.55
Fiabilidad	0.60
Uso de Recursos	0.32
Latencia	0.22

Tabla 3-3. Peso de cada parámetro de QoS.

Una vez obtenido el valor de P_{par} para cada parámetro de QoS, es necesario calcular un Peso Total de los parámetros (PT_{par}) para cada herramienta analizada, valor que como se mencionó anteriormente, corresponde al 60% de la calificación final. Para este fin, se tendrán en cuenta sólo los 4 parámetros con mayor peso mostrados en la Tabla 3-3: Disponibilidad, Tiempo de Respuesta, Fiabilidad y Precisión.

De esta forma, PT_{par} tendrá cuatro valores posibles: 0.25, 0.50, 0.75 y 1, sí el modelo tiene en cuenta 1, 2, 3 o 4 parámetros respectivamente.

La tabla 3-4 muestra el valor de PT_{par} para cada uno de los enfoques analizados.

Enfoque	Parámetros				PT_{par}
	D	TR	P	F	
1. Li et al.	X	X			0.50
2. Moser et al.	X	X	X		0.75
3. Oriol et al.	X	X	X		0.75
4. Hasan et al.	X	X			0.50
5. Haiteng et al.	X	X	X		0.75
6. Goel et al.	X	X		X	0.75
7. Sun et al.		X		X	0.50
8. Artaiam et al.	X			X	0.50
9. Raimondi et al.				X	0.25
10. Halima et al.	X	X			0.50

Tabla 3-4. Valor de PT_{par} para cada herramienta analizada.

De esta forma se obtiene el valor correspondiente al primer criterio para cada modelo y se procede a mostrar los valores obtenidos para el segundo criterio y tercer criterio, el tipo de servicios monitoreados y el tipo de implementación, los cuales se muestran en la tabla 3-5.

Enfoque	T_{sm}	T_{imp}
1. Li et al.	1	1
2. Moser et al.	1	1
3. Oriol et al.	1	0.5
4. Hasan et al.	1	0
5. Haiteng et al.	1	1
6. Goel et al.	1	1
7. Sun et al.	1	0.5
8. Artaiam et al.	0	0.5
9. Raimondi et al.	0.5	1
10. Halima et al.	1	1

Tabla 3-5. Valor de T_{sm} y T_{imp} para cada herramienta analizada.

Con los valores de los tres criterios obtenidos, se procede a calcular la calificación final para cada herramienta analizada. Los resultados se muestran en la tabla 3-6.

Enfoque	PT_{par} (60%)	T_{sm} (20%)	T_{imp} (20%)	Calificación
1. Li et al.	0.30	0.20	0.20	**0.70**
2. Moser et al.	0.45	0.20	0.20	**0.85**
3. Oriol et al.	0.45	0.20	0.10	**0.75**
4. Hasan et al.	0.30	0.20	0	**0.50**
5. Haiteng et al.	0.45	0.20	0.20	**0.85**
6. Goel et al.	0.45	0.20	0.20	**0.85**
7. Sun et al.	0.30	0.20	0.10	**0.60**
8. Artaiam et al.	0.30	0	0.10	**0.40**
9. Raimondi et al.	0.15	0.10	0.20	**0.45**
10. Halima et al.	0.30	0.20	0.20	**0.70**

Tabla 3-6. Valor de calificación final para cada herramienta analizada.

De acuerdo a esto, la clasificación ordenada de las herramientas sería:

Enfoque	Calificación
1. Moser et al.	0.85
2. Haiteng et al.	0.85
3. Goel et al.	0.85
4. Oriol et al.	0.75
5. Li et al.	0.70
6. Halima et al.	0.70
7. Sun et al.	0.60
8. Hasan et al.	0.50
9. Raimondi et al.	0.45
10. Artaiam et al.	0.40

Tabla 3-7. Herramientas ordenadas de acuerdo a la calificación.

Se puede observar que 3 enfoques obtienen una calificación mayor a 0.80, de esta forma se puede concluir que dichos modelos representan una base importante en el momento de determinar qué parámetros que serán tenidos en cuenta para el desarrollo de un esquema para el monitoreo de servicios convergentes, las métricas con las cuales estos son calculados y los métodos para determinar una posible falla en el cumplimiento de los criterios de QoS establecidos para cada uno de ellos

3.2 Métricas y Reglas propuestas

Teniendo en cuenta la evaluación de las herramientas para el monitoreo de la sección 3.3 y la tabla 3-7 donde se muestran los resultados de la misma, se ha tomado como base los enfoques presentados por Moser et al. [31], Haiteng et al. [28] y Goel et al. [29] para determinar los parámetros de QoS que serán observados en el esquema de monitoreo de servicios convergentes propuesto.

En la descripción de dichos enfoques existe una concordancia en los parámetros de QoS que son tenidos en cuenta, la herramienta propuesta por Moser et al. [31] considera el tiempo de respuesta, disponibilidad y precisión. Haiteng et al. [28] también coincide en los mismos 3 parámetros. Mientras que Goel et al. [29] considera el tiempo de respuesta, disponibilidad y la fiabilidad.

Lo anterior evidencia una correspondencia en el monitoreo del tiempo de respuesta y la disponibilidad, de esta forma, dichos parámetros serán los considerados en el presente trabajo.

Tomando como base lo anterior, y de acuerdo a la forma en que se miden los parámetros de QoS en [31], [28] y [29], se proponen las siguientes métricas para la disponibilidad y el tiempo de respuesta asociados a cada servicio atómico que compone el servicio convergente, las cuales son mostradas a continuación en la Tabla 3-8.

Parámetro	Métrica	Notación
Disponibilidad	$\mathrm{D(S)} = 1 - \frac{\mathrm{downtime}}{\mathrm{uptime}}$ (3)	El *downtime* y *uptime* son medidos en segundos.
Tiempo de Respuesta	$\mathrm{Tr(S)} = \mathrm{t_f(S)} - \mathrm{t_i(S)}$ (4)	Donde $\mathrm{t_i(S)}$ representa el tiempo en que se realiza la petición al servicio y $\mathrm{t_f(S)}$ representa el tiempo en que el usuario recibe la respuesta a esa petición.

Tabla 3-8. Métricas para medir disponibilidad y tiempo de respuesta.

Una vez definidas las métricas para los parámetros no funcionales de QoS que serán observados en el mecanismo para el monitoreo de servicios convergentes propuesto, se procede a establecer una serie de reglas que permitan calcular cuantitativamente los valores aceptados para cada uno de ellos y determinar en qué momento un parámetro excede los límites permitidos.

Actualmente no existen valores estándar aceptados para establecer cuáles deben ser los valores permitidos para cada uno de los parámetros en los servicios atómicos tanto web como telco que componen los servicios convergentes, debido a que estos valores son acordados de manera subjetiva entre proveedores de servicios y sus clientes mediante los SLA, de esta forma, dichos valores pueden variar entre un servicio y otro.

Teniendo en cuenta lo anterior, se evidencia la necesidad de establecer unos límites en los valores de QoS, los cuales son definidos de acuerdo a los diferentes escenarios donde se lleve a cabo la ejecución de los servicios convergentes, de esta forma, basados en el modelo inicial para calcular valores de disponibilidad y tiempo de respuesta en servicios compuestos establecido en [38], se proponen las siguientes reglas para el monitoreo de servicios convergentes:

- ✓ **Regla 1:** sea un Servicio Convergente SC_1 el cual incluye un requerimiento de disponibilidad D_{SC1}, y está compuesto por una serie de servicios atómicos (web y telco) $Sat_1, Sat_2, Sat_3, \dots, Sat_n$, cada uno de los cuales tendrá asociado su propio valor de disponibilidad. Luego, la disponibilidad del servicio convergente será igual al producto de los valores de disponibilidad de los servicios atómicos que lo componen, así:

$$D_{SC1} = \prod_{k=1}^{n} D_{Satk} \quad (5)$$

- ✓ **Regla 2:** sea un Servicio Convergente SC_1 el cual incluye un requerimiento de tiempo de respuesta TR_{SC1}, y está compuesto por una serie de servicios atómicos (web y telco) $Sat_1, Sat_2, Sat_3, \dots, Sat_n$, cada uno de los cuales tendrá asociado su propio valor de tiempo de respuesta. Luego, el tiempo de respuesta del servicio convergente será igual a la suma de los tiempos de respuesta de los servicios atómicos que lo componen, así:

$$TR_{SC1} = \sum_{k=1}^{n} TR_{Satk} \quad (6)$$

De esta forma, en el momento en que el valor de disponibilidad o tiempo de respuesta asociado a un determinado servicio atómico comprometa los requerimientos establecidos para el servicio convergente que compone, es necesario emitir una Alarma. La forma en que se realiza este proceso es descrita de manera detallada en el Capítulo 4, donde se describe el mecanismo de monitoreo que lleva a cabo esta labor.

RESUMEN

En este capítulo se presentó una descripción de los actuales modelos y herramientas que abordan el problema del monitoreo de servicios web, describiendo su funcionamiento y los parámetros de QoS que tienen en cuenta. Posteriormente fueron definidos una serie de criterios que permitieron realizar una evaluación de dichas herramientas; esto con el fin de determinar cuáles son los parámetros de QoS que mejor se adaptan al monitoreo de servicios convergentes y la forma en que estos serán medidos en el mecanismo de monitoreo que se propone en el presente trabajo de grado.

Capítulo 4

4. Definición de mecanismo de control para el monitoreo de servicios convergentes

En este capítulo se presenta el estudio y análisis realizado para la definición de un mecanismo para el monitoreo de servicios convergentes. En primer lugar se aborda el concepto de servicios convergentes y su implementación enmarcados dentro del proyecto Telcomp 2.0, de tal manera que sea posible seleccionar los puntos críticos donde sea adecuado realizar el monitoreo de los servicios en tiempo de ejecución. Posteriormente se detalla la implementación del mecanismo de monitoreo teniendo en cuenta un enfoque tanto reactivo como proactivo para el monitoreo de los servicios, permitiendo realizar un monitoreo constante y exhaustivo de los servicios de acuerdo a las métricas y reglas de QoS definidas en el capítulo 3.

4.1 Servicios Convergentes

El desarrollo del presente trabajo de grado está enmarcado dentro del Proyecto TelComp2.0: descubrimiento y composición de componentes complejos para la creación de Servicios Telco 2.0 [12], por lo tanto, es necesario conocer la estructura de los servicios convergentes que propone dicho proyecto y que serán monitoreados bajo el esquema de monitoreo desarrollado.

El proyecto TelComp2.0 propone la generación de una plataforma dirigida a soportar el proceso de creación, composición y ejecución de nuevos servicios convergentes, brindando a los desarrolladores y diseñadores herramientas que les permitan articular servicios atómicos (Web/Telco) sobre un entorno unificado[3], con el objetivo de definir nuevas funcionalidades de desarrollo ágil que brinden un valor agregado a los nuevos servicios. La figura 4-1 presenta una visión general de la plataforma Telcomp2.0 y su proceso para crear nuevos servicios convergentes.

[3] Entorno Unificado*:* un entorno en el cual convergen funcionalidades del mundo de las telecomunicaciones y servicios del mundo Web.

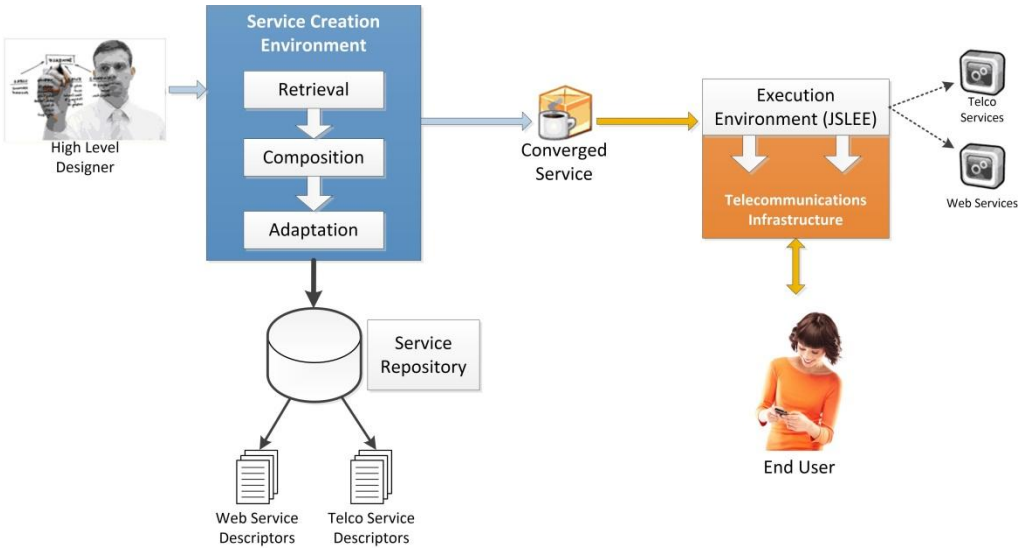

Figura 4-1. Vista General de la Plataforma TelComp2.0

En la figura anterior se puede observar que el repositorio de servicios atómicos para la creación de los nuevos servicios convergentes está compuesto por dos tipos de servicios, los primeros, brindados por el dominio de la Web (Facebook, Linkedin, Twitter, etc); mientras que los segundos, son ofrecidos desde los operadores de telecomunicaciones (buzón de mensajes, llamada, SMS, etc).

Teniendo esto en cuenta, dentro de TelComp2.0 se ha definido una estructura general que busca integrar estos dos tipos de servicios en servicios convergentes, tomando JAIN SLEE como entorno de ejecución, tal y como se muestra en la figura 4-2.

La lógica de ejecución de un servicio convergente, es manejada por un SBB (Bloque de Construcción de Servicio) orquestador, el cual se comunica con una serie de servicios atómicos a través del disparo y recepción de unos elementos proporcionados por la especificación JAIN SLEE, conocidos como eventos, los cuales representan ocurrencias o cambios de estado significativos ya sea dentro o fuera del SLEE y se utilizan para la comunicación entre entidades SBB que están localizadas en sus diferentes estructuras.

El funcionamiento general de los eventos se basa en dos tipos: los productores de eventos, los cuales disparan los eventos y los consumidores de eventos, los cuales reciben los eventos. El encargado de enviar los eventos disparados por los productores hacia los consumidores es otro elemento llamado *SLEE Event Router* [5].

Los productores de eventos incluyen 4 fuentes:

- ✓ Entidades de adaptadores de recursos: los eventos disparados por los adaptadores de recursos son conocidos también como eventos de recursos. Estos eventos son definidos por los propios adaptadores de recursos de acuerdo a su funcionamiento [5].

- ✓ SLEE: también conocido como eventos SLEE, los tipos de evento disparados por el SLEE incluyen el *Activity End Event* y el *Service Started Event* [5].

- ✓ SLEE *Facilities:* conocidos también como *SLEE Facility Events*, entre estos se puede encontrar el SLEE *Timer Facility*, encargado de disparar los *Timer Events* [5].

- ✓ Entidades SBB: los objetos SBB que se encuentren activos pueden disparar este tipo de eventos. Estos eventos se definen como eventos personalizados y son creados por el desarrollador de acuerdo a sus necesidades. Generalmente las entidades SBB utilizan eventos personalizados para comunicarse unos con otros [5].

Por otra parte, en los consumidores de eventos, dentro de la especificación JAIN SLEE sólo están definidas las entidades SBB, es decir los eventos personalizados.

Un SBB que dispare eventos debe definir los tipos de eventos que dispara. Por cada evento, el desarrollador debe declarar un método que permita disparar el evento desde la clase SBB. Un objeto SBB invoca el método de disparo del evento para pasarle dicho evento al SLEE Event Router. Así mismo, cada SBB que recibe eventos debe declarar los tipos de eventos que recibe. Para cada evento recibido, mediante el SLEE Event Router, el SBB define e implementa un método controlador en su clase SBB, este método contiene la lógica de la aplicación para procesar los eventos.

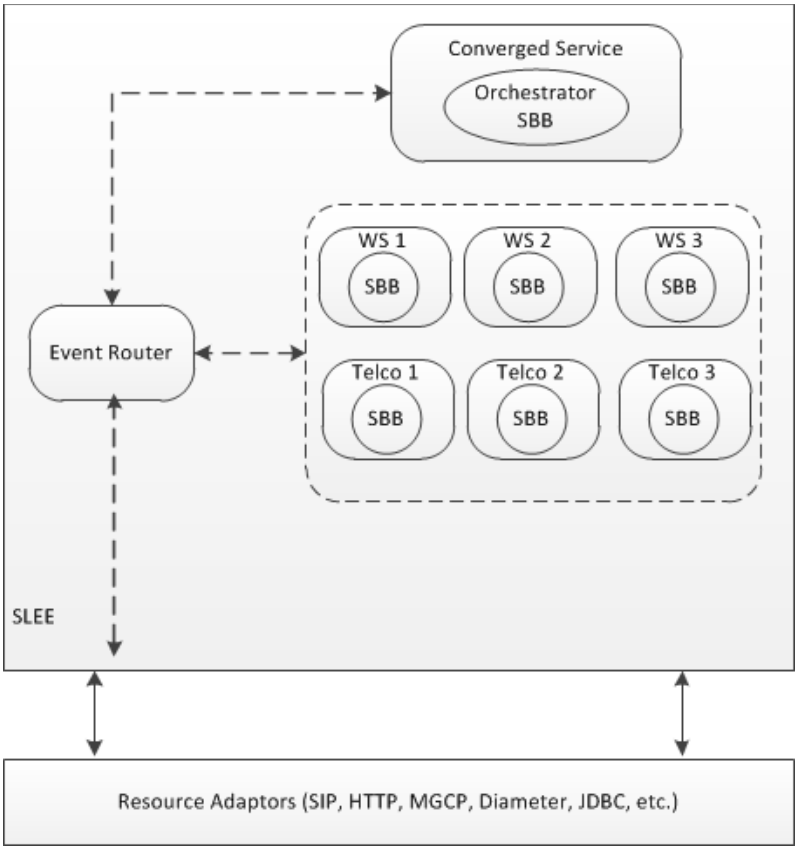

Figura 4-2. Estructura General de un Servicio Convergente

De acuerdo a lo anterior, para los servicios convergentes desarrollados bajo el proyecto Telcomp2.0, tanto la producción como el consumo de eventos se realizan siguiendo los lineamientos de las entidades SBB, ya que permiten personalizar las tareas requeridas durante el flujo de ejecución de los servicios convergentes, como por ejemplo la invocación de servicios atómicos (Web / Telco).

De esta forma, cada servicio convergente desarrollado tiene un SBB orquestador, el cual es encargado de crear y recibir los eventos asociados al flujo de los servicios atómicos que componen el servicio convergente. Como se mencionó anteriormente, ya que estos eventos son originados por medio de las entidades SBB, deben ser definidos por el desarrollador.

Para ilustrar mejor este proceso, se puede considerar un servicio convergente básico, el cual está compuesto por dos servicios atómicos, el primero de ellos web y el segundo Telco, cuyo flujo de ejecución se muestra en la figura 4-3 y su estructura de funcionamiento en la figura 4-4.

