COMPOSICIÓN DINÁMICA DE SERVICIOS SIP PARA UN ENTORNO DE EJECUCIÓN DE LÓGICA DE SERVICIO, BASADA EN FSM

Andrés Fernando Muñoz Muñoz
Fabián Antonio Hoyos Pulido

UNIVERSIDAD DEL CAUCA
FACULTAD DE INGENIERÍA ELECTRÓNICA Y TELECOMUNICACIONES
DEPARTAMENTO DE TELEMÁTICA
POPAYÁN, SEPTIEMBRE DE 2011

COMPOSICIÓN DINÁMICA DE SERVICIOS SIP PARA UN ENTORNO DE EJECUCIÓN DE LÓGICA DE SERVICIO, BASADA EN FSM

Andrés Fernando Muñoz Muñoz
Fabián Antonio Hoyos Pulido

Trabajo de grado presentado como requisito para optar al título de
Ingeniero en Electrónica y Telecomunicaciones

Director
FULVIO YESID VIVAS CANTERO
Ingeniero en Electrónica y Telecomunicaciones

Asesor
OSCAR MAURICIO CAICEDO RENDON
Magister en Ingeniería Telemática

UNIVERSIDAD DEL CAUCA
FACULTAD DE INGENIERÍA ELECTRÓNICA Y TELECOMUNICACIONES
DEPARTAMENTO DE TELEMÁTICA
POPAYÁN, SEPTIEMBRE DE 2011

Tabla de contenido

INDICE DE FIGURAS

INDICE DE TABLAS

Capítulo 1
Introducción

1.1 Generalidades

Los operadores de telecomunicaciones migrarán o están migrando sus redes hacia las redes de nueva generación (NGN – Next Generation Network) buscando disminuir los costos de los procesos de desarrollo y despliegue de nuevos servicios de valor agregado [1]. Esta migración es indispensable ya que las exigencias de los usuarios que requieren nuevas funcionalidades, es mayor cada día, dada la comparación inevitable con la cantidad innumerable de los servicios disponibles en Internet. En este contexto, es de suma importancia que los sistemas de provisión de servicios tengan facilidades de composición que permitan responder de un modo adecuado a las necesidades mutables y crecientes de los clientes.

En el ambiente de las NGN, se han elegido las plataformas para despliegue de servicios (SDP – Service Delivery Platforms) para resolver el requerimiento anterior, gracias a sus capacidades para permitir el desarrollo rápido, la composición y ejecución de los servicios que cumplen con la alta disponibilidad exigida por el dominio de los telco [2]. Pero, JSLEE (JAIN SLEE - Java APIs for Integrated Networks – Service Logic Execution Environment), la tecnología más utilizada para implementar las SDP, solo ofrece facilidades para una composición estática de servicios [3].

La composición estática permite en tiempo de diseño formar nuevos servicios que utilizan funcionalidades existentes [4]. Pero, como los servicios componentes son preestablecidos, estos no pueden ser substituidos en tiempo de ejecución. Esto genera rigidez en el proceso de desarrollo de servicios y no es acorde con el cada vez más flexible y dinámico ambiente de los servicios de telecomunicaciones. Una situación particular que ilustra esta afirmación se presenta cuando un usuario de un servicio percibe una falla por un largo tiempo y un operador requiere, en el peor de los casos, la intervención de un desarrollador para substituir el componente afectado y empaquetar nuevamente el servicio compuesto para su puesta en funcionamiento.

Por todo lo anterior, es evidente la necesidad de complementar las funcionalidades de JSLEE con esquemas de composición con capacidades de reconfiguración dinámica, que permitan no solo combinar los servicios disponibles sino también tener en cuenta, la heterogeneidad y el dinamismo del contexto frente a fallas en tiempo de ejecución, por ejemplo [5].

En respuesta a esa necesidad se desarrolló el trabajo de grado presentado en esta monografía, la cual inicialmente en este capítulo presenta de manera general el contexto de este trabajo de grado, para ello se plantea un escenario de motivación, donde se ilustra el entorno en el que está enmarcado su desarrollo; y se presenta el planteamiento del problema, que describe el problema que se pretende resolver en este trabajo de grado, dando respuesta a la pregunta de investigación planteada por medio de los objetivos descritos en la sección 1.4.

1.2 Escenario de Motivación

En la actualidad los operadores de telecomunicaciones están enfrentando un proceso necesario de migración de las redes convencionales hacia los criterios que establecen las NGN en las que los mayores objetivos son la reducción de los costos en la creación y despliegue de servicios, facilitando la introducción rápida de nuevos servicios e incrementando las ganancias por el uso de los mismos [1]. Dicho proceso, se puede señalar como indispensable puesto que las exigencias de los usuarios cada vez son más grandes; una situación particular que ejemplifica esta afirmación es la insatisfacción que siente un usuario cuando su servicio presenta una falla en el tiempo de ejecución y no es posible arreglarlo de forma inmediata, generando la pérdida de clientes; en esos casos la forma actual de resolver la falla es estáticamente, donde

un desarrollador debe deshabilitar el servicio y realizar las modificaciones en forma manual, lo cual se ve reflejado en grandes pérdidas de dinero para el operador durante el tiempo que el servicio no está disponible.

Las NGN permiten integrar servicios con características tanto de los servicios tradicionales (telefonía) como de servicios que incluyan datos o video [1]. Por lo tanto se ve la gran necesidad de utilizar técnicas eficientes en los procesos de creación, despliegue y ejecución de los servicios. Un entorno que facilita dichas técnicas es JAIN SLEE puesto que provee secciones bien definidas que permiten realizar composición para lograr este objetivo. Además es un entorno que define un nuevo paradigma orientado a eventos que tolera comportamiento reactivo del dominio de las telecomunicaciones, logrando de esta manera, permitir la inclusión de nuevos procesos que ayuden a satisfacer las exigencias de los usuarios. Por lo tanto JAIN SLEE se presenta como una de las tecnologías más adecuadas para ser el ambiente de creación y ejecución de servicios en los Telcos que utilizan un entorno NGN.

Teniendo en cuenta lo mencionado, un problema que representa una gran dificultad para los operadores son las fallas en tiempo de ejecución de los servicios, lo cual es muy común en el entorno de telecomunicaciones al ser propenso a constantes cambios. Un concepto que contempla la reconfiguración de componentes en tiempo de ejecución es la composición dinámica y facilita el diseño de nuevos mecanismos para mitigar las fallas que se puedan presentar en la ejecución de un servicio. De esta manera, integrando el concepto de composición dinámica a las características provistas por JAIN SLEE, se potencia una solución que puede contrarrestar adecuadamente los problemas inesperados que puedan ocurrir en el tiempo de ejecución de los servicios de nueva generación, generando ventajas para los operadores, tales como: adaptar automáticamente componentes en tiempo de ejecución de los servicios de forma transparente a los usuarios, aumentar la robustez del ambiente de ejecución, aumentar la satisfacción del cliente e incrementar competitividad de los operadores de telecomunicaciones.