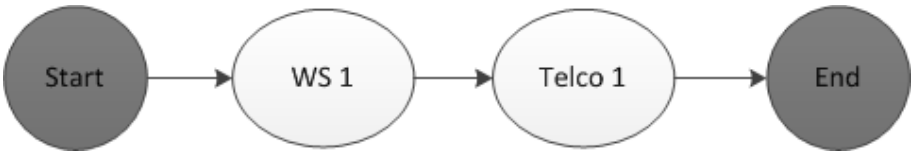

Figura 4-3. Flujo de ejecución servicio convergente básico

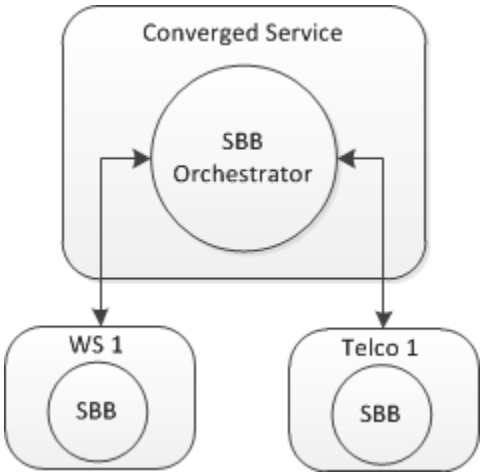

Figura 4-4. Estructura de un servicio convergente básico

Como se mencionó anteriormente, el servicio convergente que contiene el SBB orquestador es el encargado de disparar y recibir los eventos. Para el ejemplo considerado, el flujo de ejecución inicia cuando el usuario invoca el servicio convergente, de esta forma se activa el evento *Service Started Event* generado por el SLEE, de inmediato el SBB orquestador invoca el Servicio Web 1 (WS1) mediante el disparo de un evento asociado a este servicio, luego, pasa a esperar la respectiva respuesta que será recibida a través de otro evento, y posteriormente invoca al servicio Telco 1, nuevamente mediante el disparo de un evento asociado al servicio atómico Telco 1, el SBB orquestador recibe la respuesta a través de otro evento y finaliza su ejecución.

Este proceso se lleva a cabo de manera similar incluso cuando el flujo de ejecución del servicio convergente involucra más servicios atómicos y diferentes patrones de flujo. El

algoritmo asociado al flujo de ejecución de los tres eventos mencionados anteriormente para el servicio convergente básico: inicio, envío y recepción, es mostrado a continuación.

Algoritmo 1: Flujo de ejecución servicio convergente básico

```
1.  INPUTS: ws1wsdl, ts1wsdl
2.  OUTPUTS:
3.  BEGIN
4.    Detectar invocación servicio convergente = onServiceStartedEvent()
5.    /* Implementación del método… */
6.    Disparar evento hacia el primer servicio atómico = StartWS1Event ()
7.    Recibir evento de finalización del primer servicio atómico = onEndWS1Event ()
8.    /* Implementación del método… */
9.    Disparar evento hacia el segundo servicio atómico = StartTS1Event ()
10.   Recibir evento de finalización del segundo servicio atómico = onEndTS1Event ()
11.   /* Implementación del método y finalización del servicio */
12. END
```

Teniendo en cuenta las características de los servicios convergentes del proyecto TelComp2.0 que se han descrito anteriormente, se puede notar que los métodos que finalizan la ejecución de un servicio atómico y dan paso al siguiente servicio atómico, representan un punto crítico o de control debido a que son los encargados de manejar los eventos de respuesta disparados tras la invocación de cada servicio atómico y de proceder a invocar el siguiente, de esta forma, sí una de estas invocaciones dan como resultado un fallo, debido a que un servicio atómico no se encuentra disponible o el tiempo de respuesta excede los límites establecidos, puede comprometer el funcionamiento de todo el servicio convergente. Considerando lo anterior, es conveniente tener funciones de monitoreo en dichos puntos que ayuden a detectar cuál servicio atómico ha fallado y la localización exacta de la falla dentro del flujo de ejecución. Para abordar esta problemática, y además, monitorear las métricas y reglas de QoS propuestas en el Capítulo 3, ha sido desarrollado el mecanismo de monitoreo que se describe a continuación, en la sección 4.2.

4.2 Mecanismo de monitoreo

De acuerdo a lo mencionado en la sección 4.1, se hace necesario construir un mecanismo de monitoreo que permita observar constantemente los cambios que puedan presentarse durante la instalación y ejecución de nuevos servicios convergentes. Este mecanismo deberá estar asociado tanto a los puntos críticos identificados previamente, permitiendo realizar un control y seguimiento constante de los servicios durante su ejecución, como a las métricas y reglas de QoS definidas en el capítulo 3.

Para la construcción del mecanismo de monitoreo se abordan dos enfoques relacionados con el monitoreo de servicios: un enfoque proactivo y otro reactivo. De acuerdo al enfoque reactivo, un sistema de monitoreo reacciona ante un fallo, detectando cualquier violación en los acuerdos de calidad de servicio y además es capaz de identificar el origen de los

fallos y violaciones presentadas [39]. Por otra parte, bajo un enfoque proactivo, los sistemas de monitoreo son capaces de monitorear y supervisar su entorno de manera constante y sin intervención humana directa. En un enfoque proactivo, los usuarios humanos sólo se encargan de generar las políticas de funcionamiento y calidad así como de intervenir en el momento en que sea necesario tomar decisiones que afecten de manera realmente crítica al sistema [40].

El mecanismo de monitoreo propuesto es diseñado bajo la especificación JAIN SLEE y es desplegado junto a los servicios convergentes que serán monitoreados. La figura 4-5 muestra una vista general del mecanismo de monitoreo, relacionando los diferentes módulos que hacen parte del mismo y que serán descritos durante la presente sección.

Es importante señalar que debido a que el presente trabajo de grado soporta el proyecto de Maestría en Ingeniería Telemática, titulado *Reconfiguración Automática de Servicios Convergentes en Entornos JAIN SLEE*, para efectos de referencia, se ha agregado un Módulo de Reconfiguración dentro de la vista general del mecanismo de monitoreo, el cual corresponde a dicho trabajo y no será abordado en el presente capítulo.

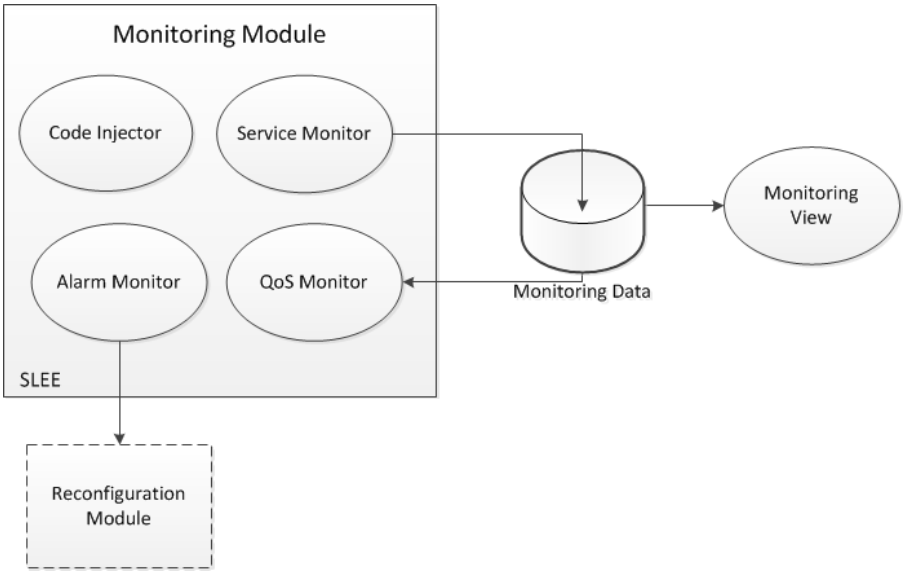

Figura 4-5. Vista General del Mecanismo de Monitoreo

4.2.1 Módulo de Monitoreo

El módulo de monitoreo se encuentra compuesto por un conjunto de servicios relacionados entre sí, los cuales son ejecutados junto a los servicios convergentes que serán monitoreados dentro del entorno de ejecución Mobicents JAIN SLEE. Los diferentes elementos y tecnologías que soportan y pertenecen a este módulo se describen a continuación.

4.2.1.1 Detección de nuevos servicios convergentes

Las tareas de monitoreo asociadas al mecanismo propuesto, inician con la detección del despliegue e instalación de nuevos servicios convergentes en el Servidor de Aplicaciones de Mobicents JSLEE, para este fin, tanto el módulo *Service Monitor* como el *Code Injector,* hacen uso del evento *Service Started Event* proporcionado por la especificación JAIN SLEE. Este evento es invocado cada vez que un nuevo servicio es instalado, permitiendo programar las tareas a ejecutar por parte de dicho evento.

Los servicios que son instalados dentro del Servidor de Aplicaciones de Mobicents JSLEE tienen definidos desde su creación, tres campos que permiten su identificación: Nombre, Proveedor y Versión. En el momento en que un servicio es instalado, se accede al Java MBean [41] que contiene la información de los servicios activos en el servidor a través de las interfaces JMX (Java Management *Extensions*) [42], proporcionadas por el SLEE, de tal forma que es posible obtener los nombres de los servicios instalados en el servidor.

El proyecto TelComp2.0 tiene una estructura para los servicios que permite agregar un identificador al final del nombre del servicio para poder identificar si éste corresponde a un servicio atómico o a un servicio convergente. Cuando un servicio corresponde a un servicio atómico, su nombre finaliza con la palabra *Service*, mientras que si el servicio desarrollado es un servicio convergente, el identificador final estará dado por las palabras *Converged Service*. Estos nombres son almacenados en una lista y posteriormente se verifica que el servicio recientemente instalado tenga un nombre correspondiente a un servicio convergente y no a un servicio atómico. Una vez se valida que el servicio instalado corresponde a un servicio convergente, se proceden a realizar las tareas de monitoreo.

4.2.1.2 Módulo Code Injector

4.2.1.2.1 Instrumentación de código en tiempo de ejecución

El ciclo normal del desarrollo de un programa es la edición del código fuente y su respectiva compilación y posterior ejecución. Sin embargo, algunas veces este ciclo puede ser muy restrictivo. Bajo este aspecto, la instrumentación de código en tiempo de ejecución se presenta como una herramienta que soporta variedad de aplicaciones, las cuales incluyen la depuración, monitoreo de componentes, composición de aplicaciones entre diferentes paquetes, entre otras. Dependiendo del uso, el código insertado puede ya sea dotar un programa existente de funciones auxiliares como la medición del rendimiento de una aplicación, o adicionar sentencias mediante la manipulación de las estructuras de datos de la aplicación.

Java, como lenguaje de programación que enmarca el presente trabajo de grado, soporta la reflexión de código mediante una API llamada *The Reflection API* [43], la cual sí bien soporta la instanciación de una clase, invocar un método a través de dicha API, u obtener el valor de un campo, limita todo su uso a la introspección[4] y no permite la instrumentación de código en tiempo de ejecución.

[4] Introspección: habilidad de interceptar y leer estructuras de datos de en un programa, como por ejemplo una clase.

Teniendo en cuenta lo anterior, en [44] se muestra el desarrollo de una herramienta llamada Javassist [45], la cual es una librería de clases que habilita la Reflexión Estructural[5] dentro de Java, lo cual brinda las funcionalidades necesarias para instrumentar los servicios convergentes con el código requerido para dotar dichos servicios con capacidades de monitoreo. A continuación se presenta una descripción detallada de Javassist.

Javassist: la tecnología Javassist [45] es una extensión de [43] y consiste en un conjunto de herramientas que permiten el desarrollo de traductores Java-bytecode, es decir, una librería de clases en Java para transformar archivos de clases compiladas de Java (bytecode) en tiempo ejecución o compilación.

En Java, el bytecode obtenido de la compilación de un programa es almacenado en archivos de extensión *.class*, cada uno de los cuales corresponde a una clase distinta. Javassist realiza reflexión estructural traduciendo las alteraciones de código que se desean realizar en bytecode equivalente de los archivos *.class* a transformar. Así, después de la transformación, los archivos de la clase modificada son cargados dentro de la Máquina Virtual de Java (JVM) para su ejecución.

Javassist provee dos niveles de API: a nivel de código fuente y a nivel de bytecode.

La API a nivel de fuente permite editar el archivo de una clase sin conocimiento de las especificaciones del bytecode Java, esta API está diseñada sólo con vocabulario en lenguaje Java, lo cual permite incluso especificar en forma de código fuente el bytecode que será insertado dentro de la clase que se quiere modificar y Javassist lo compila sobre la marcha. Por otra parte, la API a nivel de bytecode permite editar directamente el archivo de la clase como cualquier otro editor de bytecode [44].

Javassist es comúnmente usado para agregar nuevos métodos o funcionalidades a una clase compilada mediante el uso de la reflexión estructural, pero también puede ser usado con fines de introspección en tiempo de ejecución, ya que para este fin, proporciona los mismos métodos de [43]. De igual manera, permite a programas hechos en Java usar metaobjetos que controlan el llamado de los métodos de una clase, sin necesidad de usar un compilador especial o una máquina virtual [45].

4.2.1.2.2 Detección de puntos de control e inserción de código

El módulo Code Injector presenta un enfoque reactivo para el monitoreo de servicios convergentes basado en la inserción automática de código en tiempo de ejecución. De tal forma que detecta fallas en el flujo normal de ejecución del servicio convergente que está siendo monitoreado, con el fin de obtener información que permita tomar las medidas pertinentes, a partir de la información que aquí sea capturada.

Este módulo, es un servicio desarrollado bajo la especificación JAIN SLEE, de tal forma que puede ser ejecutado en cualquier entorno de ejecución lógica de servicios (SLEE). En este caso, es introducido dentro del Servidor de Aplicaciones de Mobicents JBoss y funciona en forma paralela a los servicios convergentes que van a ser monitoreados.

[5] Reflexión Estructural: habilidad de interceptar y alterar las estructuras de datos de un programa, como por ejemplo una clase o un método.

El proceso de instrumentación de código inicia con la instalación de un nuevo servicio convergente, proceso que se describió en la sección 4.2.1.1. Una vez se ha comprobado que es un servicio que debe ser monitoreado, mediante la funcionalidad de introspección que proporciona la tecnología Javassist se accede a la clase del SBB orquestador del servicio convergente y se detectan los métodos correspondientes a los eventos que finalizan la ejecución de un servicio atómico web o telco, de acuerdo al flujo de ejecución del servicio convergente, los cuales tal como se mencionó anteriormente corresponden a los puntos críticos donde es necesario realizar tareas de monitoreo.

Bajo este aspecto, por medio de la API a nivel de código fuente que proporciona la tecnología Javassist, se instrumenta cada uno de dichos métodos correspondientes al manejo de los eventos de respuesta, con el código que brinda la funcionalidad necesaria para monitorear el comportamiento de los servicios atómicos y detectar posibles fallas. El código insertado es capaz de determinar sí un servicio atómico ha sido ejecutado de manera correcta o no, y en caso de no ser así, a través de la funcionalidad *Alarm Facility*, descrita en 2.1.2.1 y definida por la especificación JAIN SLEE, se establece una Alarma que contiene información acerca de la fuente del fallo y la localización de este dentro del flujo de ejecución del servicio convergente.

Una vez se realiza este proceso, el módulo *Code Injector* empaqueta el archivo de la clase compilada dentro de la unidad desplegable del servicio convergente y la instala nuevamente en el Servidor de Aplicaciones de Mobicents JSLEE, donde el servicio instrumentado con funciones de monitoreo estará disponible para su invocación.

Es importante aclarar que el código instrumentado en la clase del SBB orquestador, no afecta el flujo de ejecución y el funcionamiento normal del servicio convergente.

El Algoritmo 2 explica de manera general el proceso de instrumentación de código realizado por parte del módulo Code Injector.

Algoritmo 2: Funcionamiento Módulo Code Injector

```
1.  INPUTS: ServiceName, ServicePath, ServerPath, AtomicService[]
2.  BEGIN
3.  while Server is Running do
4.  Detect new service deployment
5.    if ServiceName is Converged then
6.    Unpack Deployable Unit = getDU (ServerPath)
7.      for each AtomicService do
8.        Find critical method
9.        Instrument method with monitoring code
10.     end for
11.   else
12.     Keep monitoring service deployment
13.   end if
14. Package compiled class in Deployable Unit = updateDU (ServicePath)
15. Install Deployable Unit in server = deployDU (ServerPath)
16. end while
17. END
```

4.2.1.3 Módulo Service Monitor

Al igual que el módulo *Code Injector,* el *Service Monitor* detecta que un nuevo servicio convergente ha sido desplegado en el Servidor de Aplicaciones de Mobicents JSLEE, mediante el evento *Service Started Event,* el cual se describió anteriormente.

4.2.1.3.1 Monitoreo de servicios atómicos

Una vez se ha detectado que el nuevo servicio instalado es un servicio convergente, se procede a acceder al código fuente del SBB orquestador del servicio. A través de técnicas de introspección de código brindadas por Javassist, es posible inspeccionar la clase en mención y extraer los datos correspondientes a los diferentes servicios atómicos que utiliza el servicio convergente, ya sean servicios web o telco.

Los servicios atómicos son descritos al inicio de la clase del SBB orquestador mediante variables en las cuales se almacenan los valores de las WSDL asociadas a cada servicio, estos valores son guardados en una variable de tipo Lista con el fin de almacenar y registrar los servicios actualmente activos que están siendo utilizados por algún servicio convergente que se encuentre en funcionamiento. En este punto, la lista de los servicios detectados inicialmente se compara con los servicios encontrados en el nuevo servicio convergente de tal manera que solamente son considerados los servicios que no se encuentren almacenados previamente en dicha lista, esto con el fin de no redundar en un mismo servicio atómico.

Dada la naturaleza proactiva bajo la cual el módulo *Service Monitor* realiza el monitoreo de los servicios atómicos, es necesario tener en cuenta dos funcionalidades importantes: la primera está asociada a la capacidad del sistema para realizar tareas de forma periódica y automática, mientras que la segunda, está relacionada con la conexión y el acceso a los servicios atómicos para recibir su respuesta y por ende, conocer su estado actual de funcionamiento.

Para el primer objetivo se hace uso del *Timer Facility*, el cual se describió en la sección 2.1.2.2, proporcionado por la especificación JAIN SLEE y que permite realizar acciones periódicas durante la ejecución de los servicios, además puede manejar un número determinado de temporizadores, cada uno de los cuales son independientes y pueden disparar uno o varios eventos.

Cada temporizador tiene establecido un tiempo de inicio, en este caso, dicho tiempo se configura con un valor de tiempo correspondiente al momento en que el módulo *Service Monitor* ha sido desplegado en el Servidor de Aplicaciones Mobicents JSLEE. Dicho temporizador se puede ejecutar una sola vez, o también puede ser configurado para que se ejecute de manera periódica.

Para este caso en particular, es necesario utilizar un temporizador periódico, el cual funciona de la siguiente forma: el *Timer Facility* dispara el primer evento para el temporizador en el tiempo de inicio especificado y periódicamente dispara eventos para un intervalo de tiempo dado. Luego, continúa disparando eventos para un temporizador periódico hasta que este es cancelado o haya alcanzado el número de repeticiones especificado durante su configuración. En este caso, el temporizador debe estar funcionando indefinidamente, por lo tanto, el número de repeticiones se configura en “0”, lo cual indica que se repetirá de forma infinita o hasta que sea cancelado.