1.3 Planteamiento del problema

En la actualidad existe una amplia gama de servicios de telecomunicaciones y los requerimientos de los usuarios son cada vez más exigentes, debido a esto los operadores de telecomunicaciones deben mejorar su competitividad en los procesos de negocio para crecer y mantenerse en la industria. Así, los Telco han tenido que adoptar nuevas técnicas que disminuyan el tiempo y los costos para la creación de los servicios, una opción para dicho objetivo puede ser la colaboración entre servicios existentes para responder a necesidades complejas, un concepto que apunta a este tema es la composición de servicios. En este sentido, también la utilización del protocolo SIP en el desarrollo de servicios es de gran interés, ya que brinda las capacidades para la interacción entre el dominio de la web y el dominio de las telecomunicaciones, facilitando el desarrollo y despliegue de nuevas aplicaciones convergentes que integran voz, vídeo y datos; y para garantizar la alta confiabilidad en la prestación del servicio, las aplicaciones deben estar alojadas en un entorno de ejecución de lógica de servicio [6], [7], el cual permite establecer un ambiente con las condiciones óptimas para la construcción y despliegue de servicios de telecomunicaciones. Por lo tanto, en los Telco se ve la necesidad de aplicar la composición sobre los servicios que utilizan el Protocolo de Inicio de Sesión [8] en un SLEE, debido a que estos hacen parte de la construcción de servicios de nueva generación.

La composición de servicios SIP es de suma importancia, ya que ofrece una forma adecuada de utilizar los recursos para optimizar el desarrollo de nuevos servicios en un SLEE, logrando cumplir con los requisitos necesarios para competir con las nuevas demandas de negocio. Considerando que el comportamiento de los sistemas de telecomunicaciones está basado en mensajes asíncronos generados por sucesos inesperados en la red, una de las representaciones formales que mejor se adapta para trabajar en este ambiente es el modelo de las Máquinas de Estados Finitos [9], [10] debido a que está basado en cambios eventuales. Sin embargo, hasta el momento la composición de servicios SIP definida utilizando FSM, se realiza con una lógica de funcionamiento fija (composición estática); la cual se define como un proceso que se establece cuando la arquitectura y el diseño del software son planeados. Así, durante el desarrollo de los servicios, los componentes a ser usados son seleccionados, ligados y finalmente compilados y desplegados. Por lo cual,

este tipo de composición, puede ser restrictivo, especialmente cuando se requiere que los componentes se adapten automáticamente a cambios [11]. De igual manera, por defecto un SLEE construye servicios basados en la interacción de componentes, los cuales definen sus relaciones en la fase de desarrollo y son ejecutadas para realizar el flujo normal de los servicios, pero en tiempo real los servicios carecen de técnicas que puedan solucionar problemas inadvertidos que interrumpen el correcto funcionamiento de los mismos. Por tal motivo es evidente la ausencia de mecanismos que permitan cambios en las interacciones de los componentes de servicio en tiempo de ejecución (composición dinámica).

En el marco de la composición dinámica de servicios basada en FSM, se han definido múltiples algoritmos para diversos dominios de aplicación, especialmente para los Servicios Web como se presenta en [12], [13], [14]. Sin embargo, estos aún no han sido considerados para la composición dinámica de Servicios SIP dentro de un SLEE. La ausencia de dinamismo en la composición evita que al momento de crear servicios de telecomunicaciones (y entre ellos los Servicios SIP) sobre un SLEE se aprovechen ventajas como: reutilización de componentes en tiempo de ejecución, flexibilidad para adaptarse a nuevos cambios y la capacidad para el usuario de interactuar con aplicaciones de alta complejidad de forma transparente.

De acuerdo a lo expuesto hasta este punto, se plantea la siguiente pregunta de investigación:

¿Cómo realizar composición dinámica de servicios SIP en el contexto de un SLEE utilizando una representación formal basada en FSM?

1.4 Objetivos

1.4.1 Objetivo general

- Proporcionar mecanismos de composición dinámica de servicios SIP para un entorno de ejecución de lógica de servicio, utilizando FSM.

1.4.2 Objetivos específicos

- Caracterizar la composición dinámica de servicios SIP sobre un SLEE, utilizando FSM.
- Adaptar un algoritmo basado en FSM para realizar la composición dinámica de servicios SIP.
- Evaluar el algoritmo propuesto, a partir de un módulo de composición dinámica para un SLEE.

1.5 Estructura de la monografía

La presente monografía contiene los siguientes capítulos:

Capítulo 2: En este capítulo se describen los principales conceptos y se presentan los principales trabajos relacionados con la composición dinámica de servicios SIP en un SLEE basado en FSM.

Capítulo 3: En este capítulo se caracteriza la composición dinámica de servicios SIP en el SLEE, haciendo uso de FSM.

Capítulo 4: En este capítulo se detalla la arquitectura, el proceso de implementación y evaluación del algoritmo de composición dinámica de servicios SIP en el SLEE.

Capítulo 5: En este capítulo se presentan los aportes, conclusiones y trabajos futuros que finalizan el desarrollo de esta monografía.

Anexo A: En este anexo se encuentra la explicación de las herramientas empleadas para implementar el módulo de composición dinámica, las cuales son el servidor Mobicents JAIN SLEE 2.3.0.FINAL, el IDE Eclipse Helios v3.6.1 y el Plug-in EclipSLEE v1.

Anexo B: En este anexo se encuentra explicada la instalación del RA SIP 2.3.0

Anexo C: En este anexo se encuentran detalladas las características técnicas del entorno experimental utilizado para realizar las pruebas de desempeño sobre el módulo de composición y componente mediador para evaluar el rendimiento del algoritmo.

Anexo D: En este anexo se encuentra el artículo, en el cual se basó este trabajo de grado para adaptar el algoritmo de composición dinámica al entorno SLEE.

Anexo E: En este anexo se encuentra un artículo de divulgación que describe la arquitectura de referencia de este trabajo de grado.

Anexo F: En este anexo se encuentra un artículo de divulgación que describe el algoritmo de composición dinámica presentado en este trabajo de grado.

Anexo G: En este anexo digital se encuentran los Logs, generados a partir de las pruebas hechas para evaluar el algoritmo de composición dinámica planteado en este trabajo de grado.

Capítulo 2
Composición de Servicios en un Ambiente SIP

2.1 Conceptualización

2.1.1 Composición de servicios

La composición de servicios se refiere a establecer mecanismos que permitan a dos o más servicios cooperar entre sí para resolver requisitos que van más allá de sus capacidades individuales [15]. De esta manera, se puede construir servicios con características específicas e independientes unos de otros para que sean combinados y se obtenga el conjunto de funciones necesarias para conseguir un nuevo resultado.

A continuación, se describen los tipos de composición basados en algunas formas de interacción entre los servicios: composición estática, composición automática y composición dinámica, además se presentan los principales lenguajes asociados a ellas.