Cada vez que el *Timer Event* es disparado, se invoca un método que se encarga de realizar las peticiones Http que apuntan a la dirección de las WSDL asociadas a los servicios atómicos que han sido almacenados previamente en una lista. Dentro de este método y a través de un recurso externo proveído por un Adaptador de Recursos, se configura un cliente Http para realizar las peticiones hacia cada uno de los servicios atómicos, almacenando el tiempo en el que inicia el proceso para posteriormente compararlo con el tiempo en el que se recibe la respuesta, de tal manera que sea posible obtener el valor del parámetro de QoS correspondiente al Tiempo de Respuesta.

El proceso para obtener la respuesta se realiza mediante el evento *Response Event*, el cual recibe la información de las respuestas correspondientes a las peticiones HTTP realizadas hacia los servicios atómicos. La implementación de esta sección permite clasificar los resultados de la petición realizada a través de las definiciones de los códigos de estado [46], en este caso, se consideran 3 estados cuyas respuestas indican estados críticos, los cuales se muestran en la tabla 4-1.

Código	Estado	Detalle
200	*OK*	La petición fue enviada y respondida exitosamente.
404	*Not Found*	El servidor no encontró ninguna coincidencia con la petición realizada.
500	*Internal Server Error*	El servidor encontró una condición inesperada que le impidió cumplir con la solicitud.

Tabla 4-1. Códigos de Estado de una respuesta HTTP

Una vez se recibe la respuesta, el sistema procede a almacenar los datos para cada uno de los servicios atómicos en la base de datos de acuerdo a la respuesta, de tal forma que se cuente con la información necesaria para poder realizar los cálculos y el análisis de las métricas y reglas definidas en el capítulo 3.

4.2.1.3.2 Persistencia de Datos

Tanto los resultados y datos recolectados por este módulo como los del resto del sistema de monitoreo, serán almacenados en un base de datos externa construida en MongoDB [47].

Como se mencionó en la sección anterior, una vez se recibe la respuesta de la petición HTTP, los datos son almacenados de acuerdo al código de estado recibido. De esta forma, una respuesta es exitosa solamente cuando se recibe una respuesta de código 200, en este caso se almacenan los valores del tiempo de respuesta y de disponibilidad, cuyo valor en este caso será de 1 para ese intervalo de tiempo en el cual se realiza la petición, dado que el servicio se encuentra disponible, además de la identificación de la WSDL del servicio atómico y el nombre del servicio convergente que lo está utilizando

Si el código de estado no corresponde al número 200, significa que ocurrió una falla al realizar la petición o al recibir la respuesta. En este caso, el tiempo de respuesta es almacenado con un valor de 0, lo cual indica que la petición no fue exitosa o el servidor no es capaz de responder ante la solicitud, y un valor de disponibilidad, el cual tendrá igualmente un valor de 0, para indicar que el servicio no está disponible en el intervalo

de tiempo en que se realizó la petición HTTP, además de la identificación de la WSDL del servicio atómico y el nombre del servicio convergente que lo está utilizando.

Una descripción más detallada acerca del almacenamiento de datos se muestra en la sección 4.2.2, la cual trata específicamente el módulo de datos implementado en el sistema.

El Algoritmo 3 explica de manera general el flujo de funcionamiento del módulo Service Monitor.

Algoritmo 3: Funcionamiento Módulo Service Monitor

```
1.  INPUTS: ServiceName, ServicePath, ServerPath, WSDL[], WSDLList[], Timer
2.  OUTPUTS: QoSData
3.  BEGIN
4.  while Server is Running do
5.  Detect new service deployment
6.    if ServiceName is Converged then
7.    Unpack Deployable Unit = getDU (ServerPath)
8.      if ServiceName is a new Converged Service then
9.        store new WSDLs in WSDLList
10.     end if
11.   end if
12. Activate timer
13.   while timer is active do
14.     send Http requests from WSDLList
15.     receive response from requests
16.       if response is OK then
17.         store data in the database with received values
18.       if response is NOT OK then
19.         store data in the database with detected errors
20.       end if
21.       end if
22.   end while
23. end while
24. END
```

4.2.1.4 Módulo QoS Monitor

Este módulo es el responsable de analizar los datos correspondientes al monitoreo, los cuales han sido almacenados previamente en la base de datos por parte del módulo *Service Monitor*, así mismo, se encarga de calcular los valores de las métricas y reglas de QoS con el fin de detectar posibles violaciones. Esta tarea se realiza de forma periódica para cada servicio atómico que compone el servicio convergente. Una vez se detecta una violación, se establece una Alarma en el Servidor de Aplicaciones de Mobicents JSLEE, haciendo uso del *Alarm Facility*, la cual contiene información asociada al servicio atómico web o telco que ha dejado de funcionar correctamente, así como el servicio convergente asociado a dicho componente.

4.2.1.4.1 Acceso a datos

El módulo QoS monitor tiene como tarea principal analizar los datos recolectados y detectar posibles fallos durante la ejecución de los servicios, el acceso a dichos datos se realiza estableciendo la conexión con la base de datos en donde los datos correspondientes a los diferentes servicios atómicos están siendo almacenados periódicamente a través del módulo *Service Monitor*, como se mencionó en la sección 4.2.1.3.2.

MongoDB provee herramientas que facilitan de gran manera la forma en que los datos son accedidos y leídos, el acceso a dichos datos se realiza periódicamente haciendo uso de un temporizador, el cual se activa cada vez que ha transcurrido un intervalo de tiempo definido, y obtiene un número preestablecido de muestras para poder realizar el cálculo de las métricas y reglas de QoS.

La conexión a la base de datos continúa activa mientras el módulo *QoS Monitor* se encuentre activo. Una vez el módulo ha sido desinstalado o desactivado, la conexión a la base de datos de MongoDB también finaliza, el módulo *Monitoring Data* correspondiente al mecanismo de monitoreo, se explica más adelante en el documento, de manera más detallada.

4.2.1.4.2 Cálculo de métricas y reglas de QoS

El cálculo de las métricas y reglas de QoS, se debe hacer de manera periódica con el fin de monitorear el comportamiento de los diferentes servicios atómicos web y telco en tiempo real, de esta forma, para realizar los cálculos, también se hace necesario el uso de un temporizador que permita realizar un cálculo constante de las métricas y reglas de QoS. Una vez el temporizador se activa, se realiza el acceso a los datos e inmediatamente se procede a calcular las métricas y las reglas de QoS de acuerdo a lo establecido el capítulo 3.

Las métricas correspondientes a la disponibilidad y el tiempo de respuesta definidas en las ecuaciones (3) y (4) respectivamente, se calculan para cada servicio atómico que compone al servicio convergente que está siendo monitoreado, este proceso se realiza de manera periódica para un tiempo que será establecido de acuerdo al escenario y a los requerimientos de disponibilidad que sean planteados. Así, una vez se hayan calculado una serie de valores asociados a la disponibilidad y el tiempo de respuesta con los cuales se puede tener un valor promedio de estos para cada servicio atómico, se procede a verificar el cumplimiento de las reglas establecidas en las ecuaciones (5) y (6).

Como se mencionó en el capítulo 3, cada regla corresponde a los requerimientos planteados por el escenario donde se ejecutan los servicios convergentes, de esta forma.

El Algoritmo 4 explica de manera general el flujo de funcionamiento del módulo QoS Monitor.

Algoritmo 4: Funcionamiento Módulo QoS Monitor

```
1.  INPUTS: QoSDefinedRules, timer
2.  OUTPUTS: QoSMetrics, QoSRules, Alarm
3.  BEGIN
4.  while Server is Running do
5.  Connect to database
6.  Activate timer
7.    while timer is active do
8.      calculate QoSMetrics
9.      calculate QoSRules
10.     if QoSRules is different to QoSDefinedRules then
11.       activate Alarm
12.       end if
13.   end while
14. end while
15. END
```

4.2.1.5 Módulo Alarm Monitor

Este módulo se encarga de analizar constantemente el entorno de ejecución de JAIN SLEE, con el fin de detectar la activación de nuevas Alarmas que hayan sido establecidas por parte del módulo *QoS Monitor* en caso de violación de una regla en los valores de los parámetros de QoS, o por parte del propio servicio convergente, a través de la funcionalidad de monitoreo instrumentada desde el módulo *Code Injector.* Haciendo uso de un temporizador, este módulo es capaz de analizar los datos que contiene cada Alarma y finalmente enviar esta información hacia el módulo de reconfiguración.

4.2.1.5.1 Detección de Alarmas y conexión con el Módulo de Reconfiguración

Para detectar una nueva alarma, el módulo Alarm Monitor utiliza un temporizador que es activado cuando inicia la ejecución del mismo. Cuando el temporizador es activado por primera vez, el módulo accede al Java MBean que contiene la información relacionada con las alarmas y almacena en una lista las que se encuentran activas en ese momento. El módulo continúa monitoreando las Alarmas, comparando las listas almacenadas durante el proceso, de tal manera que cuando una nueva Alarma es detectada, de inmediato procede a ordenar los datos obtenidos con el fin de que estos puedan ser enviados al Módulo de Reconfiguración.

Una vez se obtienen los datos de una nueva Alarma, estos son organizados en un *HashMap*[6]*,* de acuerdo al formato establecido para recibir los datos por parte del

[6] HashMap: colección de objetos que no tienen un orden establecido.

Módulo de Reconfiguración, conteniendo toda la información de interés para dicho módulo. Finalmente, se dispara un evento personalizado que contiene los datos ordenados en el *HashMap*, este es recibido por parte del módulo de reconfiguración, el cual da inicio a las labores correspondientes al proceso que ese módulo implementa.

El siguiente algoritmo explica de manera general el flujo de funcionamiento del módulo *Alarm Monitor*.

Algoritmo 5: Funcionamiento Módulo Alarm Monitor

```
1.  INPUTS: alarmsList[], timer
2.  OUTPUTS: reconfigInputs<>, reconfigModEvent
3.  BEGIN
4.  while Server is Running do
5.  Activate timer
6.    while timer is active do
7.      store alarms in alarmList[]
8.      activate alarmListener
9.        while alarmListener is active do
10.         search for new alarms
11.           if new Alarm is found then
12.             get new Alarm data
13.             store data in reconfigInputs<>
14.             fire ReconfigModEvent
15.           end if
16.       end while
17.     end while
18. end while
19. END
```

4.2.2 Módulo Monitoring Data

Tradicionalmente, el tratamiento de datos en las aplicaciones de software está basado en sistemas DBMS (*Database Management System*), los cuales permiten realizar funciones dentro de la base de datos tales como almacenar, consultar, añadir, borrar y modificar los diferentes datos. A este tipo de sistemas pertenecen los RDBMS (*Relational Database Management Systems*), los cuales permiten implementar bases de datos relacionales. Una base de datos relacional es una colección de datos organizados como un conjunto de tablas formalmente descritas y relacionadas entre sí. Una de las aplicaciones estándar para el manejo de bases de datos relacionales es el Lenguaje de Consulta Estructurado (SQL) [48].

Actualmente y como respuesta a la rápida expansión de las aplicaciones basadas en la Web 2.0, han emergido otro tipo de sistemas para el almacenamiento de datos denominados *NoSQL*[7]. Estos sistemas se han diseñado para proveer un mejor desempeño en cuanto a la escalabilidad horizontal durante operaciones simples de

[7] La definición de *NoSQL* puede hacer referencia a *Not Only SQL* o a *Not Relational*.

escritura y lectura de datos [49]. De acuerdo a [50], la escalabilidad horizontal se define como una arquitectura de sistema que tiene la habilidad de distribuir tanto los datos como la carga del sistema de la forma más uniforme posible, sobre la mayor cantidad de servidores posible y mediante una arquitectura de "nada compartido". En contraste, los sistemas de bases de datos tradicionales no tienen o tienen muy poca habilidad para escalar horizontalmente sobre este tipo de aplicaciones.

Entre las principales ventajas de los modelos de bases de datos NoSQL respecto al tradicional modelo de bases de datos relacional, se pueden encontrar las siguientes:

- Los modelos NoSQL son más eficientes soportando "operaciones simples" [50].
- Las implementaciones NoSQL son escalables horizontalmente [50].
- Los modelos NoSQL realizan una distribución eficiente durante la indexación de datos. De igual manera, el uso de la memoria RAM durante el almacenamiento de datos es eficiente [49].
- Una base de datos NoSQL tiene la habilidad de añadir dinámicamente nuevos atributos a los datos guardados [49].

Dadas las características de los datos que se requieren almacenar en el mecanismo de monitoreo propuesto y las ventajas que presentan las bases de datos NoSQL en cuanto a escalabilidad, eficiencia y consumo de recursos, la base de datos para el sistema de monitoreo es implementada utilizando un sistema NoSQL.

Las bases de datos NoSQL se clasifican en 4 tipos diferentes de acuerdo al modelo de datos utilizado: el modelo de documento, el modelo de grafo, el modelo de clave-valor y el modelo de columnas extensas. Mientras las bases de datos relacionales almacenan los datos en filas y columnas, el modelo de documentos permite almacenar los datos en una estructura de datos compleja llamada documento. Los documentos utilizan una estructura parecida a JSON (*JavaScript Object Notation*) [51], un formato que ha obtenido gran popularidad entre los desarrolladores en los últimos años. A diferencia del acceso a datos que se realiza en una base de datos relacional mediante la consulta a múltiples columnas y tablas para acceder a un registro, en una base de datos NoSQL implementada bajo el modelo de documento cada registro y los datos asociados al mismo son almacenados en un único documento, lo cual simplifica el acceso a datos y reduce la necesidad de transacciones complejas durante la manipulación de los datos. Las bases de datos orientadas a documentos permiten un desarrollo rápido, con estructuras de información amigables al desarrollador y son muy eficientes en la implementación de aplicaciones web [52].

MongoDB es un sistema de bases de datos no relacional de código abierto orientado a documentos, reconocido por ser el sistema de base de datos NoSQL más utilizado y que cuenta con características que permiten reducir el tiempo de producción debido a las facilidades que brinda su experiencia de desarrollo [53]. Los registros en MongoDB se almacenan como documentos y cada conjunto de documentos se denomina colección. Un documento y una colección podrían compararse con el concepto de fila y tabla respectivamente dentro de una base de datos relacional. Los documentos son almacenados en formato BSON (*Binary JSON*) con un esquema dinámico, lo cual hace que la velocidad de consulta de datos sea más rápida y eficiente en comparación a las bases de datos relacionales, además facilita la integración de datos durante el desarrollo de aplicaciones [47].

Teniendo en cuenta lo anterior, se utiliza MongoDB para implementar la base de datos del sistema de monitoreo, pues permite desarrollar una base de datos orientada a documentos la cual se adapta de mejor manera a las características y cantidad de datos que serán almacenados, además facilitará su posterior uso e implementación durante el desarrollo del módulo de vista, el cual será descrito más adelante.

4.2.2.1 Estructura de la base de datos

Dentro del diseño del modelo de datos para una base de datos orientada a documentos se consideran dos modelos principales: El modelo de datos incrustados o empotrados y el modelo de datos normalizados. En el modelo de datos incrustados, todos los datos relacionados son insertados dentro de una misma estructura o documento, permitiendo a las aplicaciones almacenar las piezas de información relacionadas en el mismo registro de la base de datos. En general, el uso de este modelo permite un mejor rendimiento durante las operaciones de lectura, así como la habilidad de consultar y recuperar datos relacionados utilizando operaciones sencillas [54]. La figura 4-6 muestra el esquema general de un documento utilizando el modelo de datos incrustados.

Figura 4-6. Esquema general de un documento para el modelo de datos incrustados

Debido a la simplicidad de los datos recolectados durante el proceso de monitoreo, pero teniendo en cuenta el gran volumen de datos que se almacenan, se utiliza el modelo de datos incrustados para la persistencia de datos en la base de datos.

Como se mencionó en la sección 4.2.1.3.2, el módulo *Service Monitor* es el encargado de realizar el almacenamiento de datos de acuerdo a la respuesta recibida durante la ejecución de los distintos servicios atómicos que se están monitoreando. Los datos almacenados y que posteriormente serán utilizados para el cálculo de las métricas y reglas tanto en el módulo *QoS Monitor* como para mostrar gráficas de los valores de QoS en tiempo real, por parte del Módulo de Vista, se muestran en la siguiente tabla.

Nombre	Tipo	Descripción
wsdl	String	Nombre de la WSDL asociada al servicio atómico que se está monitoreando.
wsdl_id	Int	Id de la WSDL que está siendo monitoreada.
responseTime	Int	Tiempo que tarda en responder el servicio atómico que está siendo monitoreado medido en milisegundos.
availability	int	Disponibilidad del servicio atómico actual. 1 Indica disponibilidad, 0 indica que el servicio no está disponible en el instante en que fue consultado.
convergentService	String	Nombre del servicio convergente al cuál se encuentra asociado el servicio atómico que está siendo monitoreado.

Tabla 4-2. Datos almacenados en la base de datos implementada

Teniendo en cuenta lo anterior, el esquema general de un documento asociado a la base de datos del mecanismo de monitoreo desarrollado tiene la forma mostrada en la figura 4-7.

```
{
"_id" : ObjectId("525a008a44aea333278bfe4c"),
"wsdl" : "http://192.168.190.55:8084/EWApp-war/LinkedInWebService?wsdl",
"wsdl_id" : 1,
"responseTime" : 45,
"availability" : 1,
"convergentService" : "LinkedInJobNotificatorCS"
}
```

Figura 4-7. Documento asociado al mecanismo de monitoreo

La figura 4-8 muestra la estructura general de la base de datos implementada, la cual está compuesta por una base de datos denominada “MonitoringDb” dentro de la cual se tiene una colección llamada “QoSCollection”, que a su vez contiene los documentos con los valores y datos asociados al monitoreo de los diferentes servicios.

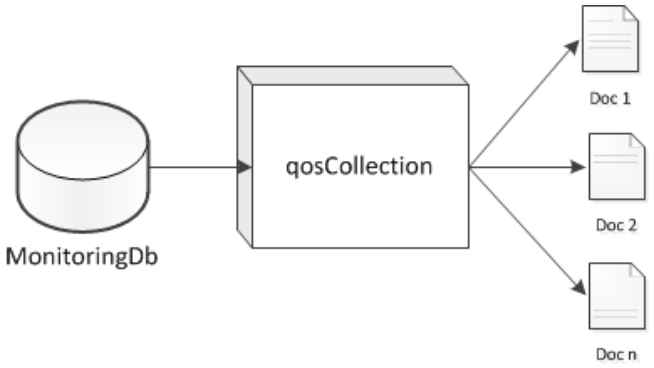

Figura 4-8. Estructura general de la base de datos implementada

4.2.3 Módulo Monitoring View

Para permitir una mejor observación del comportamiento de los parámetros de QoS asociados a los servicios que están siendo monitoreados, se desarrolla un módulo de vista basado en tecnologías web. De esta forma, este módulo permite observar mediante gráficos el comportamiento, en tiempo real, de los Estructura general de la base de datos implementada parámetros de disponibilidad y tiempo de respuesta asociados a cada servicio atómico web o telco que componen los servicios convergentes, posibilitando realizar un seguimiento constante de los mismos.

4.2.3.1 Estructura general del Módulo Monitoring View

La figura 4-9 muestra la estructura general del diseño del módulo de vista, indicando los diferentes componentes utilizados para su implementación.