2.1.1.1 Composición estática

Se habla de formas estáticas de componer servicios, cuando la secuencia de la ejecución está determinada en algún archivo, utilizando mecanismos que son previamente conocidos y dependen de las condiciones determinadas durante el desarrollo del servicio (fase de diseño), por consiguiente siempre se genera el mismo resultado. La composición estática está enfocada en formar servicios bajo condiciones que no van a variar y su lógica de funcionamiento es predecible [11]. Sin embargo, hacia un enfoque práctico, los servicios de telecomunicaciones siempre están en ambientes expuestos a cambios, variaciones en los sistemas, introducción de nuevas tecnologías, cambios en los modelos de negocio, diferentes condiciones y requisitos de los usuarios. Por lo tanto, realizar procesos de composición estática es limitado para el desarrollo de nuevos servicios aptos para competir en la industrial actual, como son la capacidad de adaptarse a diferentes sucesos y la capacidad de reutilización para optimizar el uso de recursos [14].

2.1.1.2 Composición automática

Un concepto ampliamente utilizado por algunos autores es el de composición automática [12, 16, 17], definida como el proceso mediante el cual se determina el flujo de ejecución de un nuevo servicio a partir de otros servicios para que en un proceso posterior pueda ser ejecutado, es decir, la composición automática no se encarga de la ejecución de un servicio, sino de construir la síntesis de composición del servicio (flujo de ejecución de un servicio a partir de la interacción de sus componentes para realizar su lógica), este proceso está basado en los requerimientos solicitados y puede estar inmerso tanto en una composición estática como dinámica, ya que se encarga de crear el árbol de ejecución a partir de la coordinación de componentes del servicio, el cual es necesario para realizar todo mecanismo que permita la interacción de servicios con el fin de cumplir un objetivo común.

El concepto de composición automática ha sido ampliamente investigado e implementado en el dominio de los servicios Web, a partir de ahí se ha dado paso para la integración de este concepto al dominio de las telecomunicaciones tal como se hace en [18] donde se integra un motor de servicios basado en JAIN SLEE para orquestar entidades de servicio tanto del dominio de tecnologías de información como de telecomunicaciones haciendo uso de un motor BPEL, se plantea que integrando las características de BPEL con las de JAIN SLEE dentro de un contenedor que permita su interacción, es posible la creación automática del flujo de ejecución de un servicio de telecomunicaciones, sin embargo, en general el concepto de composición automática en el dominio de las telecomunicaciones tiene un escaso proceso de investigación hasta el momento.

2.1.1.3 Composición dinámica

A diario, en la Internet se encuentran disponibles nuevos servicios y el número de proveedores crece constantemente, debido a esto surgen nuevas necesidades para mantener y mejorar la tolerancia a fallos de los servicios en un entorno que posee condiciones altamente variables para el desarrollo y despliegue de los mismos. Este problema se puede contrarrestar con la composición dinámica que se caracteriza por la capacidad de adaptarse a cambios inesperados que afecten el flujo de ejecución normal de un servicio y generar las soluciones adecuadas para garantizar el funcionamiento correcto en tiempo de ejecución del mismo. Por lo tanto, se puede definir como un proceso de composición flexible que en tiempo de ejecución brinda diferentes soluciones para mantener servicios en funcionamiento, capaz de responder a fallos que no fueron tenidos en cuenta o son imprevisibles en la fase de diseño [11, 19, 20].

Una forma de realizar composición dinámica es por medio de un módulo inteligente de composición que se encargue de adaptar el componente candidato al flujo de ejecución del servicio y solucionar la falla de un componente original del servicio durante su ejecución de forma óptima y transparente a los usuarios, dicho módulo, creado por un desarrollador de aplicaciones, debe tener la capacidad de poder adaptarse a los diferentes eventos que se presenten y dar una respuesta efectiva. Con este nuevo enfoque, se habla de la creación de aplicaciones encargadas de modificar libremente o adaptar en tiempo de ejecución otras aplicaciones para ser reutilizadas en nuevos contextos con el fin de dar solución a problemas complejos [21].

La composición de servicios, a través de los años, ha sido ampliamente estudiada y numerosas técnicas y lenguajes han sido propuestos en búsqueda de una solución más adecuada, y aplicadas a diferentes dominios; especialmente a los WS, existen dos términos comúnmente usados en este contexto, orquestación y coreografía.

- **Orquestación**

La orquestación se encarga de coordinar como los servicios pueden interactuar a nivel de mensajes, lógica de negocio y el orden de ejecución. Este proceso se caracteriza por ser controlado por una única entidad, puede entenderse como privado y ejecutable, ya que sólo dicha entidad conoce el flujo de control e información del servicio involucrado y está en la capacidad de llevar a cabo las funciones determinadas [22].

A pesar de que el proceso de orquestación inicialmente fue concebido para trabajar en el dominio de los Servicios Web, también se han hecho investigaciones para integrar este concepto en el dominio de las telecomunicaciones, un ejemplo de ello se observa en [23] donde se presenta un análisis detallado de una arquitectura mejorada de JAIN SLEE en la cual se ha integrado un motor de flujo de trabajo siendo de esta manera capaz de orquestar entidades de servicio del dominio de las telecomunicaciones y tecnologías de información.

- **Coreografía**

La coreografía describe las interacciones observables entre los servicios, con el fin de lograr los objetivos propuestos; el término observable se refiere a aquellas interacciones que son catalogadas como pertinentes para alcanzar un objetivo durante la integración. La coreografía puede entenderse como un proceso público y no ejecutable; es público porque define el comportamiento común y visible entre los diferentes servicios implicados, y no ejecutable porque actúa como un protocolo de negocio que dicta las reglas de interacción que deben ser cumplidas por las entidades participantes [24]. La orquestación y la coreografía pueden ser realizadas de forma estática como también de forma dinámica, dependiendo de las condiciones de composición específicas establecidas por el desarrollador.

En [25] se proponen algunos requerimientos que deben cumplir los ambientes de creación de servicios, para ello se introduce el concepto de coreografía, con el fin de proveer servicios compuestos por un conjunto de componentes que integren características del dominio de las telecomunicaciones, es decir, el concepto de

coreografía puede ser integrado en el dominio de las telecomunicaciones con el fin de integrar el concepto de composición.

El concepto de composición dinámica que se trabaja en esta monografía se enfoca directamente sobre la orquestación de servicios, atacando exclusivamente la coordinación de servicios en tiempo real.

2.1.1.4 Lenguajes de composición

Para realizar la orquestación de servicios se utiliza principalmente el Lenguaje de Ejecución de Procesos de Negocios (BPEL-execution - Business Process Execution Language).