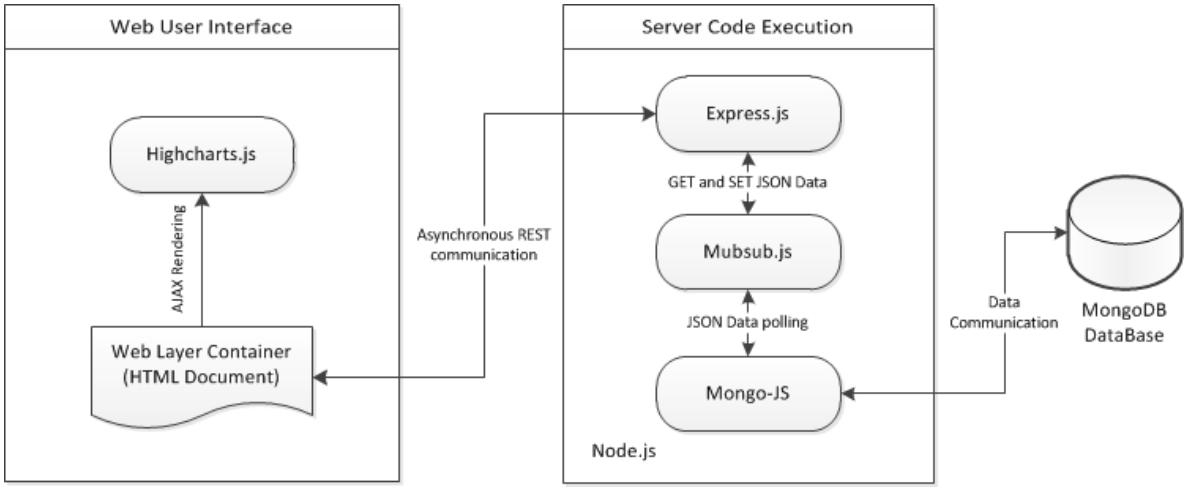

Figura 4-9. Estructura general del módulo Monitoring View

Como se puede observar en la figura 4-9, el módulo *Monitoring View* está distribuido en 2 sub módulos principales: *Web User Interface* y *Server Code Execution*, los cuales se describen a continuación.

4.2.3.1.1 *Web User Interface*

El sub módulo de la Interfaz de Usuario Web permite observar el comportamiento de los parámetros de QoS asociados a los servicios atómicos en tiempo real, a través de una interfaz web diseñada para tal propósito, luego de realizarse el procesamiento de datos en el lado del servidor.

La implementación de este sub modulo está basada en archivos de páginas web diseñadas bajo el estándar HTML5, las cuales pueden ser ejecutadas y visualizadas en cualquier navegador web moderno que soporte JavaScript, como por ejemplo, Firefox, Google Chrome o Internet Explorer [55]. Estos archivos contienen el código con las

instrucciones necesarias para conectarse al servidor a través de operaciones REST asíncronas, con el fin de consultar los valores de QoS que serán mostrados en la interfaz web.

Para graficar los valores obtenidos se hace uso de la API *Highcharts JS* [56], una librería escrita en JavaScript/HTML5, la cual permite la implementación de gráficos interactivos en sitios o aplicaciones web. De esta forma, las instrucciones para realizar los gráficos de los valores de QoS, son codificadas dentro de los archivos HTML5, así, estos valores son mostrados en tiempo real y de manera interactiva a medida que transcurre el tiempo.

4.2.3.1.2 Server Code Execution

El sub módulo donde se encuentra el servidor que ejecuta la lógica para el acceso a los datos almacenados en la base de datos descrita en la sección 4.2.2, está basado en Node.js [57], una plataforma basada en JavaScript, que permite diseñar y construir aplicaciones web rápidas y escalables desde la capa del servidor, la cual utiliza un modelo orientado a eventos. Node.js permite el uso de módulos que pueden ser definidos por el desarrollador o utilizarse como referencias externas durante la ejecución de una aplicación.

Para el desarrollo de este sub módulo se ha utilizado el módulo Express.js, un framework diseñado para realizar aplicaciones web utilizando Node.js, el cual provee un conjunto robusto de características para su construcción [58]. Mediante las funcionalidades que ofrece Express.js es posible conectarse fácilmente con el módulo Mubsub.js, el cual permite suscribirse de manera simple a colecciones de documentos pertenecientes a una determinada base de datos diseñada en MongoDB, de tal forma que los suscriptores son notificados cuando se insertan nuevos documentos en la base de datos, que coincidan con determinados parámetros definidos previamente por parte de dichos suscriptores [59], esto permite la obtención de datos en tiempo real, pues el módulo es notificado inmediatamente se encuentra una nueva coincidencia de acuerdo los datos definidos en la suscripción, permitiendo graficar los valores de QoS de manera dinámica, a medida estos datos se van recolectando por parte del mecanismo de monitoreo a través del módulo *Service Monitor*.

RESUMEN

En este capítulo se presentó la definición del mecanismo para el monitoreo de servicios convergentes en entornos JAIN SLEE. Inicialmente se abordó la descripción del funcionamiento de los servicios convergentes enmarcados dentro del proyecto TelComp2.0, resaltando el funcionamiento del flujo de ejecución de los mismos y la necesidad de contar con un mecanismo de control en tiempo de ejecución, que permita monitorear los puntos críticos de ese proceso.

Posteriormente se describió de manera detallada cada uno de los componentes asociados a los 3 módulos principales del mecanismo de monitoreo propuesto: *Monitoring Module, Monitoring Data* y *Monitoring View.*

La descripción del módulo de monitoreo se inicia abordando el proceso que soporta la detección del despliegue de nuevos servicios convergentes, lo cual da el paso a las labores de monitoreo asociadas a cada enfoque del mecanismo. Luego, se realiza una descripción detallada del funcionamiento de los 4 servicios que componen el módulo de monitoreo: *Code Injector, Service Monitor, QoS Monitor* y *Alarm Monitor*, a su vez que se muestran los algoritmos asociados al funcionamiento de cada uno de ellos.

En el módulo *Monitoring Data* inicialmente se realiza una aproximación acerca del uso de las bases de datos basadas en un modelo NoSQL, de tal manera que se pueda realizar una descripción de la estructura general de la base de datos implementada para su uso en el mecanismo de monitoreo propuesto en el presente trabajo de grado.

Finalmente se explica el funcionamiento del módulo *Monitoring View,* describiendo su estructura general, además de la aplicación web que soporta la presentación de gráficas de los valores de QoS en tiempo real a través de una interfaz web.

Capítulo 5

5. Implementación y Resultados

En este capítulo se busca evaluar experimentalmente a través de un prototipo, el monitoreo de servicios convergentes en entornos JAIN SLEE, para esto se exponen dos secciones principales, la primera hace referencia a la descripción del proceso de desarrollo del mecanismo , mientras que la segunda corresponde a la fase de evaluación y pruebas del mismo, teniendo en cuenta los dos enfoques de monitoreo de servicios que aborda: reactivo y proactivo. Dicha evaluación será realizada haciendo uso de un servicio convergente desplegado en un escenario de prueba con ciertos requerimientos de QoS, de tal manera que se pueda demostrar la aplicabilidad del mecanismo propuesto en este tipo de servicios.

5.1 Descripción del Prototipo

El prototipo construido consiste en un sistema compuesto por cuatro servicios desarrollados en lenguaje Java, bajo la especificación JAIN SLEE. Los cuales han sido implementados a partir de la adopción del Modelo de Construcción de Soluciones [60], como referencia metodológica para soportar el proceso de desarrollo. Las secciones siguientes, se centran en la aplicación de dos de los componentes más relevantes de dicho modelo: Estructura para la Descripción del Sistema y el Modelo del Proceso de Desarrollo. El primero de ellos involucra la especificación de los elementos más importantes, para un correcto entendimiento de la funcionalidad y el comportamiento esperado del sistema/solución (Modelo de Casos de Uso, Arquitectura de Referencia, Modelo de Despliegue y Plan de Pruebas); mientras que el modelo del Proceso de Desarrollo, define cuatro fases de referencia, las cuales, ejecutadas en conjunto de forma iterativa e incremental, orientan el proceso de desarrollo de la solución propuesta.

5.1.1 Modelo de Casos de Uso del Sistema

Nombre caso de uso	1. Detectar Servicio Convergente
Iniciador	Agente Externo: Aplicación
Propósito	Detectar el despliegue de un nuevo servicio convergente que será monitoreado.
Resumen	Una vez un servicio convergente ha sido compuesto y desplegado por medio de la plataforma de Telcomp2.0 desarrollada para ese fin, el sistema detecta la instalación y despliegue de un nuevo servicio convergente y activa los módulos correspondientes al monitoreo.

Tabla 5-1. Descripción caso de uso: Detectar Servicio Convergente.

Nombre caso de uso	2. Instrumentar Código
Iniciador	Agente Externo: Aplicación
Propósito	Instrumentar los servicios convergentes con el código de monitoreo.
Resumen	El sistema detecta los puntos de control en el flujo de ejecución del servicio convergente donde es necesario realizar un control del mismo, e instrumenta el código necesario para establecer una alarma en el caso de que se presente un fallo.

Tabla 5-2. Descripción caso de uso: Instrumentar Código.

Nombre caso de uso	3. Recolectar valores de QoS
Iniciador	Agente Externo: Aplicación
Propósito	Monitorear de manera periódica los valores de QoS brindados por los servicios atómicos que componen el servicio convergente.
Resumen	El sistema reconoce las WSDL pertenecientes a los servicios atómicos que componen el servicio convergente y las extrae para realizar peticiones a estas de manera periódica, con el fin de recolectar en tiempo real y almacenar en la base de datos valores de QoS asociados a la disponibilidad y el tiempo de respuesta.

Tabla 5-3. Descripción caso de uso: Recolectar valores de QoS.

Nombre caso de uso	4. Calcular métricas y reglas de QoS
Iniciador	Agente Externo: Aplicación
Propósito	Calcular las métricas y reglas de QoS definidas para la disponibilidad y tiempo de respuesta.
Resumen	El sistema consulta de manera periódica la base de datos donde se almacenan los valores de QoS asociados a la disponibilidad y tiempo de respuesta, y mediante las métricas definidas para cada propiedad, calcula los valores de estas.

Tabla 5-4. Descripción caso de uso: Calcular métricas de QoS.

Nombre caso de uso	5. Generar Alarmas
Iniciador	Agente Externo: Aplicación
Propósito	Generar alarmas en el caso de que un parámetro de QoS no cumpla con los valores establecidos.
Resumen	El sistema verifica que los valores calculados por las métricas de cada parámetro cumplan con los requerimientos establecidos, en caso de que no sea así, se establece una alarma en el servidor de aplicaciones, la cual contiene información acerca del servicio que presenta el fallo.

Tabla 5-5. Descripción caso de uso: Generar Alarmas.

Nombre caso de uso	6. Informar Fallos
Iniciador	Agente Externo: Aplicación
Propósito	Informar al módulo de recuperación acerca de la detección de una nueva alarma.
Resumen	El sistema monitorea de manera periódica las alarmas en el servidor de aplicaciones, con el fin de detectar cuándo se establece una alarma, para inmediatamente informar al módulo de recuperación que se ha presentado un fallo en la ejecución del servicio convergente o se ha excedido un valor de QoS permitido en los servicios atómicos.

Tabla 5-6. Descripción caso de uso: Informar Fallos.

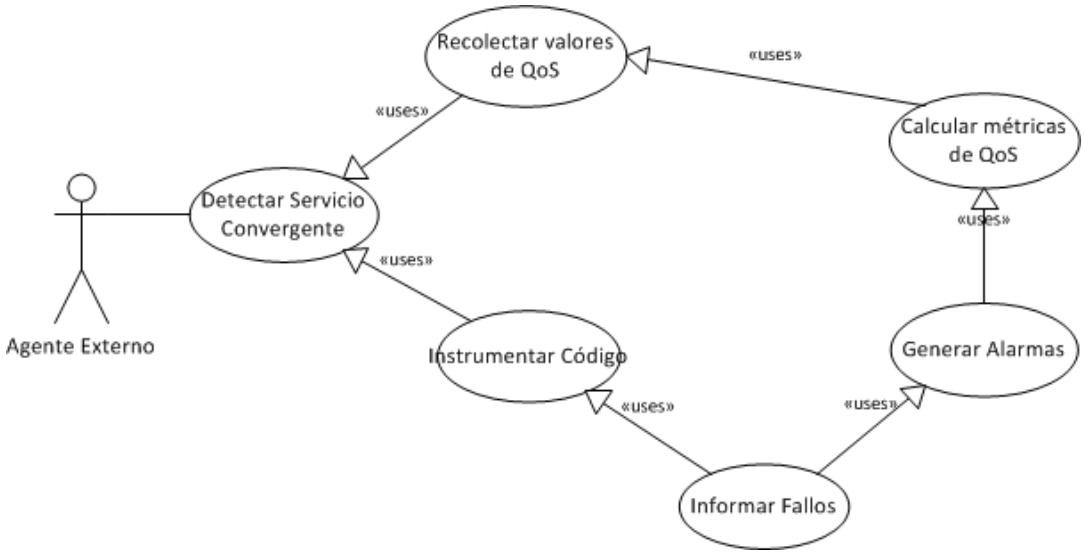

Figura 5-1. Diagrama de Casos de Uso del Sistema.

Adicional a los casos de uso definidos, es pertinente definir una arquitectura de referencia, que proporcione una visión general de los componentes del sistema. En la siguiente sección, se describe la arquitectura propuesta para el sistema implementado.

5.1.2 Descripción de la Arquitectura de Referencia

El prototipo desarrollado para el mecanismo de monitoreo de servicios convergentes propuesto en este documento, adopta una arquitectura de aplicación en tres capas: la capa de Lógica de Presentación, la capa de Lógica de Negocio y una capa de Soporte. La figura 5-2, presenta un diagrama de la arquitectura de referencia del sistema.

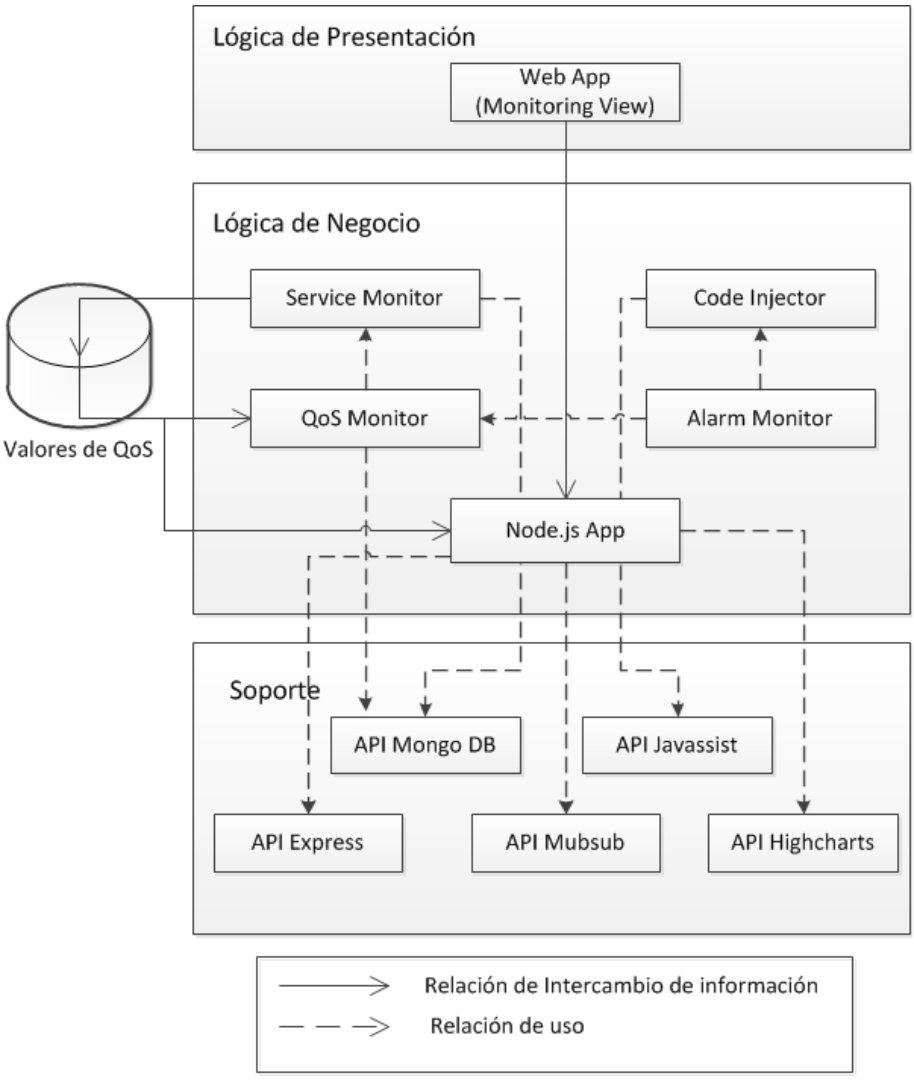

Figura 5-2. Arquitectura de Referencia del Sistema.

5.1.2.1 Soporte

Esta capa inferior de la arquitectura está constituida por un conjunto de herramientas software, que hacen posible el proceso del monitoreo de servicios convergentes.

5.1.2.1.1 Software de implementación de la capa de soporte

Como sistema operativo host fue elegida una distribución de Linux, en este caso Ubuntu 13.04, el cual permite la ejecución de todas las APIs y módulos mencionados anteriormente. Respecto al entorno de desarrollo, fue seleccionado Eclipse Juno como el IDE para la elaboración de los servicios que componen el mecanismo de monitoreo. Adicionalmente se tuvieron en cuenta las siguientes consideraciones para su escogencia:

1. Es un IDE multiplataforma implementado y desarrollado en lenguaje Java.
2. Las APIs mencionadas como *Javassist, API MongoBD,* están disponibles para trabajarlas como librerías en dicho entorno.
3. Es un IDE de carácter libre, además permite la integración con el plugin *EclipSLEE*, el cual brinda las funcionalidades necesarias para la creación de proyectos basados en la especificación JAIN SLEE.

A continuación se describen especificaciones técnicas de las herramientas usadas en el sistema implementado.

Herramienta	Versión
JDK	1.7.0.45
Mobicents JSLEE	2.7.0
Mongo DB	2.4.9

Tabla 5-7. Especificaciones del sistema.

5.1.2.2 Lógica de Negocio

En esta capa se implementan y articulan, todos los componentes necesarios para el funcionamiento del mecanismo de monitoreo de servicios convergentes. Este nivel de arquitectura es compuesto por 5 módulos fundamentales: *Code Injector, Service Monitor, QoS Monitor, Alarm Monitor y* la aplicación desarrollada en *Node.js*, los cuales interactúan con la capa de Lógica de Presentación, la capa de Soporte y la Base de Datos, los cuales fueron explicados de manera detallada en el Capítulo 4.

5.1.2.3 Lógica de Presentación

Esta capa fue desarrollada para propósitos de observación por parte de un usuario administrador del sistema, de tal forma que pueda conocer el funcionamiento del módulo *Service Monitor* de manera gráfica y en tiempo real, como se mencionó en la descripción de la lógica de negocio, este proceso es llevado a cabo a través de la aplicación desarrollada en Node.js y es visualizado mediante la interfaz web desarrollada en HTML5.

5.1.3 Diagrama de Despliegue

En la figura 5-1 se observan los componentes del sistema, dentro de los cuales existen dos servidores: un servidor Web y un SLEE; adicionalmente se encuentra la base de datos y el browser correspondiente a la parte de vista del sistema de monitoreo.

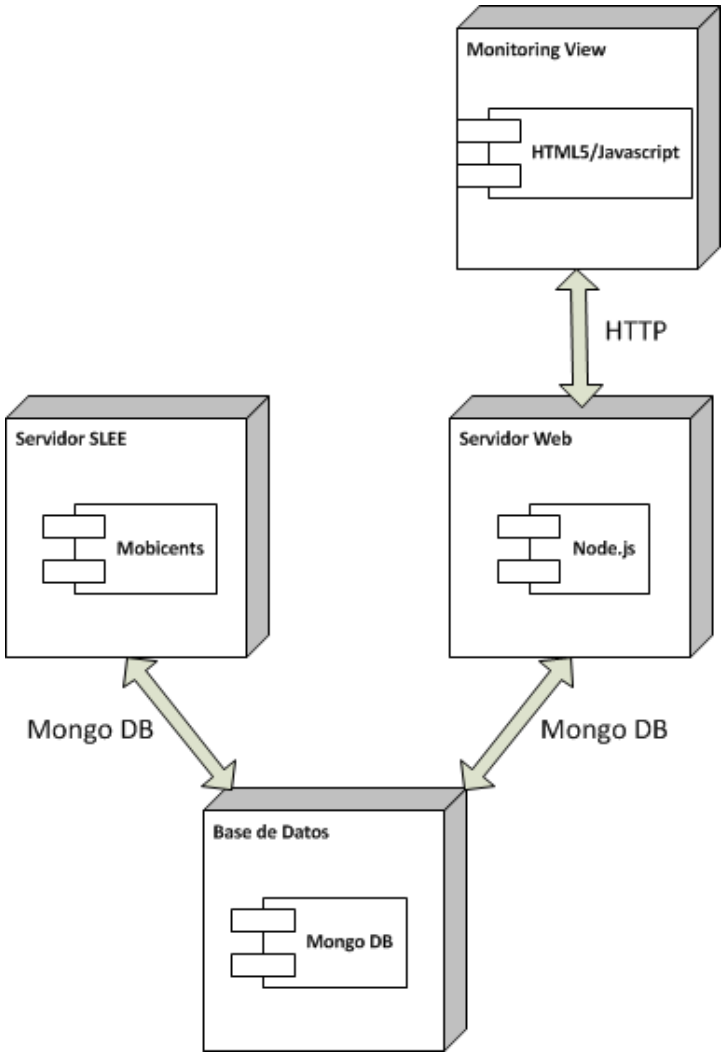

Figura 5-3. Diagrama de Despliegue del sistema

- ✓ **SLEE:** es el componente principal de la arquitectura de negocio, puesto que gestiona la lógica, el acceso y el ciclo de vida de los servicios que componen el mecanismo de monitoreo. Como se mencionó anteriormente el SLEE seleccionado para esta arquitectura es Mobicents.

- ✓ **Servidor Web:** es un servidor de aplicaciones web, donde se encuentra alojada la aplicación que permite acceder a los valores de QoS en tiempo real. Node.js [57] fue escogido como el servidor web debido a su carácter libre y a su arquitectura orientada a eventos.

- ✓ **Base de Datos:** utilizada para almacenar la información que sobre los valores de QoS que es capturada en tiempo real por medio del módulo *Service Monitor*, creando una capa de acceso a datos para el sistema, que además es accedida desde la aplicación Web. El sistema de base de datos escogido para este fin, ha sido MongoDB.

- ✓ **Monitoring View:** mediante una interfaz amigable, muestra en tiempo real gráficas asociadas a los valores de QoS que están siendo monitoreados, para este fin se usa HTML5 y JavaScript.

5.2 Pruebas del Prototipo

Una vez concluida la implementación del prototipo, es necesario someterlo a un proceso de evaluación, con el fin de garantizar su óptimo desempeño. Las actividades de evaluación funcional se realizan de acuerdo a los dos enfoques de monitoreo definidos en la sección 4.2: Reactivo y Proactivo, usando para esto los componentes principales de la propuesta: *Code Injector*, *Service Monitor* y *QoS Monitor* y *Alarm Monitor*, mientras que la evaluación de rendimiento se realiza con el fin de determinar el impacto del mecanismo de monitoreo en el funcionamiento del entorno de ejecución donde han sido desplegado. De esta forma, el objetivo general de esta evaluación, consiste en determinar dos aspectos esenciales: i) validación de las características funcionales del mecanismo de monitoreo propuesto y ii) rendimiento del sistema.

5.2.1 Evaluación Funcional

Para la evaluación funcional del mecanismo propuesto, se hace uso de un servicio convergente llamado *LinkedIn Job* Notificator, el cual ha sido desarrollado de acuerdo a los lineamientos definidos para este tipo de servicios en la plataforma TelComp2.0 y se describe a continuación.

5.2.1.1 Servicio *LinkedIn Job Notificator*

El servicio *LinkedIn Job Notificator* toma una oferta de trabajo asociada al usuario, la cual está actualmente registrada en su cuenta de LinkedIn, y la envía a su red social Twitter, al correo personal, y al celular a través de una llamada de voz, transformando el texto de la oferta de trabajo en un mensaje de audio.

El flujo de control de la Figura 5-4 define el orden de ejecución de los servicios atómicos que componen el servicio convergente. La estructura de control está compuesta de dos servicios Web (LinkedIn, Twitter), de dos servicios Telco (Email, Media Call), de un servicio de acceso a datos (DataAccess), y de patrones de control (Trigger, Sequence, Parallel Split, Synchronization).

El servicio inicia cuando el usuario activa la aplicación desde su dispositivo móvil. El sistema reacciona a la petición haciendo uso del servicio DataAccess para obtener el identificador del usuario en la red social LinkedIn (paso 1 de ejecución); después de realizada la consulta, el sistema invoca el servicio LinkedIn-WS con el fin de obtener las ofertas de trabajo registradas en esa red social (paso 2 de ejecución); una vez se tenga

respuesta, se consulta tanto el identificador de la red social Twitter como el identificador de correo electrónico del usuario (paso 4 de ejecución), ambos necesarios para el envío de las ofertas de trabajo a los medios respectivos (paso 5 de ejecución). Posteriormente, las respuestas de los servicios Email Telco Service y Twitter-WS se sincronizan para dar inicio al servicio de notificación por voz (paso 6 de ejecución). Finalmente, se consulta en un repositorio el número móvil del usuario y se procede hacer la llamada de voz indicándole, a través de un mensaje de audio, las ofertas de trabajo disponibles (pasos 7 y 8 de ejecución).

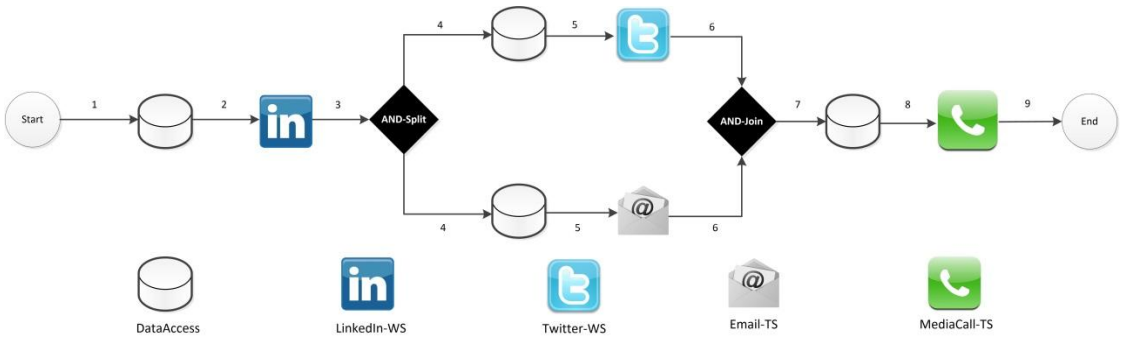

Figura 5-4. Flujo de ejecución del servicio *LinkedIn Job Notificator Service*

5.2.1.2 Enfoque Reactivo del Mecanismo de Monitoreo

El módulo *Code Injector* corresponde al enfoque reactivo del mecanismo de monitoreo propuesto, a continuación se listan las pruebas funcionales ejecutadas para verificar el correcto funcionamiento del mecanismo de monitoreo desarrollado, teniendo en cuenta este tipo de enfoque de monitoreo.

Plan de Pruebas	
Detección de la ejecución de un nuevo Servicio Convergente	**Prueba Funcional 1:** permite comprobar la detección de un nuevo servicio convergente, una vez este haya sido instalado y desplegado en el Servidor de Aplicaciones de Mobicents, de tal forma que se pueda dar paso a la función de instrumentación de código.
Instrumentación de Código	**Prueba Funcional 2**: permite evaluar la funcionalidad de la instrumentación de código con funciones de monitoreo en los servicios convergentes.
Establecimiento de Alarma	**Prueba Funcional 3:** permite comprobar que el servicio convergente instrumentado con la funcionalidad de monitoreo, establece una Alarma en caso de fallo, se lleve a cabo de manera correcta.

Tabla 5-8. Pruebas enfoque reactivo del mecanismo de monitoreo

Prueba Funcional 1: la ejecución de esta prueba permite determinar sí el mecanismo propuesto es capaz de detectar la instalación y despliegue de un nuevo servicio convergente, diferenciando este de un servicio atómico web o telco que también pueda ser instalado en el Servidor de Aplicaciones de Mobicents JSLEE.

Inicialmente, el servicio *Code Injector* es instalado y desplegado en el Servidor de Aplicaciones de Mobicents JSLEE, el cual estará esperando por el despliegue de nuevos servicios convergentes. Luego, se despliega un servicio atómico para comprobar que efectivamente el mecanismo es capaz de diferenciar este tipo de servicios de un servicio convergente. Para este caso, es desplegado el servicio atómico Telco *llamado Email Telco Service,* el cual hace parte de la composición del servicio convergente de prueba, *LinkedIn Job Notificator.* La figura 5-5 muestra la captura de la consola de registro de Mobicents JSLEE, donde se comprueba que el servicio instalado es un servicio atómico y no debe ser instrumentado con tareas de monitoreo.

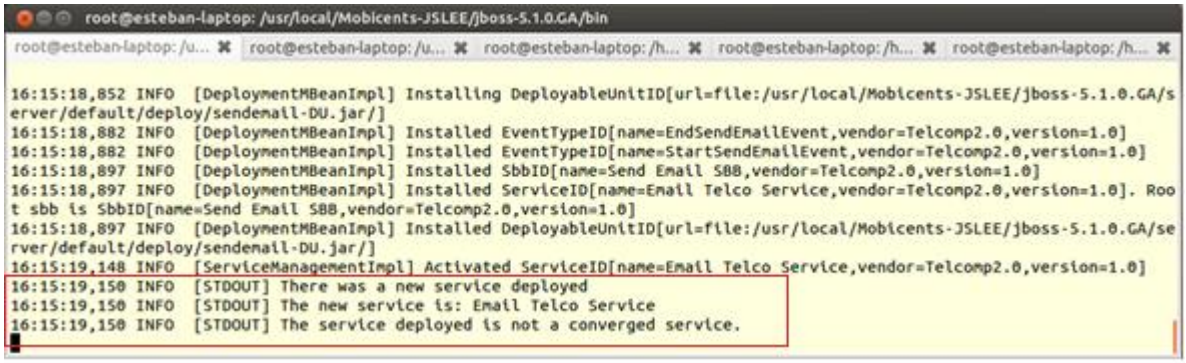

Figura 5-5. Consola de Administración Mobicents JSLEE, despliegue servicio atómico

Posteriormente, se realiza el despliegue del servicio convergente de prueba *LinkedIn Job Notificator*, el cual permitirá comprobar que el mecanismo de monitoreo constata que el servicio instalado es un nuevo servicio convergente, de esta forma, el servicio *Cojde Injector*, de inmediato realiza el proceso de análisis e instrumentación del código con las funciones de monitoreo, el cual tarda solo 40 ms en ser implementado en el servicio convergente. Sin embargo, debido a los cambios en la clase orquestadora del servicio, es necesario realizar un re-despliegue del servicio, el cual es ejecutado de manera automática por el Servidor de Aplicaciones de Mobicents JSLEE, incrementando en el proceso en un tiempo promedio de 6 segundos, después de esto, el servicio convergente instrumentado con funciones de monitoreo, está listo para procesar las solicitudes por parte del usuario.

La figura 5-6 muestra la captura de la consola de registro de Mobicents JSLEE, donde se comprueba el proceso mencionado.

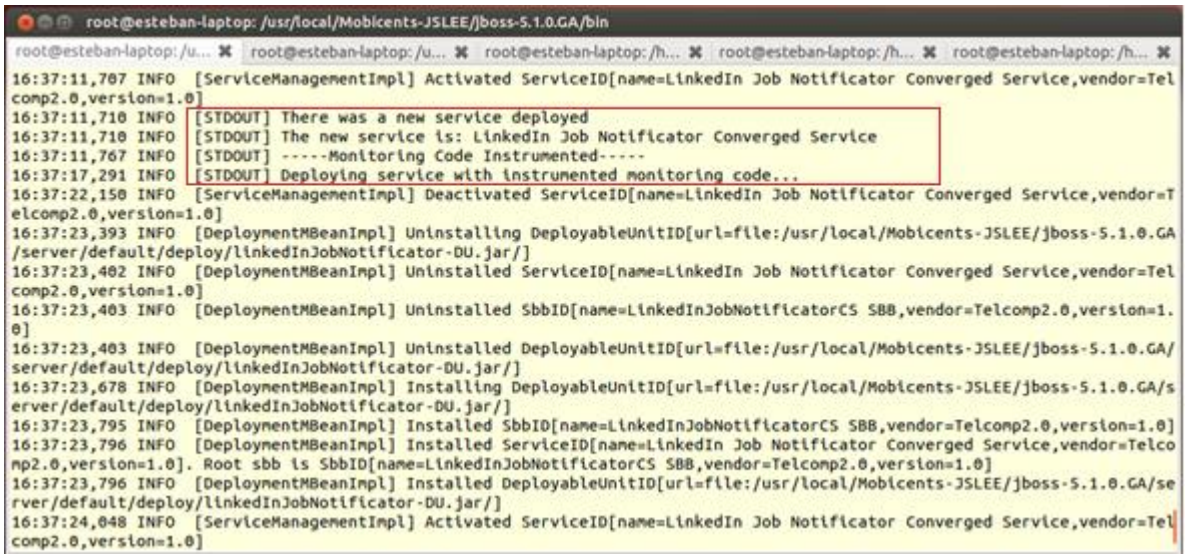

Figura 5-6. Consola de Administración Mobicents JSLEE, despliegue servicio convergente

Prueba Funcional 2: la ejecución de esta prueba permite comprobar la funcionalidad que instrumenta el código de monitoreo dentro de los servicios convergentes.

Una vez el servicio *Code Injector* ha instrumentado los métodos de control de la clase SBB orquestadora con el código de monitoreo que permite emitir una alarma en caso de que se presente una falla en los puntos críticos del flujo de ejecución del servicio convergente, la unidad desplegable instalada en el Servidor de Aplicaciones de Mobicents JSLEE, contiene la nueva clase SBB orquestadora compilada. Las figuras 5-7 y 5-8 presentan respectivamente un fragmento de la clase SBB orquestadora, antes y después de ser instrumentada con el código de monitoreo que maneja las respuestas de los servicios atómicos, en este caso del servicio web *LinkedIn-WS,* el cual corresponde al segundo paso de ejecución del servicio convergente, estas capturas se pueden llevar a cabo a través de la herramienta online llamada *Java Decompiler* [61], la cual permite observar el código de una clase Java compilada.

```
133. public void onEndWSInvocatorEvent(EndWSInvocatorEvent event, ActivityContextInterface aci)
134. {
135.  if(mainControlFlow == 2)
136.  {
137.   system.out.println((new StringBuilder("EndWebService Event received on execution step: ")).append(mai
138.   mainAci = aci;
139.   mainControlFlow = 3;
140.   ws0opv0 = (list)event.getOperationOutputs().get(ws0opn0);
141.   mainControlFlow = 4;
142.   andSplit1Aci = createNullActivityACI();
143.   andSplit1Aci.attach(sbbContext.getSbbLocalObject());
144.   andSplit2Aci = createNullActivityACI();
145.   andSplit2Aci.attach(sbbContext.getSbbLocalObject());
146.   branchControlFlow1 = 1;
147.   ts2ip0 = startpv0;
148.   hashmap operationInputs1 = new hashmap();
149.   operationInputs1.put("parameter", ts2ip1);
150.   operationInputs1.put("identification", ts2ip0);
151.   StartGetDataTelcoServiceEvent dataAccessEvent1 = new StartGetDataTelcoServiceEvent(operationInputs1);
152.   fireStartGetDataTelcoServiceEvent(dataAccessEvent1, andSplit1Aci, null);
153.   branchControlFlow2 = 1;
154.   ts1ip0 = startpv0;
155.   hashmap operationInputs2 = new hashmap();
156.   operationInputs2.put("parameter", ts1ip1);
157.   operationInputs2.put("identification", ts1ip0);
158.   StartGetDataTelcoServiceEvent dataAccessEvent2 = new StartGetDataTelcoServiceEvent(operationInputs2);
159.   fireStartGetDataTelcoServiceEvent(dataAccessEvent2, andSplit2Aci, null);
160.  } else
161.  if(mainControlFlow == 4)
162.  {
```

Figura 5-7. Clase SBB Orquestadora original

```
139. public void onEndWSInvocatorEvent(EndWSInvocatorEvent endwsinvocatorevent, ActivityContextInterface activitycontextinte
140. {
141.  if(!endwsinvocatorevent.isSuccess())
142.   try
143.   {
144.       object obj = org/telcomp/sbb/LinkedInJobNotificatorCSSbb;
145.       for(int i = 1; i <= branchSize2; i++)
146.       {
147.           field field = ((class) (obj)).getDeclaredField("branchControlFlow" + i);
148.           valorMensaje2 = valorBranch2 + string.valueOf(field.getInt(obj)) + ";";
149.           valorBranch2 = valorMensaje2;
150.       }
151.
152.       alarmFacility2.raiseAlarm("javax.slee.management.alarm.sbb", "01", AlarmLevel.MAJOR, ";LinkedInJobNotificator;");
153.
154.   }
155.   catch(exception exception) { }
156.  if(mainControlFlow == 2)
157.  {
158.      system.out.println((new StringBuilder()).append("EndWebService Event received on execution step: ").append(mainCon
159.      mainAci = activitycontextinterface;
160.      mainControlFlow = 3;
161.      ws0opv0 = (list)endwsinvocatorevent.getOperationOutputs().get(ws0opn0);
162.      mainControlFlow = 4;
163.      andSplit1Aci = createNullActivityACI();
164.      andSplit1Aci.attach(sbbContext.getSbbLocalObject());
165.      andSplit2Aci = createNullActivityACI();
166.      andSplit2Aci.attach(sbbContext.getSbbLocalObject());
```

Figura 5-8. Clase SBB Orquestadora instrumentada

Como se observa en la Figura 5-8, mediante las líneas de código insertadas (141-155) el método *onEndWSInvocatorEvent* ahora cuenta con una funcionalidad para establecer una Alarma en caso de que la ejecución del servicio atómico *LinkedIn-WS* no finalice de forma correcta.