- **BPEL –execution**

Es un lenguaje de alto nivel que proporciona métodos de definición y soporte para flujos de trabajo y procesos de negocio, concebido entre otros por Oracle, Bea Systems, IBM, SAP y Microsoft, utiliza una sintaxis basada en XML para representar la lógica y los elementos asociados (Servicios Web), está diseñado para el control centralizado de la invocación de diferentes Servicios Web, con cierta lógica de negocio añadida que ayuda a la programación en gran escala. Este lenguaje provee un motor encargado de describir cambios de información interna o externamente, coordinar todas las actividades y compensar el proceso global cuando ocurre algún error con mensajes de inactividad o advertencia. BPEL-execution define una notación estándar para manejar de manera muy explícita los aspectos funcionales de los Procesos de Negocios: Control de flujo (bucle, paralela, etc), conversaciones o transacciones asíncronas o síncronas y soporta dos tipos de procesos de negocio: ejecutable y abstracto. El primero modela el comportamiento actual de un participante en una interacción de negocios y el segundo modela la descripción del intercambio de mensajes en la interacción de negocios sin revelar el comportamiento interno [11].

A pesar de que BPEL no está optimizado para ser usado en el dominio de las telecomunicaciones, se puede adaptar tal como se hace en [26], donde se propone un analizador (parser) para traducir la descripción del proceso de negocio contenida en documentos BPEL a un código java y soportar así el despliegue del servicio en un ambiente de ejecución de servicios basado en JAIN SLEE. Esto conduce a nuevas oportunidades para la creación rápida y eficiente de servicios usando un ambiente de creación de servicios con un alto nivel de abstracción y generación automatizada de servicios.

Los lenguajes utilizados para realizar coreografía de servicios son principalmente: Lenguaje de Descripción de Coreografía de Servicios Web (WS-CDL - Web Service Choreography Description Language) e Interfaz de Coreografía de Servicios Web (WSCI - Web Service Choreography Interface).

- **WS-CDL**

Este lenguaje permite describir las operaciones que ofrece un servicio, los mensajes de entrada y salida que soporta y los tipos de datos usados, los cuales son definidos en términos de esquemas XML, facilitando la colaboración entre pares (peer to peer) mediante la definición de los comportamientos comunes de cada participante de un proceso de negocio, pero no define lo que hace el servicio, ni cómo lo hace, por lo que no es posible expresar la semántica de los mensajes intercambiados ni el orden correcto de llamada [27].
WS-CDL es el lenguaje estándar con mayor aceptación para realizar composición de Servicios Web mediante el proceso de coreografía, no obstante, también puede ser integrado en el dominio de las telecomunicaciones tal como se hace en [25], donde se utiliza este lenguaje con el fin de introducir un nuevo sistema para integrar servicios con capacidades de las tecnologías de información y de los servicios de telecomunicaciones.

- **WSCI**

Es un lenguaje de descripción de interfaces basado en XML, describe el comportamiento observable de los Servicios Web expresado en términos de dependencias lógicas y temporales entre los mensajes

intercambiados, también realiza una descripción del orden de invocación y la capacidad de realizar operaciones en el contexto de intercambio de mensajes en el cual los WS participan. Trabaja en conjunto con el lenguaje de descripción de Servicios Web (WSDL - Web Service Description Language), así como con otros lenguajes para definición de servicios que presenten las mismas características que WSDL [28], [11].

2.1.2 Representación formal

Las representaciones formales son modelos científicos basados en algún lenguaje matemático para expresar relaciones, propiedades, parámetros, ecuaciones, etc. para estudiar el comportamiento de sistemas complejos difíciles de estudiar en un contexto real [29]. Para realizar composición de servicios se utilizan modelos de representación formal como grafos, redes de Petri, máquinas de estado finito entre otros.

2.1.2.1 Grafos

Los grafos son una estructura general de datos que sirve para representar objetos y conceptos. En una representación de grafos los nodos o vértices corresponden a los objetos o conceptos mientras los arcos o aristas definen las relaciones existentes entre los conceptos [20]. En general se pueden diferenciar dos tipos de grafo, grafo dirigido y grafo no dirigido; el no dirigido se caracteriza porque sus aristas no están orientadas hacia ninguno de sus dos extremos siendo así bidireccionales, mientras que en el dirigido sus aristas están orientadas hacia uno de sus dos extremos, indicando el sentido de la relación, consecuentemente, un grafo no dirigido puede ser representado mediante un grafo dirigido en el que cada par de nodos relacionados tiene dos aristas que están orientadas en sentidos diferentes, indicando que la relación es bidireccional [30]. Un grafo queda definido si se tienen dos conjuntos, un conjunto de aristas y otro de nodos, siendo el objetivo principal de un grafo identificar los vértices a los que están unidas cada una de sus aristas. Un grafo en su representación más básica es un modelo que no se podría adaptar a los sistemas de un contexto orientado a eventos, puesto que carecen de una función de transición que permita cambios eventuales tal y como se hace en las FSM, aunque es posible representar una FSM mediante un grafo enriquecido en el que exista un estado inicial único, donde los nodos representan los estados y los arcos las transiciones y las transiciones pueden ser etiquetadas para expresar el conjunto de mensajes como se afirma en [20].

Esta representación formal puede ser aplicada a diferentes contextos, en particular, en el dominio de las telecomunicaciones, es una representación muy útil ya que permite modelar situaciones específicas tal como se hace en [31] donde se presenta un sistema autónomo de aprovisionamiento de servicios para establecer calidad de servicio garantizada para comunicaciones extremo a extremo, donde se muestra a través de la representación de grafos que el problema de la adaptación y composición puede ser reducido al problema clásico de camino óptimo *k-multi-restricciones* (*k*-MCOP – *k*-multiconstrained optimal path).

2.1.2.2 Redes de Petri

Son un lenguaje formal y gráfico para modelar sistemas concurrentes, su principal característica es la forma natural como capturan los conceptos básicos de este tipo de sistemas [20]. Una red de Petri es un grafo dirigido y bipartito en el cual se pueden identificar dos tipos de nodos llamados lugares y transiciones, un lugar es representado por una circunferencia, puede contener marcas que identifican si una acción o salida está activa o no y además puede ser el destino u origen de varias transiciones, una transición es representada con una barra, puede ser el destino u origen de varios lugares. Los lugares y las transiciones son unidos por medio de flechas, las cuales representa la relación entre ellas, donde una flecha siempre une lugares con transiciones y nunca dos lugares o dos transiciones. Generalmente los lugares representan condiciones del sistema y las transiciones representan eventos sobre el sistema, cuando ocurre la activación de una transición en el sistema, el lugar anterior a la transición modela las precondiciones y el lugar después de la transición modela las post-condiciones [32].

2.1.2.3 FSM

El modelo de FSM permite diseñar algoritmos con características adecuadas para realizar composición dinámica, ya que la lógica llevada a cabo, depende de las diferentes entradas que dan lugar a transiciones o cambios para generar la mejor salida. Este modelo es muy utilizado en los WS y en otras áreas de servicio para el desarrollo de módulos inteligentes encargados de crear diversos mecanismos de colaboración de servicios.

Las FSM son un modelo formal de comportamiento compuesto por un número finito de estados y las transiciones entre estos. El comportamiento de un servicio se representa a través de un diagrama de estados en el que se examina la forma de ejecutar la lógica cuando se cumplen ciertas condiciones. Una transición es un cambio de estado provocado por un evento de entrada aplicado a un estado determinado.

Una máquina de estados finitos determinista de acuerdo a [9] es una tupla (Q, Σ, q_0, Δ, F), donde

- Q es un conjunto finito no vacío de estados
- Σ es el alfabeto de entrada (un conjunto finito no vacío de símbolos).
- q_0 es un estado inicial, un elemento de S
- Δ es la función de transición
- F es el conjunto de estados finales

A partir de esta definición, las FSM generan diagramas de transición de estados para describir modelos de comportamiento. Las FSM son ampliamente utilizadas para el diseño de hardware de los sistemas digitales, la simulación del comportamiento de aplicaciones software, la ingeniería de software, los compiladores, los protocolos de red entre otras aplicaciones.

En el área de la composición de servicios de telecomunicaciones, las FSM juegan un papel de gran importancia puesto que poseen sintaxis y semántica formales, y gracias al manejo de estados y transiciones eventuales sirven para representar aspectos dinámicos que son difíciles de expresar con otro tipo de diagrama.

2.1.2.4 Comparación de los modelos formales para la composición dinámica en un SLEE

En esta sección se presenta un balance de las representaciones que pueden ser utilizadas para formalizar la composición dinámica de servicios SIP en un SLEE, considerando las propiedades intrínsecas que establece la especificación JAIN SLEE 1.1 [33].

La arquitectura de JAIN SLEE está construida con base en el comportamiento reactivo (reacciona a posibles estímulos o eventos de la red) de los servicios de telecomunicaciones por esta razón un servidor de aplicaciones implementado con base en las características de un SLEE está diseñado y optimizado para soportar aplicaciones orientadas a eventos también conocidas como aplicaciones asíncronas; generalmente dichas aplicaciones no tienen activo un hilo de ejecución, típicamente definen métodos que son invocados cuando los eventos son entregados a la aplicación, estos métodos contienen códigos que reconocen el evento y realizan un procesamiento adicional para manipularlo [33].

La orientación a eventos de JAIN SLEE, le ha permitido adoptar las FSM como el modelo formal para desarrollar sus servicios, pero no define exactamente la forma de utilizar dicho modelo, consecuentemente, es deber del desarrollador escoger la mejor manera de implementar las aplicaciones considerando los preceptos de las FSM [33-35]. Esta afirmación deja entrever que la forma natural de trabajar con el entorno se presenta al considerar las FSM como el modelo que permite definir el comportamiento de los servicios, por esta razón, en este trabajo de grado se adoptan las FSM para facilitar el proceso de composición dinámica en el SLEE.

2.1.3 Servicios SIP

El protocolo SIP definido por el Grupo de Trabajo de Ingeniería de Internet (IETF - Internet Engineering Task Force) [8], es un protocolo de señalización basado en el modelo cliente servidor que hereda algunas características del protocolo de transferencia de hipertexto (HTTP - Hyper Text Transport Protocol), útiles para la navegación sobre la web, y además posee características para interactuar con sistemas de telecomunicaciones. Esto permite que SIP sea un protocolo que posee gran flexibilidad para incorporar nuevas funciones y reducir la complejidad en la implementación de servicios de telecomunicaciones de nueva generación [36].

SIP es un protocolo de la capa de aplicación, su objetivo es iniciar, establecer y finalizar la comunicación por medio de sesiones. SIP controla el direccionamiento de una llamada en Internet, sin embargo, para su establecimiento total, SIP trabaja en conjunto con otros protocolos que se encargan del flujo de información y de los datos propiamente dichos. Para esta actividad se utiliza el protocolo de descripción de sesión (SDP - Session Description Protocol) y el protocolo de transporte de tiempo real (RTP - Real-Time Transport Protocol). SDP se encarga de coordinar las características y protocolos de transmisión utilizados por los clientes. La tarea del RTP es, por otra parte, servir de mediador en el flujo de datos multimedia (audio, video, texto, etc.), es decir, codificar los datos, separarlos en paquetes y enviarlos [36], [37].

Los servicios SIP pueden ser descritos como aplicaciones o bloques funcionales, encargados de construir la señalización utilizando el protocolo SIP para establecer la comunicación entre usuarios finales. Se ha aplicado la composición sobre estos servicios para permitir la interacción de varios componentes de servicio, éstos pueden ser combinados según las necesidades de aplicación y por lo tanto desarrollados independientemente, proporcionando una característica de modularidad a este contexto y finalmente dando respuesta a una petición SIP. Eso permite por otra parte un mejor control de las interacciones de servicio. Por ejemplo, un mecanismo de composición de servicios SIP utilizado actualmente en los contenedores de aplicaciones SIP es el módulo del Enrutador de Aplicación (AR – Application Router). Encargado de invocar múltiples aplicaciones en un orden específico a partir de una petición SIP (el AR será descrito con más detalle en los trabajos relacionados, sección 2.2).

2.1.4 JAIN SLEE

2.1.4.1 SLEE

SLEE es un concepto maduro que define un entorno de ejecución de servicios con baja latencia, alto rendimiento y orientación a eventos de forma asíncrona; características básicas de los servicios de telecomunicaciones. Provee además una arquitectura para la construcción de contenedores que cumple con funciones como: desarrollo y despliegue de los servicios, manejo del ciclo de vida de la aplicación en tiempo de ejecución y una Interfaz de Programación de Aplicación (API - Application Programming Interface) dirigido a terceros con el fin de facilitar la comunicación con otros sistemas [34].

Actualmente, la arquitectura estándar más utilizada para desarrollar un SLEE es la especificación de JAIN SLEE [33] propuesta por SUN Microsystem. La cual, utiliza un modelo de componentes para construir aplicaciones orientadas a eventos y define la forma de interacción de estos con su respectivo contenedor en tiempo de ejecución. En JAIN SLEE los servicios son construidos fundamentalmente utilizando componentes llamados Bloques de Construcción de Servicio (SBB - Service Building Blocks), los cuales se comunican y realizan su lógica por medio del envío y recepción de eventos. Cada SBB tiene métodos de control de eventos que identifica el tipo de entrada y ejecuta el procesamiento correspondiente a la solicitud. El SLEE crea instancias de estos componentes para que ejecuten sus funciones y permite por medio de un adaptador de recursos la comunicación con entidades externas al entorno, como elementos de red. En este momento se pueden desarrollar y desplegar servicios que utilicen esta especificación con implementaciones como Rhino [38] y Mobicents [39].

A continuación se describen los principales atributos del SLEE teniendo en cuenta la especificación JAIN SLEE 1.1 [33].