Prueba Funcional 3: la ejecución de esta prueba permite comprobar que el servicio convergente dotado con funciones de monitoreo, establece una Alarma, inmediatamente se detecta un fallo en la ejecución normal del servicio convergente.

Para el desarrollo de esta prueba se establece un escenario no deseado en el cual el servicio *LinkedIn-WS* no se encuentra disponible, con el fin de comprobar que el proceso de establecimiento de la Alarma y la visualización de esta dentro de la consola de gestión de Mobicents JSLEE son realizadas correctamente. Para ello, se deshabilita el servicio atómico en el servidor del Proyecto TelComp2.0 donde se está ejecutando y se procede a invocar el servicio *LinkedIn Job Notificator*. Como era de esperarse, una vez invocado el servicio convergente, este inicia su ejecución de manera normal hasta que llega al paso 2 de ejecución, momento en que se presenta el fallo y una Alarma con información acerca de este, es establecida gracias a la nueva funcionalidad agregada con el código instrumentado. La Figura 5-9 muestra la Alarma activada en la consola de gestión de Mobicents JSLEE

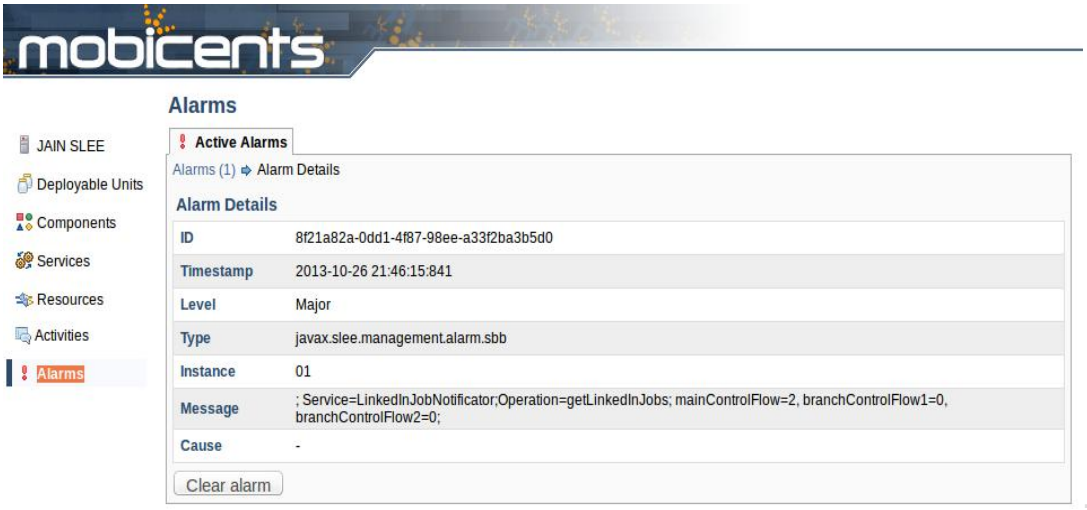

Figura 5-9. Alarma activada en consola de gestión de Mobicents JSLEE

5.2.1.3 Enfoque Proactivo del Mecanismo de Monitoreo

Los módulos *Service Monitor* y *QoS Monitor,* corresponden al enfoque proactivo del mecanismo de monitoreo propuesto, a continuación se listan las pruebas funcionales ejecutadas para verificar el correcto funcionamiento del mecanismo de monitoreo desarrollado, teniendo en cuenta este tipo de enfoque de monitoreo.

Plan de Pruebas	
Recolección valores de QoS	**Prueba Funcional 4:** permite comprobar que el sistema reconoce las WSDL que pertenecen a los servicios atómicos que componen el servicio convergente y realiza peticiones a estos con el fin de recolectar valores de QoS y guardarlos en la base de datos diseñada para este fin.
Calculo de métricas y reglas de QoS	**Prueba Funcional 5**: permite comprobar que el sistema calcula de manera correcta las métricas y reglas de QoS definidas para la disponibilidad y el tiempo de respuesta, basado en los valores de estos que han sido almacenados en la base de datos por parte del módulo *Service Monitor.*
Establecimiento de Alarma	**Prueba Funcional 6:** permite comprobar que en caso de una violación de las reglas de QoS establecidas para la disponibilidad y tiempo de respuesta, se genera una alarma en el servidor de aplicaciones, la cual contiene información sobre el fallo.

Tabla 5-9. Pruebas enfoque proactivo del mecanismo de monitoreo

Prueba Funcional 4: la ejecución de esta prueba permite comprobar la funcionalidad que recolecta valores de QoS en tiempo real y los almacena en la base de datos. Esta funcionalidad es implementada por parte del módulo *Service Monitor*, una vez es detectada la instalación de un nuevo servicio convergente, proceso que se demostró en la Prueba Funcional 1.

Mediante la introspección de código se reconocen las WSDL pertenecientes a los servicios atómicos que componen el servicio *LinkedIn Job Notificator,* en este caso de los servicios Web (LinkedIn, Twitter), y de los servicios Telco (Email, Media Call). Se obtienen dichas WSDL y tal como se describió en la sección 4.2.1.3, el sistema inicia a realizar peticiones HTTP de manera periódica, con el fin de almacenar tiempos de respuesta y valores de disponibilidad para cada uno de ellos. Este proceso es llevado a cabo cada 10 segundos. La figura 5-10 muestra la consola de registro de Mobicents JSLEE, en el momento en que se realizan las peticiones HTTP a las WSDL de los servicios atómicos, mientras que la figura 5-11 muestra la respuesta recibida con la información asociada a los valores de disponibilidad y tiempo de respuesta, los cuales serán almacenados en la base de datos.

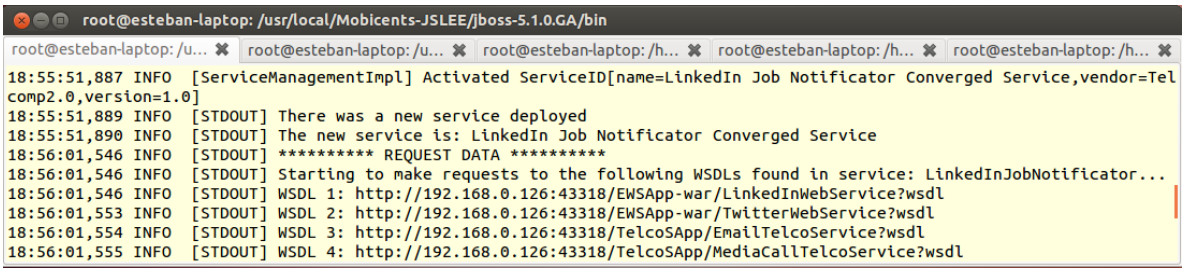

Figura 5-10. Envío de peticiones a las WSDL de servicios atómicos

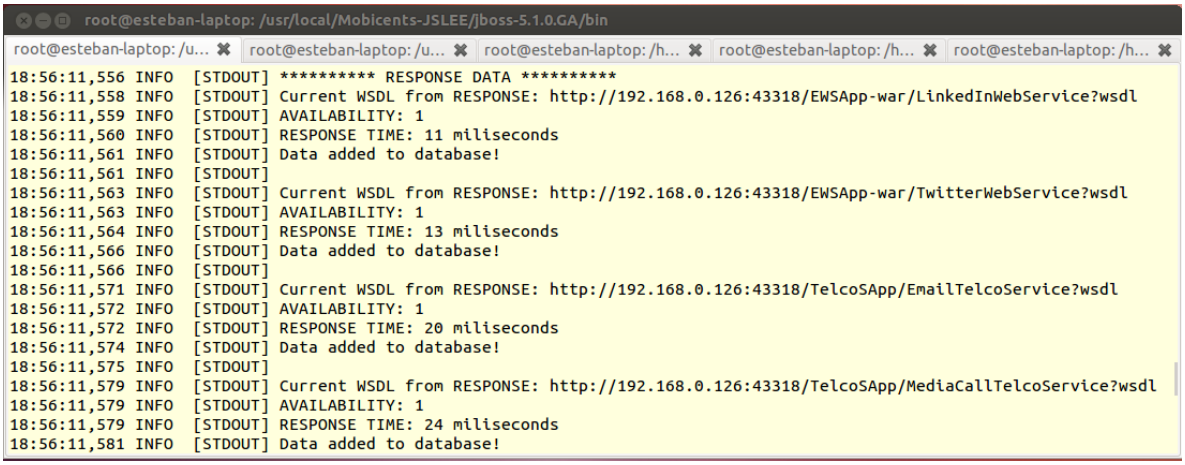

Figura 5-11. Recepción de respuesta de peticiones a las WSDL de servicios atómicos

De esta forma, dichos valores son almacenados en la base de datos creada en MongoDB. A través de una herramienta para la gestión de bases de datos implementadas en MongoDB, llamada *Robomongo* [62], es posible leer a través de una interfaz gráfica, los documentos que han sido almacenados. La figura 5-12 muestra una captura de *Robomongo,* en la cual se pueden observar los valores de QoS que han sido guardados en la base de datos por parte del módulo *Service Monitor*.

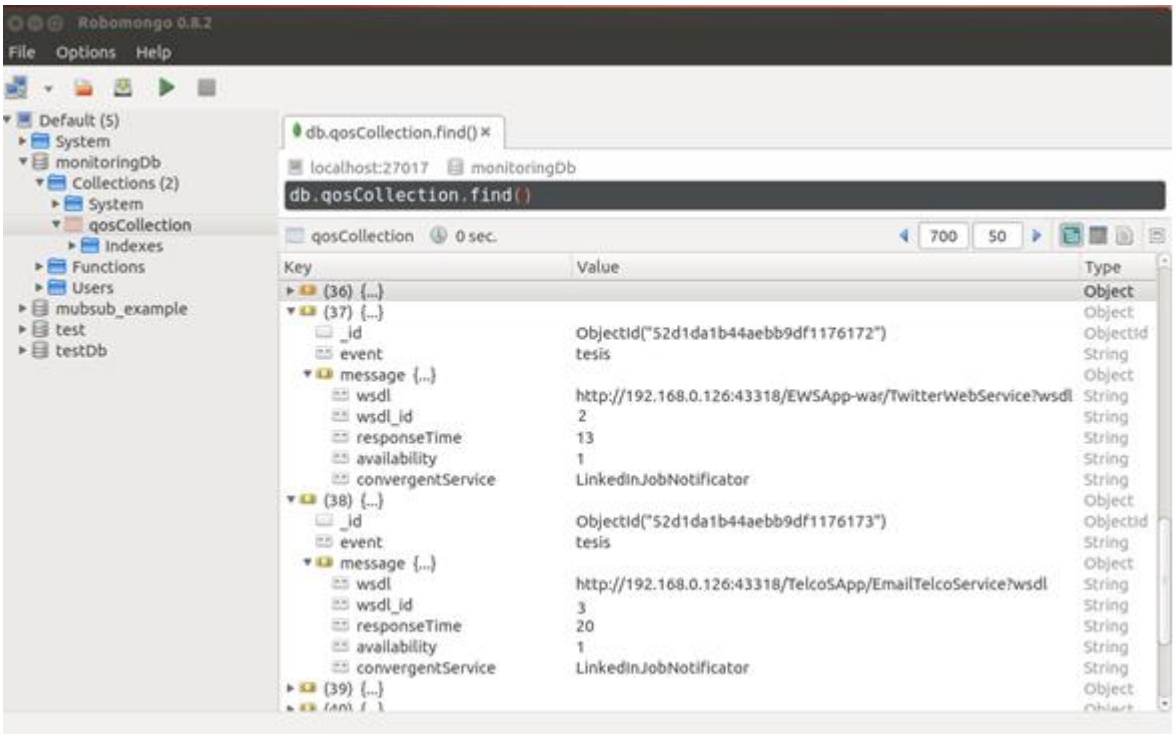

Figura 5-12. Base de datos en MongoDB con valores de QoS almacenados

Prueba Funcional 5: la ejecución de esta prueba permite comprobar la funcionalidad que calcula las métricas y reglas de QoS de acuerdo a los valores de disponibilidad y tiempo de respuesta recolectados en la base de datos.

Para esta prueba se plantea un escenario donde se establecen ciertos requerimientos en los valores de QoS y de acuerdo a las métricas y reglas propuestas en la sección 3.2, el módulo *QoS Monitor* calcula los valores de estas.

Los servicios atómicos que componen al servicio de prueba *LinkedIn Job Notificator* estarán enumerados así:

- ✓ Servicio Atómico 1 (*Sat1*): *LinkedIn Web Service*
- ✓ Servicio Atómico 2 (*Sat2*): *Twitter Web Service*
- ✓ Servicio Atómico 3 (*Sat3*): *Email Telco Service*
- ✓ Servicio Atómico 4 (*Sat4*): *Media Call Telco Service*

Luego, se define un escenario en donde se tienen los siguientes requerimientos de QoS:

- ✓ Requerimiento de Disponibilidad: $\geq 90\%$
- ✓ Requerimiento de Tiempo de respuesta: $\leq 80\ ms$

Posteriormente, con base en las ecuaciones (5) y (6), las reglas para el monitoreo de los valores de disponibilidad y tiempo de respuesta respectivamente, estarán definidas así:

$$D_{SC1} = \prod_{k=1}^{n} D_{Satk} \geq 90\%$$

$$TR_{SC1} = \sum_{k=1}^{n} TR_{Satk} \leq 80\ ms$$

Las muestras de disponibilidad y tiempo de respuesta de cada servicio atómico son recolectadas y almacenadas en la base de datos cada 10 segundos por parte del módulo *Service Monitor*, de esta forma, cada 5 minutos, el módulo *QoS Monitor* calcula la disponibilidad y tiempo de respuesta para cada servicio atómico con base en las muestras acumuladas durante dicho periodo de tiempo. La figura 5-13 muestra la consola de registro de Mobicents JSLEE, en donde se pueden observar los valores de los parámetros de QoS calculados para cada servicio atómico.

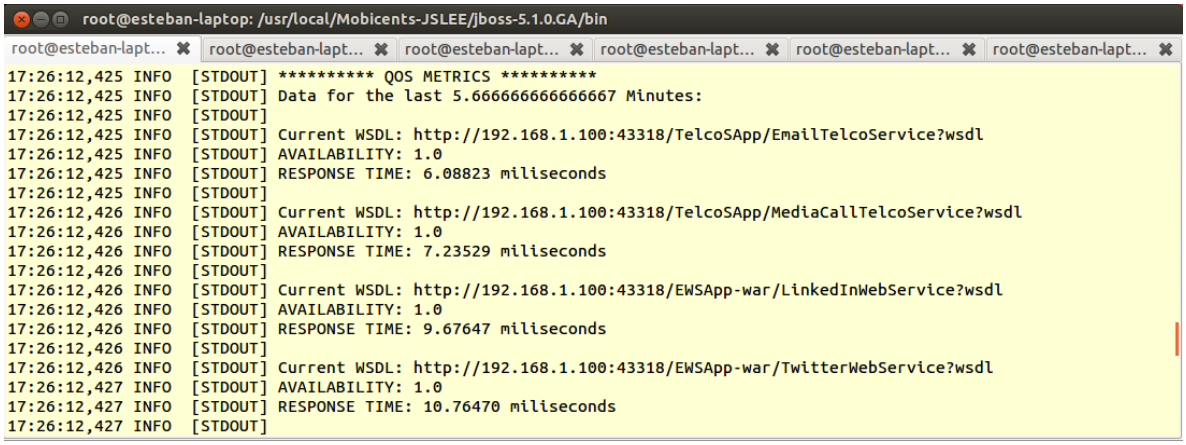

Figura 5-13. Calculo de parámetros de QoS por parte del módulo *QoS Monitor*

Una vez se calculan estos valores, se procede a verificar el cumplimiento de las reglas donde se establece que la disponibilidad del servicio *LinkedIn Job Notificator* debe ser mayor o igual al 90% y el tiempo de respuesta menor o igual a 60 ms. La figura 5-14 muestra la consola de registro de Mobicents JSLEE, en donde se pueden observar los valores obtenidos de acuerdo a las reglas establecidas.

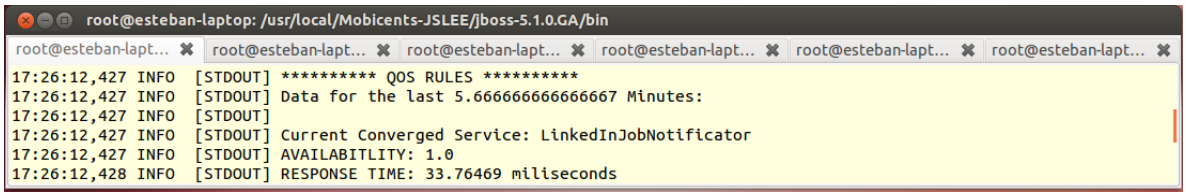

Figura 5-14. Valores de QoS para el servicio convergente

Prueba Funcional 6: la ejecución de esta prueba permite comprobar la funcionalidad que establece una alarma en el caso de que se presente una violación en los valores de QoS establecidos para la disponibilidad y el tiempo de respuesta.

Para el desarrollo de esta prueba se plantea un escenario no deseado, en el cual se presentan violaciones en los parámetros de QoS establecidos para la disponibilidad. En esta prueba se conservan los requerimientos establecidos en la Prueba Funcional 5 y se establece un tiempo aproximado de 30 minutos para la recolección muestras por parte del *Service Monitor*, de tal forma que teniendo en cuenta que estas son recogidas cada 10 segundos, se tendrán un total de 180 muestras para calcular las métricas y posteriormente las reglas de QoS.

En este escenario se afecta de manera intencional la disponibilidad del servicio web *Twitter-WS*, deshabilitando del servidor, la unidad desplegable por un periodo de tiempo aproximado de 5 minutos, de tal forma que el valor de la disponibilidad para ese servicio atómico en el momento en que se calcula con base en las 180 muestras recolectadas, tendrá un valor menor al 90%, dando como resultado una violación en el requerimiento establecido, una vez que la regla de QoS sea calculada. La figura 5-15 muestra la consola de registro de Mobicents JSLEE, en donde se puede observar el cálculo de los valores de QoS para el escenario no deseado número 1.