2.1.4.2 Arquitectura

JAIN SLEE es un entorno de ejecución enfocado a aplicaciones orientadas a eventos, diseñado para soportar las características de los sistemas de telecomunicaciones. Su arquitectura permite la comunicación con sistemas externos mediante el uso de los adaptadores de recursos, cuyo comportamiento está determinado por las FSM, reafirmando así la decisión de utilizar dicho modelo en este trabajo de grado.

La especificación de JAIN SLEE ha sido definida para soportar servicios de telecomunicaciones, además proporciona un modelo de programación estándar que puede ser usado por la gran comunidad de desarrolladores Java. Este modelo de programación ha sido diseñado para simplificar el trabajo a los desarrolladores de servicios, eliminar los errores comunes de los programadores y asegurarse que los servicios sean robustos y puedan crearse rápidamente [36]. También brinda ventajas como portabilidad de servicios, modelo de eventos dinámico y flexible, independencia de cualquier protocolo de red y fácil integración con sistemas externos.

JAIN SLEE ha sido definida en una arquitectura en capas mostrada en la figura 1 [36]. A continuación se describe cada una de ellas:

- **Capa de gestión del SLEE:** especifica el mecanismo mediante el cual un administrador gestiona un SLEE (incluyendo la suscripción, servicio de instalación, etc.) y permite a los desarrolladores de servicios definir los datos necesarios para un servicio en particular**.**

- **Capa del sistema de componentes del SLEE:** define el enrutamiento de eventos tanto externos como internos y proporciona componentes comunes a todo el SLEE que pueden ser utilizados por los desarrolladores de servicios como herramientas de funcionamiento para las aplicaciones. Esta capa permite la comunicación entre los servicios del SLEE, estos llegan a los servicios según la prioridad definida, es decir, permite utilizar las características de los servicios de una forma ordenada generando así un proceso de composición controlado por el enrutador de eventos (ER – Event Router).

- **Capa del modelo de componentes**: específica cómo se crea la lógica de servicio, para posteriormente empaquetarla y que pueda recibir eventos externos que darán lugar a la ejecución de los servicios. Cada servicio está constituido por componentes que proveen una funcionalidad bien definida, de tal manera que puedan ser inter-relacionados con el fin de permitir la composición de servicios en el SLEE.

- **Capa de los adaptadores de recursos, API**: los adaptadores de recursos son responsables de la comunicación con un recurso en particular, generalmente externo al modelo de programación de JAIN SLEE, de la comunicación con la lógica de enrutamiento de eventos en la implementación del SLEE y además se provee un API al desarrollador para poder utilizar las premisas de un recurso externo y construir servicios que hagan uso de sus funcionalidades.

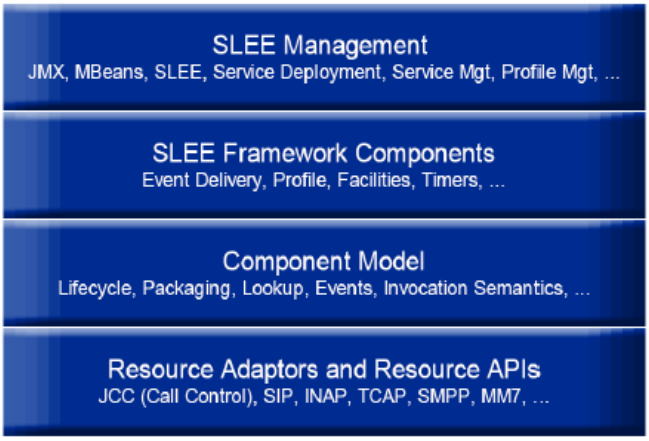

Figura 1. Arquitectura en capas de JAIN SLEE [36]

2.1.4.3 Elementos fundamentales

El SLEE está formado por un modelo de componentes que permite realizar la lógica de los servicios [36], [33] (ver figura 2), comunicarse entre ellos por medio de eventos y de conectarse con sistemas externos a él. A continuación se describe sus principales elementos.

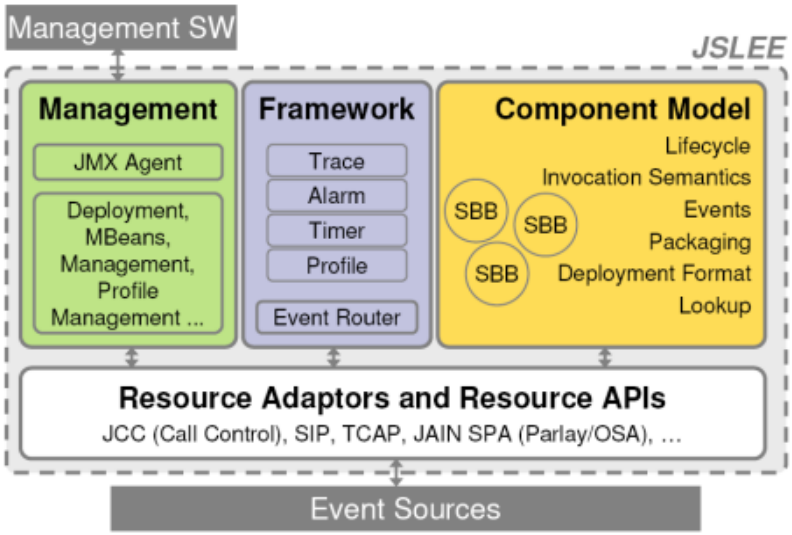

Figura 2. Modelo de componentes del SLEE [40]

5. Bloque de construcción de servicios (SBB – Service Building Block)

Son los componentes fundamentales de la arquitectura del SLEE. Un servicio contiene una o más instancias de diferentes SBB. Contienen la lógica del servicio, ya que tienen definidos los métodos controladores para los diferentes eventos que se reciben. Un SBB puede procesar solo un evento a la vez, pero varios SBB pueden ejecutarse concurrentemente. Los SBB pueden tener relaciones de herencia y varios SBB hijos.

Los SBB interpretan una lógica basada en la recepción de eventos, estos eventos se utilizan para representar hechos que pueden ocurrir en instantes de tiempo diferentes. Por ejemplo, en cualquier momento un sistema externo puede generar un evento dirigido hacia el SLEE y éste pasa a ser parte de su lógica de enrutamiento de eventos. Cada evento que llega al SLEE es procesado

por los componentes SBB que lo tengan definido en su archivo descriptor en donde se incluye información que describe al SBB, los eventos que recibe y los recursos que utiliza [36].

5. Relaciones Hijas

Los componentes SBB pueden tener cero o más SBB hijos. El componente SBB especifica las relaciones con sus hijos a través del elemento descriptor de despliegue y declara un método de acceso de la relación hija para hacer uso de ella en tiempo de ejecución.

6. Adaptadores de recursos (RA - Resource Adaptors)

Los recursos representan entidades externas que interactúan con otros sistemas fuera del SLEE, como son los elementos de red (HLR, MSC, etc.), las pilas de protocolos, directorios y bases de datos. Un adaptador de recursos adapta los requisitos y las interfaces particulares de un recurso a los requisitos e interfaces de JAIN SLEE [36]. Existe un tipo de RA para cada recurso, por ejemplo el protocolo SIP tiene su respectivo adaptador de recursos, el cual se encarga de adaptar el conjunto de peticiones y respuestas SIP a eventos que son utilizados dentro del SLEE para desarrollar la lógica de los servicios [36].