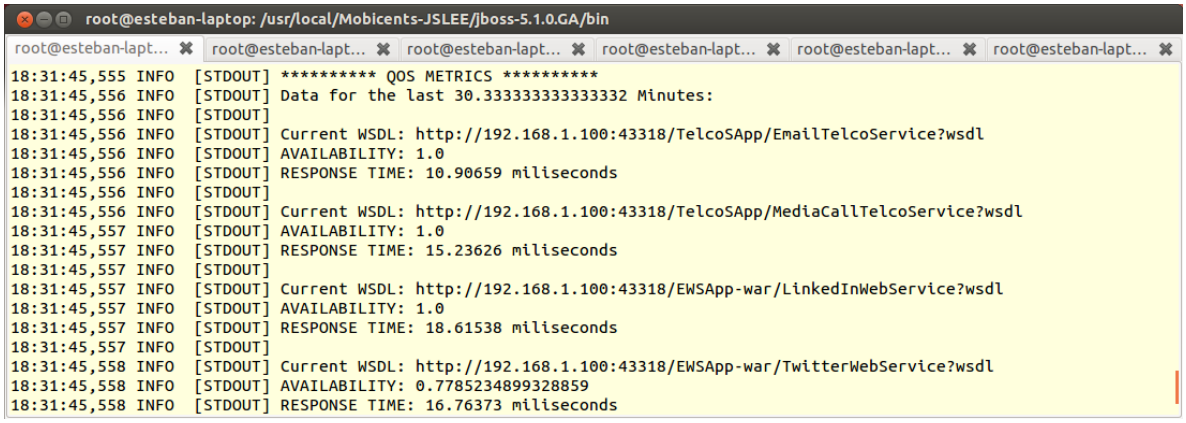

Figura 5-15. Calculo de parámetros de QoS para el escenario no deseado

Luego, basado en lo anterior, el módulo *QoS Monitor* procederá a calcular las reglas y como es de esperarse, la disponibilidad para el servicio *LinkedIn Job Notificator* en este caso es menor al 90%. La figura 5-16 muestra el valor obtenido para las reglas de QoS en este escenario.

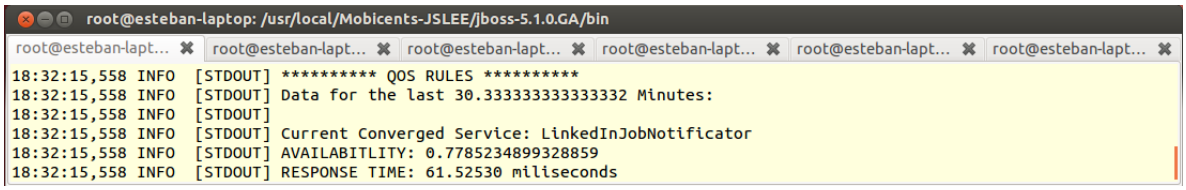

Figura 5-16. Valores de QoS para el escenario no deseado

Una vez se detecta que una regla no cumple con los requerimientos establecidos para esta, se activa una Alarma con información acerca del parámetro que presenta el detrimento en los valores de QoS y los valores del mismo para cada servicio atómico en el momento en que se presentó la falla, de tal forma que el módulo de reconfiguración pueda contar con la información necesaria para tomar las medidas pertinentes y corregir la falla. La figura 5-17 muestra la Alarma activa en la consola de gestión de Mobicents JSLEE.

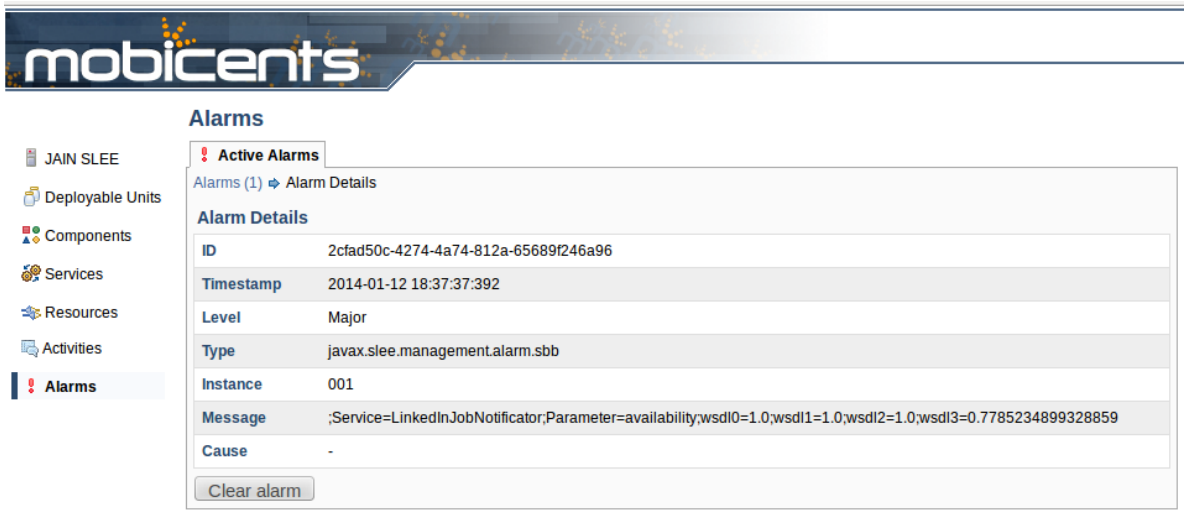

Figura 5-17. Alarma activada en consola de gestión de Mobicents JSLEE

5.2.1.4 Módulo Alarm Monitor y conexión con el Módulo de Reconfiguración

El módulo *Alarm Monitor*, está vinculado a los dos enfoques de monitoreo considerados en el mecanismo propuesto, ya que es el encargado de detectar nuevas alarmas establecidas, independientemente de la fuente que las originó. A continuación se describe la prueba funcional ejecutada sobre este módulo del prototipo desarrollado.

Plan de Pruebas	
Detección de nuevas Alarmas	**Prueba Funcional 7:** permite comprobar que el sistema monitorea de manera periódica las Alarmas en el servidor de aplicaciones, de tal forma que inmediatamente se detecte la activación de alguna, se informe de esta al módulo de recuperación.

Tabla 5-10. Pruebas módulo *Alarm Monitor*

Prueba Funcional 7: la ejecución de esta prueba permite comprobar la funcionalidad que detecta el establecimiento de nuevas Alarmas en el servidor de aplicaciones de Mobicents JSLEE, las cuales pueden ser originadas por dos fuentes distintas, la primera por parte del propio servicio convergente que está siendo monitoreado, en caso de que se presente un fallo en el flujo de ejecución del mismo, gracias a la funcionalidad agregada con el código instrumentado; la segunda por parte del módulo *QoS Monitor*, en el momento en que se presenta una violación en valores de los parámetros de QoS.

Independientemente de la fuente que origine la Alarma, el *Alarm Monitor* realiza el mismo procedimiento para informar al módulo de recuperación en caso de que se presente un fallo. Para la presente prueba se hará uso de una Alarma establecida por parte del *LinkedIn Job Notificator,* asociada a un fallo en su flujo normal de ejecución. Inmediatamente esta es establecida, los datos asociados a esta se organizan dentro de un HashMap y son enviados al módulo de reconfiguración a través de un evento personalizado para tal fin. Las figuras 5-18 y 5-19, muestran la consola de registro de Mobicents JSLEE, donde se puede observar el envío y recepción del evento respectivamente.

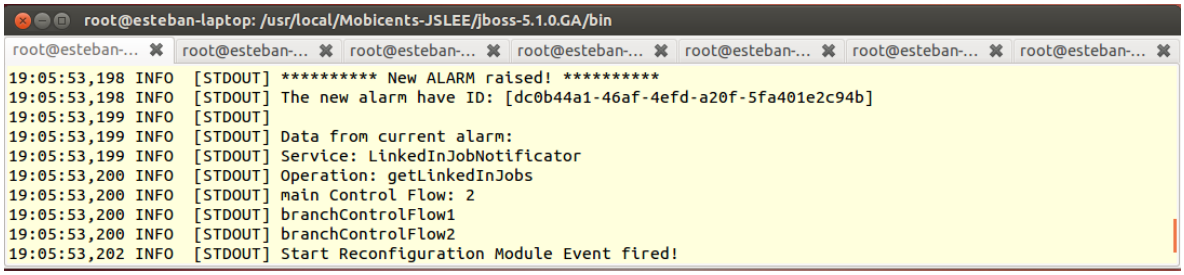

Figura 5-18. Envío de evento por parte del *Alarm Monitor*

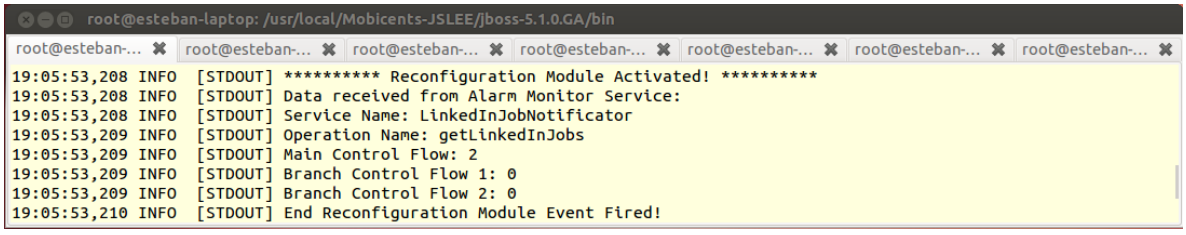

Figura 5-19. Recepción del evento por parte del Módulo de Reconfiguración

5.2.1.5 Módulo de Monitoring View

El desarrollo del módulo *Monitoring View*, fue realizado con fines de observación, de tal forma que sea posible por parte de un usuario administrador del sistema, poder conocer de manera gráfica el comportamiento en tiempo real de los valores de disponibilidad y tiempo de respuesta de los servicios atómicos.

Plan de Pruebas	
Grafica de valores de QoS	**Prueba Funcional 8:** permite comprobar que el sistema grafica en tiempo real los valores de disponibilidad y tiempo de respuesta de los servicios atómicos web y telco, que están siendo almacenados por parte del módulo *Service Monitor.*

Tabla 5-11. Pruebas módulo *Monitoring View*

Prueba Funcional 8: la ejecución de esta prueba permite comprobar la funcionalidad que grafica los valores de QoS para cada servicio atómico web o telco que compone un servicio convergente.

Para esta prueba se inicia el módulo *Service Monitor,* de tal forma que se estén almacenando valores de manera periódica en la base de datos. Luego, se procede a abrir el archivo de HTML5 que contiene la interfaz web asociada al módulo *Monitoring View,* donde se muestran las gráficas en tiempo real para los valores que están siendo almacenados por parte del *Service Monitor.* En este caso, se muestra la captura para los valores del servicio *LinkedIn-WS.* La figura 5-20 muestra la captura de la interfaz web en donde se muestra en tiempo real los valores asociados a la disponibilidad y tiempo de respuesta para este servicio atómico.

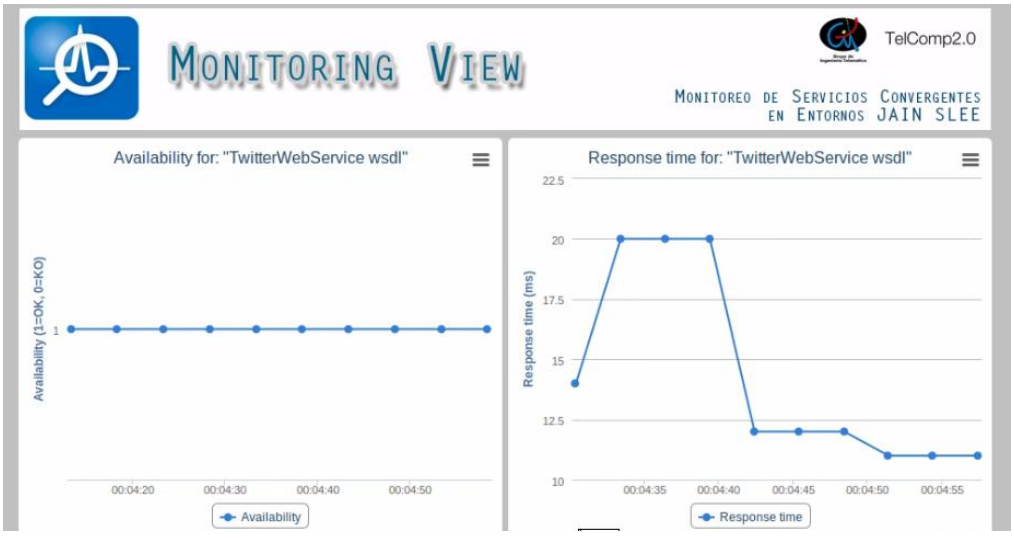

Figura 5-20. Gráfica de valores en tiempo real de disponibilidad y tiempo de respuesta

5.2.2 Evaluación de Rendimiento del Sistema

Para el análisis del rendimiento del sistema con el mecanismo de monitoreo ejecutado, se hace uso de una herramienta web llamada *Jolokia* [63], la cual permite el acceso de manera remota a los Java MBeans del servidor, definidos por la especificación JMX (*Java Management Extensions*), a través de un puente HTTP/JSON. De esta forma, es posible acceder a las interfaces de administración del servidor Mobicents JSLEE y recuperar información acerca del comportamiento del sistema. Específicamente, para este análisis, se ha accedido al MBean que provee información acerca del uso de memoria del servidor.

Los resultados de las pruebas de rendimiento, están condicionados por las prestaciones del equipo donde se ejecuta el software del prototipo. En este caso, el equipo empleado para desplegar el plan de pruebas descrito cuenta con las siguientes especificaciones técnicas:

Microprocesador	Intel Core i5 2450M @2.50 Ghz
Caché del Microprocesador	4 Mb de caché de nivel 2
Memoria RAM	6 Gb
Disco Duro	500 Gb (5400 RPM)
Sistema Operativo	Ubuntu 13.04
Red	Ethernet 1Gbps

Tabla 5-12. Especificaciones técnicas del Equipo empleado para las pruebas

5.2.2.1 Enfoque Reactivo

En primer lugar se mide el uso de memoria del servidor cuando los servicios atómicos básicos y el servicio *LinkedIn Job Notificator* se encuentran activos durante un periodo de tiempo en el que se toman muestras cada 10 segundos. Posteriormente se instalan los servicios que hacen parte del enfoque reactivo, es decir el servicio *Code Injector* y el servicio *Alarm Monitor* y se realiza nuevamente la medición del uso de memoria para el mismo intervalo de tiempo. La figura 5-21 muestra los resultados de las medidas realizadas.

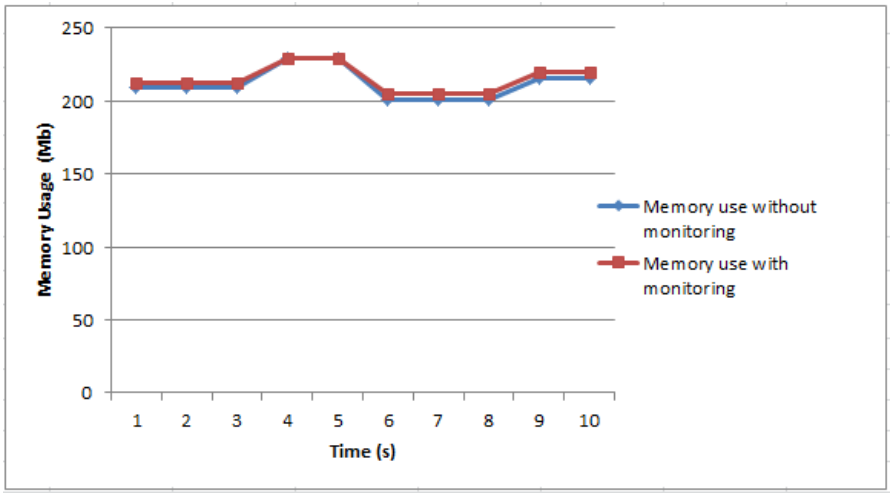

Figura 5-21. Uso de memoria con el enfoque reactivo del mecanismo

El incremento en el uso de memoria por parte del entorno de ejecución de Mobicents JSLEE cuando los servicios de monitoreo correspondientes al enfoque reactivo han sido instalados tiene un promedio de 2.6 Mb para el intervalo de tiempo medido.

Por lo anterior, es posible concluir que una vez los servicios de monitoreo se incluyen en el entorno de ejecución, el impacto correspondiente al consumo de memoria del sistema es mínimo y por lo tanto no afecta el normal funcionamiento del sistema.

5.2.2.2 Enfoque Proactivo

Al igual que las pruebas realizadas para el enfoque reactivo, en primer lugar se mide el uso de memoria cuando únicamente se encuentran activos los servicios atómicos básicos y el servicio *LinkedIn Job Notificator* durante un periodo de tiempo donde se toman muestras en intervalos de 10 segundos. Luego se instalan los servicios *Service Monitor*, *QoS Monitor* y *Alarm Monitor*, los cuales hacen parte del enfoque reactivo y se realizan nuevamente las mediciones del uso de memoria para el mismo intervalo de tiempo. La figura 5-22 muestra los resultados de las medidas realizadas.

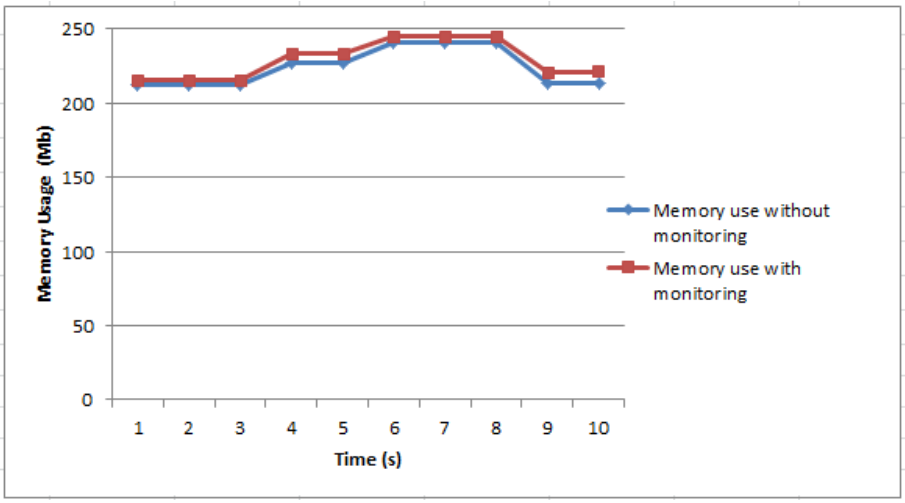

Figura 5-22. Uso de memoria con el enfoque proactivo del mecanismo

El incremento en el uso de memoria por parte del entorno de ejecución cuando se incluyen los servicios de monitoreo correspondientes al enfoque proactivo tiene un promedio de 5.2 Mb para el intervalo de tiempo medido.

De acuerdo a lo anterior, el incremento en el consumo de memoria una vez los servicios correspondientes al enfoque proactivo son incluidos, no genera un impacto de mayor consideración que pueda alterar el funcionamiento general del sistema.

Se puede observar que el consumo de memoria aumenta en comparación al consumo realizado cuando se implementan los servicios para el enfoque reactivo, esto se debe a que para el enfoque proactivo se instalan 3 servicios, los cuales consumen más recursos que los 2 servicios que se utilizan en el enfoque reactivo.

RESUMEN

En este capítulo se abordó una descripción detallada del proceso de desarrollo llevado a cabo, alrededor de la construcción del sistema software que implementa el mecanismo propuesto para el monitoreo de servicios convergentes en entornos JAIN SLEE. Se definieron además, los casos de uso del sistema, la arquitectura de referencia y el diagrama de despliegue que permitió soportar el proceso de desarrollo del mismo.

Por otra parte, se mostraron las pruebas desarrolladas para comprobar el correcto funcionamiento del mecanismo de monitoreo propuesto en el presente trabajo de grado. Para ello, se tuvieron en cuenta dos etapas, la primera de ellas asociada a la evaluación funcional de acuerdo a los dos enfoques del monitoreo: reactivo y proactivo; mientras que la segunda fue enfocada en determinar el impacto del mecanismo de monitoreo en el funcionamiento del entorno de ejecución donde fue desplegado.