7. Eventos

Un evento representa una ocurrencia que requiere del procesamiento de una aplicación. El evento lleva información de descripción del origen y enrutamiento. Cada evento se ejecuta dentro del SLEE siendo reconocido según el tipo, por ejemplo, para el RA SIP, existen tipos de eventos Register, invite, Bye, etc. Cada evento es procesado en su método controlador de eventos dentro de cada SBB. Un evento puede tener su origen en diferentes fuentes, por ejemplo un recurso externo como puede ser la pila de un protocolo de comunicaciones, el mismo SLEE o componentes de aplicación dentro del mismo SLEE.

8. Contexto de actividad (AC - Activity Context)

El flujo de eventos producidos por una determinada entidad o recurso externo se define como actividad, y el contexto de actividad es la entidad lógica que representa y encapsula a la actividad en el SLEE, contiene atributos que permiten compartir información entre los SBB de un mismo componente y además sirve como un canal de eventos donde recibe todos los eventos dirigidos a este contexto y los entrega a los SBB correspondientes subscritos a él.

La importancia del contexto de actividad y el hecho de que una entidad SBB solamente puede recibir eventos disparados en contextos de actividad a los que está unido, radica en que si, por ejemplo, un SBB dispara un evento dirigido a otro SBB, lo hará a través de un contexto de actividad que estará unido al SBB destino y, por tanto, se evita que ese evento llegue a todos los SBB que no están implicados, pero que están a la “escucha” del mismo tipo de evento. Se tiene, de esta forma, un mayor control sobre los eventos.

9. Contenedor para la administración de datos persistentes (CMP - Container Managed Persistent)

Es un contenedor que se encarga de guardar el conjunto de atributos o campos que se definen en un SBB para que almacene y acceda a ciertos datos durante su ejecución. Estos atributos son accesibles desde los SBB a través de los métodos abstractos “getter” y “setter” que se deben definir para cada campo y que son implementados en tiempo de despliegue, es decir, no se encuentran implementados hasta que se realiza la primera instancia del SBB.

10. Servicio

Un servicio es un conjunto de SBB, donde se especifica por medio de archivos de despliegue todas las relaciones del SLEE con el SBB raíz (SBB padre del conjunto de SBB pertenecientes al servicio) y sus relaciones con los SBB hijos. En general este conjunto de SBB implementa una aplicación que ejecuta un servicio de telecomunicaciones. Estos servicios son encapsulados en lo que se denomina unidad de despliegue, que ya es el módulo final para ser ejecutado en un SLEE. Cada servicio tiene un SBB raíz, encargado de iniciar la lógica del servicio a partir de la recepción de un evento inicial y a su vez todos los SBB raíz tiene un padre lógico representado por el SLEE.

11. Perfiles

Los perfiles son un mecanismo muy eficiente para el almacenamiento y acceso de datos. Estos datos tienen un esquema que es un conjunto de propiedades y atributos. Los perfiles contienen información del servicio y del subscritor, y los SBB que se ejecutan en el SLEE pueden acceder a ella como parte de su lógica de aplicación.

- **Facilidades**

La especificación JAIN SLEE 1.1 define una serie de funcionalidades para gestionar de forma eficiente el uso de una aplicación, sus servicios, recursos y flujo de eventos. Además permite medir el rendimiento de la plataforma y su estado por medio de: estadísticas, logs, etc. Estas facilidades mejoran el desarrollo y administración de servicios al brindar una serie de funcionalidades genéricas que son comúnmente usadas por los servicios, haciendo más fácil y eficiente el trabajo de los desarrolladores. Consecuentemente, esta plataforma ofrece un conjunto estandarizado de módulos funcionales que son reutilizados por sus componentes para optimizar la implementación y administración de nuevos servicios permitiendo un uso de funcionalidades genéricas para realizar la comunicación y gestión de los módulos que componen un servicio. A continuación se describe las facilidades incorporadas en la especificación [33], [41]

- ✓ **Facilidad de Temporización (Timer Facility):** Ofrece temporizadores que pueden ser usados por los SBB para realizar tareas periódicas o que necesitan dar conteos específicos de tiempo para el inicio de una acción determinada. La facilidad de temporización controla un número determinado de temporizadores totalmente independientes entre sí, cada temporizador puede lanzar 0 ó n eventos tras su vencimiento.

- ✓ **Facilidad de Alarmas (Alarm Facility):** Provee la funcionalidad de registrar alarmas dentro del SLEE con el objetivo de que sean monitoreadas por herramientas de gestión. Los RA, SBB y los Perfiles pueden registrar y quitar alarmas.

- ✓ **Facilidad de Trazado (Trace Facility):** Brinda la funcionalidad de registrar trazas de información que puedan ser monitoreadas desde herramientas de gestión. Los RA, SBB, Perfiles y componentes internos del SLEE pueden hacer uso de esta facilidad.

- ✓ **Facilidad de Nombramiento de AC (Activity Context Naming Facility):** Permite a los SBB dar nombres o alias a contextos de actividad de modo que puedan ser usados desde otros SBB.

- ✓ **Facilidad de Perfiles (Profile Facility):** Esta facilidad le permite a los SBB consultar y modificar datos almacenados en tablas de perfiles.

- ✓ **Facilidad de Búsqueda de Servicios (Service Lookup Facility):** Es usada solo por los RA con el propósito de consultar información sobre los tipos de eventos que un servicio instalado en el SLEE puede recibir.

- ✓ **Facilidad de Búsqueda de Eventos (Event Lookup Facility):** Esta facilidad solo puede ser usada por los RA instalados en el SLEE y les permite consultar información sobre los tipos de eventos que están instalados.

2.1.4.4 Mecanismos de comunicación en el SLEE

JAIN SLEE es un entorno totalmente orientado a eventos, la lógica que maneja está basada en componentes de servicio y mensajes (eventos) modelados con FSM, de tal manera que facilita el proceso de composición estática de servicios al poder establecer en la etapa de desarrollo las relaciones entre los componentes (SBB) que ejecutan la lógica de un servicio, o también establecer las relaciones requeridas entre diferentes servicios. Es decir, el entorno brinda las condiciones necesarias para lograr un proceso de composición estática, pero carece de procesos que permitan reconfigurar el flujo de ejecución de un servicio en tiempo real.

JAIN SLEE define diferentes elementos (sección 2.1.4.3) para conformar toda la lógica orientada a eventos y con esta surgen dos tipos de comunicación entre los SBB: síncrona en la que los SBB se comunican directamente y asíncrona en la que la comunicación se realiza a través del SLEE [33].