De esta manera, se demostró la importancia de contar con este tipo de mecanismos que permitan llevar una adecuada gestión durante el tiempo de ejecución de los servicios convergentes.

Capítulo 6

6. Conclusiones, Contribuciones y Trabajo Futuro

En este capítulo son presentadas las conclusiones a las cuales se llegó con el desarrollo de este trabajo de grado, los resultados obtenidos y finalmente se proponen algunos trabajos futuros que podrían ser realizados tomando como base el presente trabajo.

6.1 Conclusiones

A continuación se describen las principales conclusiones que surgieron durante la ejecución del presente trabajo de grado:

- ✓ Se realizó un análisis de las investigaciones y trabajos relacionados con el monitoreo de servicios, en los cuales se encontró una carencia de mecanismos que tuvieran en cuenta no sólo el dominio web, sí no también el de las telecomunicaciones, de tal forma que se pudieran ejecutar en entornos convergentes como JAIN SLEE.

- ✓ En este trabajo de grado se propone un enfoque proactivo y no intrusivo para el monitoreo constante de los servicios atómicos que componen los diferentes servicios convergentes desplegados en el entorno de ejecución, permitiendo detectar violaciones en los valores de QoS para disponibilidad y tiempo de respuesta y generando notificaciones en caso de que estas se presenten. Este enfoque representa además, una herramienta muy útil para que los administradores del sistema, tengan una visión clara de los componentes que hacen parte de los servicios convergentes, ayudándolos a detectar comportamientos no deseados, de tal forma que puedan tomar las medidas necesarias para prevenir fallas mayores. Sin el enfoque propuesto, las violaciones en los valores de QoS no podrían ser detectadas hasta que suceda una falla grave en el servicio convergente, causando un impacto negativo para los proveedores de servicio, así como una mala experiencia para el usuario final.

- ✓ En este trabajo de grado se propone una nueva iniciativa para afrontar el problema del monitoreo de servicios convergentes, introduciendo el uso de técnicas automáticas de instrumentación de código en tiempo de ejecución que analizan la estructura de los servicios convergentes, de tal manera que fuese posible proveerles funcionalidades de monitoreo y detección de fallos sin modificar la lógica del negocio de dichos servicios. De esta forma, en caso de que se presente un fallo en la ejecución de un servicio convergente, se pueden tomar las medidas necesarias para su reconfiguración, de manera automática.

- ✓ Las técnicas de instrumentación de código en tiempo de ejecución, representan una importante herramienta en el área de los servicios convergentes, debido a que su aplicación no se encuentra limitada a las funciones de monitoreo, ya que estas se pueden extender a procesos de reconfiguración y adaptación, así como el de recolección de datos en tiempo de ejecución.

- ✓ El mecanismo de monitoreo de servicios convergentes desarrollado, fue validado mediante un servicio llamado *LinkedIn Job Notificator,* el cual ha sido desarrollado dentro del marco del entorno de creación de servicios de la plataforma TelComp2.0, lo cual permitió comprobar su aplicabilidad dentro del proyecto en el cual se encuentra enmarcado el presente trabajo de grado. De igual manera, la evaluación mostró resultados prometedores que permiten habilitar el monitoreo en tiempo real de los servicios convergentes, generando un impacto mínimo en el consumo de memoria del entorno de ejecución, de tal forma que no hay un efecto perceptible en el rendimiento de la ejecución del servicio convergente.

- ✓ Se utilizó la tecnología JAIN SLEE, la cual se basa en una especificación abierta y estándar para un servidor de aplicación de telecomunicaciones diseñado para altos niveles de desempeño en aplicaciones orientadas a eventos, y que permite el reúso de componentes. Adicionalmente, permite a través de su arquitectura de adaptadores de recursos, realizar una abstracción de nivel de control de un entorno Telco 2.0, logrando su aplicación dentro de un entorno convergente como el propuesto en el proyecto Telcomp2.0.

6.2 Contribuciones

Dentro de las principales contribuciones del presente proyecto de grado se destacan las siguientes:

- ✓ Se llevó a cabo un análisis y evaluación de las herramientas de carácter libre, vigentes en el área de monitoreo de servicios web, teniendo en cuenta una serie de parámetros no funcionales de QoS, que pueden ser aplicables a un entorno convergente, con el fin de definir un conjunto de métricas y reglas para el monitoreo de servicios convergentes.

- ✓ El desarrollo de un mecanismo para el monitoreo automático de servicios convergentes en entornos JAIN SLEE, el cual está basado en software de libre distribución y tiene en cuenta tanto el dominio web como el de las telecomunicaciones. Así mismo, este mecanismo cubre dos enfoques importantes en el área de monitoreo de sistemas de información, el reactivo y proactivo, los cuales funcionan de manera independiente de acuerdo a los requerimientos de monitoreo exigidos en diferentes escenarios.

- ✓ Se realizó un aporte a la comunidad de desarrolladores de Mobicents, debido a que se lograron integrar funcionalidades de instrumentación de código en tiempo de ejecución, dentro de un entorno como JAIN SLEE. Por otra parte, el presente trabajo es pionero en cuanto al uso de las Alarmas definidas en la especificación JAIN SLEE, ya que hasta el momento en la comunidad de desarrolladores no existe documentación o prototipos en donde se evidencie el uso de las mismas.

- ✓ Aporte al proyecto de Maestría en Ingeniería Telemática de la Universidad del Cauca, titulado *Reconfiguración Automática de Servicios Convergentes en Entornos JAIN SLEE,* desarrollado por el Mag. (c) Julián Andrés Rojas Meléndez.

- ✓ Se destaca la producción del artículo titulado: "*Automatic Code Instrumentation for Converged Service Monitoring and Fault Detection*" aceptado en la conferencia "*The 28th IEEE International Conference on Advanced Information Networking and Applications (AINA-2014)*", la cual se celebrará en Victoria, Canada, los días 13-16 de Mayo de 2014. Este artículo se presentará dentro del Workshop "*2nd International Workshop on Network Management and Monitoring (NetMM 2014*)". Los artículos aceptados en esta conferencia son indexados dentro de la base de datos de IEEE Computer Society, IEEE Xplore y DBLP.

- ✓ Se destaca la producción de artículo titulado: "*Automatic Monitoring of Converged Services on a Telco2.0 Environment based on QoS Constraints*" enviado a la revista Ingeniería y Universidad de la Pontifica Universidad Javeriana, indexada en Categoría A2 de Publindex - Colciencias, el cual se encuentra en este momento en periodo de revisión.

6.3 Trabajo Futuro

En el campo de investigación del presente proyecto de grado, se proponen los siguientes trabajos futuros:

- ✓ Definir un modelo formal que abarque todos los parámetros no funcionales de QoS que se deben tener en cuenta al momento de realizar el monitoreo de servicios convergentes.

- ✓ El mecanismo desarrollado en el presente trabajo, calcula los valores de los parámetros de QoS basados en lo propuesto inicialmente en [38], sería de gran interés desarrollar un mecanismo que permita medir los valores de QoS en servicios convergentes, teniendo en cuenta representaciones formales de los patrones de flujo de ejecución de los mismos.

- ✓ Extender las funcionalidades de monitoreo, teniendo en cuenta fallos asociados al comportamiento de la red en el entorno de ejecución de los servicios convergentes.

- ✓ Implementar un entorno gráfico en el cual se definan los valores y parámetros de QoS que se tendrán en cuenta para el monitoreo y posterior reconfiguración de los servicios convergentes. Así mismo, que permita la gestión de alarmas desde un entorno tanto web como móvil.

- ✓ Se propone realizar la experimentación del prototipo de monitoreo en un ambiente real con múltiples usuarios reales, para validar así, el desempeño del mecanismo propuesto en actividades de monitoreo con un proveedor de servicios real.

Referencias

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

[2] Ericsson, “Ericsson Converged Service Studio,” 2013. [Online]. Disponible en: http://www.ericsson.com/ourportfolio/telecom-operators/converged-service-studio. [Consultado: 12-Feb-2013].

[3] uReach Technologies, “Converged Services Framework.” [Online]. Disponible en: http://www.ureachtech.com/csf.html. [Consultado: 12-Feb-2013].

[4] JSRs: Java Specification Request, “JSR 22: JAIN SLEE API.” 2004.

[5] JSRs: Java Specification Request, “JSR 240: JAIN SLEE (JSLEE) v1.1.” 2008.

[6] J. O. Kephart and D. M. Chess, “The vision of autonomic computing,” *Computer*, vol. 36, no. 1. pp. 41–50, 2003.

[7] I. Braun, M. Wauer, S. Reichert, J. Spillner, A. Strunk, and A. Schill, “A Monitoring And Adaptation Architecture For The Future Internet Of Services,” 2009, pp. 67–71.

[8] J. Kosinski, P. Nawrocki, D. Radziszowski, K. Zielinski, S. Zielinski, G. Przybylski, and P. Wnek, “SLA Monitoring and Management Framework for Telecommunication Services,” in *Networking and Services, 2008. ICNS 2008. Fourth International Conference on*, 2008, pp. 170–175.

[9] P. G. Binod and R. Parthasarathy, “Converged application framework for creating web 2.0 - IMS mashups,” in *Internet Multimedia Services Architecture and Applications (IMSAA), 2009 IEEE International Conference on*, 2009, pp. 1–6.

[10] Mobicents, “Mobicents JAIN SLEE.” [Online]. Disponible en: http://www.mobicents.org/slee/intro.html. [Consultado: 10-Feb-2013].

[11] Open Cloud, “Rhino Telecom Application Server.” [Online]. Disponible en: http://www.opencloud.com/products/rhino-application-server/. [Consultado: 14-Feb-2013].

[12] Grupo de Ingeniería Telemática, “Telcomp2.0 Project Website.” [Online]. Disponible en: http://190.90.112.7:8080/TelComp-SCE/. [Consultado: 22-Oct-2013].

[13] COLCIENCIAS, “Departamento Administrativo de Ciencia, Tecnología e Innovación.” [Online]. Disponible en: www.colciencias.gov.co. [Consultado: 22-Oct-2013].

[14] OASIS, “Web Services Business Process Execution Language Version 2.0.” [Online]. Disponible en: http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.html. [Consultado: 26-Oct-2013]

[15] 3GPP, “IP-Multimedia Subsystem.” [Online]. Disponible en: http://www.3gpp.org/technologies/keywords-acronyms/109-ims. [Consultado: 16-Feb-2013].

[16] 3GPP, “The Evolved Packet Core.” [Online]. Disponible en: http://www.3gpp.org/technologies/keywords-acronyms/100-the-evolved-packet-core. [Consultado: 16-Feb-2013].

[17] ITU-T, “Y.2013: Converged services framework functional requirements and architecture,” in *Definitions vol. 2013*, 2006, p. 39.

[18] Open Cloud, “A Slee for all Seasons,” 2007. [Online]. Disponible en: http://www.opencloud.com/documents/Whitepaper A SLEE for all Seasons.pdf. [Consultado: 04-Feb-2013].

[19] Open Cloud, “An Introduction to JAIN SLEE,” 2006. [Online]. Disponible en: http://www.opencloud.com/documents/Profile JAIN SLEE.pdf. [Consultado: 05-Feb-2013].

[20] ITU-T, “E.800: Definition of Terms related to Quality of Service, Telecommunications Standardization Sector of ITU.” 2008.

[21] W3C Working Group, “QoS for Web Services: Requirements and Possible Approaches.” 2003.

[22] M. H. Hasan, J. Jaafar, and M. F. Hassan, “A review on monitoring vague quality of service (QoS) compliance for web services,” in *Computer Information Science (ICCIS), 2012 International Conference on*, 2012, vol. 1, pp. 517–521.

[23] F. Z. Safy, M. El-Ramly, and A. Salah, “Runtime Monitoring of SOA Applications: Importance, Implementations and Challenges,” *2013 IEEE Seventh Int. Symp. Serv. Syst. Eng.*, vol. 0, pp. 315–319, 2013.

[24] ITU-T, “M.3400: TMN management functions.” 2000.

[25] O. Cabrera and X. Franch, “A quality model for analysing web service monitoring tools,” in *Research Challenges in Information Science (RCIS), 2012 Sixth International Conference on*, 2012, pp. 1–12.

[26] ISO, “ISO/IEC 9126-1: Software engineering -- Product quality.” 2001.

[27] P. Falcarin and L. W. Goix, “An Aspect-Oriented Approach for Dynamic Monitoring of a Service Logic Execution Environment,” in *IEC Annual Review of Communications, vol. 59*, Chicago: International Engineering Consortium (IEC), Ed. 2006, pp. 237–242.

[28] Z. Haiteng, S. Zhiqing, and Z. Hong, “Runtime monitoring Web services implemented in BPEL,” in *Uncertainty Reasoning and Knowledge Engineering (URKE), 2011 International Conference on*, 2011, vol. 1, pp. 228–231.

[29] N. Goel and R. K. Shyamasundar, “Automatic Monitoring of SLAs of Web Services,” in *Services Computing Conference (APSCC), 2010 IEEE Asia-Pacific*, 2010, pp. 99–106.

[30] F. Li, B. Zhang, and L. Ge, “Composite Service Adaptation Supported Environmental Monitoring System,” in *Genetic and Evolutionary Computing (ICGEC), 2011 Fifth International Conference on*, 2011, pp. 315–318.

[31] O. Moser, F. Rosenberg, and S. Dustdar, “Non-intrusive Monitoring and Service Adaptation for WS-BPEL,” in *Proceedings of the 17th International Conference on World Wide Web*, 2008, pp. 815–824.

[32] M. Oriol Hilari, J. Marco Gómez, J. Franch Gutiérrez, and D. Ameller, “Monitoring Adaptable SOA Systems using SALMon,” Sep. 2008.

[33] M. Sun, B. Li, and P. Zhang, “Monitoring BPEL-Based Web Service Composition Using AOP,” in *Computer and Information Science, 2009. ICIS 2009. Eighth IEEE/ACIS International Conference on*, 2009, pp. 1172–1177.

[34] N. Artaiam and T. Senivongse, “Enhancing Service-Side QoS Monitoring for Web Services,” in *Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing, 2008. SNPD ’08. Ninth ACIS International Conference on*, 2008, pp. 765–770.

[35] F. Raimondi and J. S. W. Emmerich, “A Methodology for On-line Monitoring Non-Functional Specifications of Web-Services.” .

[36] R. Ben Halima, K. Guennoun, K. Drira, and M. Jmaiel, “Non-intrusive QoS monitoring and analysis for self-healing web services,” in *Applications of Digital Information and Web Technologies, 2008. ICADIWT 2008. First International Conference on the*, 2008, pp. 549–554.

[37] I. Grida Ben Yahia, E. Bertin, J.-P. Deschrevel, and N. Crespi, “Service Definition for Next Generation Networks,” in *Networking, International Conference on Systems and International Conference on Mobile Communications and Learning Technologies, 2006. ICN/ICONS/MCL 2006. International Conference on*, 2006, p. 22.

[38] T. Yu, Y. Zhang, and K.-J. Lin, “Efficient Algorithms for Web Services Selection with End-to-end QoS Constraints,” *ACM Trans. Web*, vol. 1, no. 1, May 2007.

[39] M. W. Dalia Khader, Julian Padget, *Grids and Service-Oriented Architectures for Service Level Agreements*. Springer US, 2010, pp. 13–22.

[40] A. Pakonen, T. Pirttioja, I. Seilonen, and T. Tommila, “Proactive Computing in Process Monitoring: Information Agents for Operator Support,” in *Emerging Technologies and Factory Automation, 2006. ETFA ’06. IEEE Conference on*, 2006, pp. 153–158.

[41] The Java Tutorials, “Lesson: Introducing MBeans,” 2013. [Online]. Disponible en: http://docs.oracle.com/javase/tutorial/jmx/mbeans/. [Consultado: 10-Sep-2013].

[42] Oracle, “Java Management Extensions (JMX) Technology,” 2013. [Online]. Disponible en: http://www.oracle.com/technetwork/java/javase/tech/javamanagement-140525.html. [Consultado: 28-Sep-2013].

[43] The Java Tutorials, “Trail: The Reflection API.” [Online]. Disponible en: http://docs.oracle.com/javase/tutorial/reflect/.

[44] S. Chiba, “Load-Time Structural Reflection in Java,” in *Proceedings of the 14th European Conference on Object-Oriented Programming*, 2000, pp. 313–336.

[45] Jb. Community, “Javassist Project Web Site.” [Online]. Disponible en: http://www.jboss.org/javassist/. [Consultado: 05-Oct-2013].

[46] W3C Working Group, “Status Code Definitions.” [Online]. Disponible en: http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html. [Consultado: 20-Sep-2013].

[47] MongoDB, “Introduction to MongoDB.” [Online]. Disponible en: http://www.mongodb.org/about/introduction/. [Consultado: 22-Oct-2013].

[48] Search SQL Server, “Relational Database.”

[49] R. Cattell, “Scalable SQL and NoSQL Data Stores,” *SIGMOD Rec.*, vol. 39, no. 4, pp. 12–27, May 2011.

[50] R. Burtica, E. M. Mocanu, M. I. Andreica, and N. Tapus, “Practical application and evaluation of no-SQL databases in Cloud Computing,” in *Systems Conference (SysCon), 2012 IEEE International*, 2012, pp. 1–6.

[51] JSON, “Introducing JSON.” [Online]. Disponible en: http://www.json.org/. [Consultado: 28-Oct-2013].

[52] MongoDB, “Top 5 Considerations When Evaluating NoSQL Databases.” 2013.

[53] A. Boicea, F. Radulescu, and L. I. Agapin, “MongoDB vs Oracle -- Database Comparison,” in *Emerging Intelligent Data and Web Technologies (EIDWT), 2012 Third International Conference on*, 2012, pp. 330–335.

[54] MongoDB, “Data Model Design,” 2013. [Online]. Disponible en: http://docs.mongodb.org/manual/core/data-model-design/. [Consultado: 22-Oct-2013].

[55] JQuery, “Browser Support.” [Online]. Disponible en: http://jquery.com/browser-support/. [Consultado: 23-Oct-2013].

[56] Highcharts JS, “Highcharts and Highstock documentation.” [Online]. Disponible en: http://www.highcharts.com/docs. [Consultado: 02-Nov-2013].

[57] “Node.js.” [Online]. Disponible en: http://nodejs.org/. [Consultado: 01-Nov-2013].

[58] Express.js, “Web application framework for Node.js.” [Online]. Disponible en: http://expressjs.com/. [Consultado: 02-Nov-2013].

[59] GitHub, “Mubsub for Node.js.” [Online]. Disponible en: https://github.com/scttnlsn/mubsub. [Consultado: 02-Nov-2013].

[60] C. E. Serrano, “Modelo para la Construcción de Soluciones,” in *Modelo Integral Para El Profesional En Ingeniería*, Popayán: Universidad del Cauca, 2005, pp. 75–94.

[61] Java Decompiler, “Java Decompiler Project.” [Online]. Disponible en: http://jd.benow.ca/. [Consultado: 05-Nov-2013].

[62] Robomongo, “Shell-centric cross-platform MongoDB management tool.” [Online]. Disponible en: http://robomongo.org/. [Consultado: 10-Nov-2013].

[63] Jolokia, “Jolokia, JXM on Capsaicin.” [Online]. Disponible en: http://www.jolokia.org/. [Consultado: 11-Nov-2013].