2.1.5 Comunicación síncrona

Este tipo de comunicación se caracteriza porque los mensajes entre los SBB se transmiten directamente, haciendo uso de las relaciones hijas (Child Relation) que comunican un SBB padre (Parent SBB) con un SBB hijo (Child SBB). El SBB padre define la relación hija mediante su archivo XML descriptor, y ésta a su vez define el SBB hijo y la prioridad por defecto del evento entregado. En la figura 3 se observa la composición de SBB de forma síncrona donde el SLEE hace una instancia de los SBB raíz (Root SBB) X e Y para poder entregarles los eventos iniciales. Los SBB raíz de forma síncrona y paso a paso transfieren los eventos recibidos del SLEE o invocan funciones a los SBB con los cuales definieron relaciones hijas, representadas en la figura 3 por una arista dirigida y acompañadas por una etiqueta cuyo número indica la prioridad de entrega del evento al SBB hijo, definida en la relación hija correspondiente. Se puede notar también que un componente SBB puede ser padre de cero o más SBB incluyéndose a sí mismo como ejemplifica el SBB Z, y al mismo tiempo ser hijo de cero o más SBB.

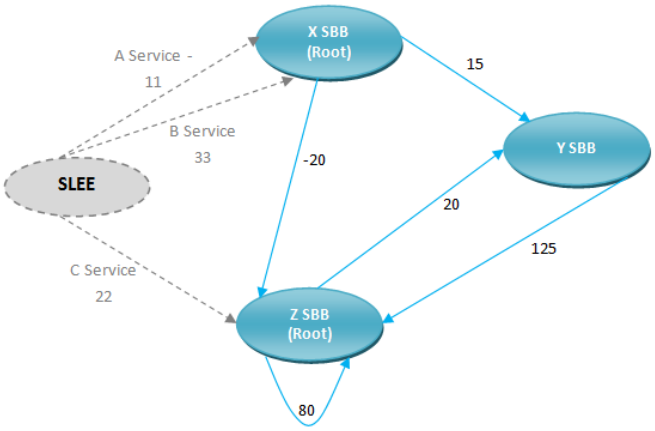

Figura 3. Composición síncrona de SBB (adaptación de [33])

La anterior gráfica indica como los SBB interactúan entre sí por medio de las relaciones hijas, dichas relaciones quedan definidas en la fase de desarrollo de un nuevo componente, de esta manera, el SBB padre

puede interactuar con el hijo de dos formas, la primera, haciendo uso de la interfaz local denominada *SbbLocalObject* para utilizar las capacidades y funcionalidades del SBB hijo y la segunda, transfiriendo un evento para que este sea procesado en el SBB hijo.

2.1.6 Comunicación asíncrona

Este tipo de comunicación se caracteriza porque los eventos son gestionados por el enrutador de eventos de JAIN SLEE [33]y es utilizada para transferir eventos entre servicios y no entre SBB como se realiza en la comunicación síncrona. El servicio puede disparar eventos hacia el SLEE (representado por el enrutador de eventos) desde cualquiera de sus componentes SBB, por el contrario, cuando recibe eventos desde el SLEE únicamente lo puede hacer en su SBB raíz y a partir de éste, realiza toda la lógica de ejecución que se le haya definido en la etapa de desarrollo.

En la figura 4 se muestra una representación de comunicación asíncrona, donde el servicio X, lanza un evento, a través del SLEE, que va dirigido hacia el servicio Z, éste al realizar su lógica, vuelve a lanzar un evento que va dirigido al servicio A.

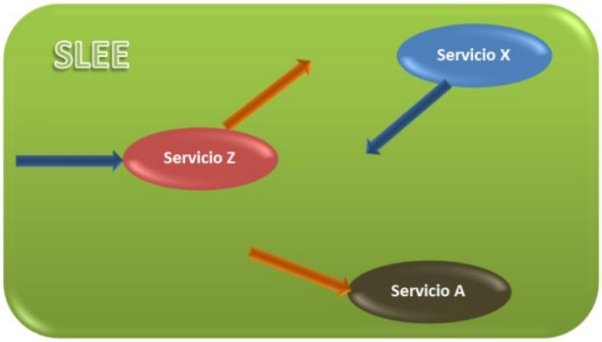

Figura 4. Composición asíncrona de servicios

Como se puede observar, el SLEE permite la interacción de componentes de servicio (SBB), o interacción entre servicios, esta característica, facilita el proceso de composición estática en el SLEE, pero como ya se mencionó, el SLEE no posee mecanismos que permitan realizar composición dinámica de servicios como se plantea en este trabajo de grado. En este sentido, se plantea realizar un módulo que mediante comunicación asíncrona permita reconfigurar un servicio en tiempo real, cuando haya ocurrido alguna falla.

Cabe mencionar que en este trabajo de grado se considera composición dinámica de componentes SBB que se comuniquen por medio de eventos, descartándose la interacción que se puede realizar mediante la interfaz local *SbbLocalObject* (en la que un SBB padre puede utilizar las funcionalidades de sus SBB hijos) puesto que, además de que este proceso está muy ligado al código del servicio, la idea de este trabajo es resolver fallos que se presenten en un ambiente asíncrono, en el que la comunicación este regida por cambios eventuales.

2.1.7 Enrutador de eventos

JAIN SLEE define un enrutador lógico de eventos encargado de direccionar los eventos hacia los servicios y manejar el orden en que estos pueden ejecutarse según sus prioridades de ejecución. Mediante este componente, el SLEE permite realizar un mecanismo de composición, en el que los servicios pueden interactuar entre ellos haciendo uso de la comunicación asíncrona (ver sección 2.1.6), para ello, en la etapa de desarrollo se debe especificar en el archivo descriptor de los SBB correspondientes de cada servicio el evento que será lanzado hacia el enrutador o que será recibido por el SBB raíz de un servicio. Este mecanismo *se clasifica dentro de la composición estática, ya que el algoritmo de enrutamiento de este módulo no prevé cambios en tiempo real para reorganizar el orden de ejecución de los SBB en caso de situaciones que no se hayan tenido en cuenta durante la etapa de diseño.* De esta manera, el módulo

planteado en este trabajo de grado, en colaboración con el proceso de composición realizado por el enrutador de eventos ayuda a corregir este problema.

El algoritmo de enrutamiento del ER en general permite controlar el flujo de eventos externos e internos del SLEE. Cuando un evento es disparado desde los RA (evento externo) o a partir de algún SBB de un servicio (comunicación asíncrona) es entregado al ER y éste se encarga de reconocer el servicio al cual está dirigido, en particular selecciona el SBB raíz subscrito al contexto de actividad involucrado, encargado de recibir este tipo de evento, si existe más de un servicio cuyo SBB raíz sea capaz de soportar dicho evento, el ER define la prioridad de entrega y ejecuta la acción disparo de dicho evento. La lógica del ER también se encarga del ciclo de vida de los SBB y demás componentes para optimizar la ejecución de los servicios y evitar problemas de rendimiento en la plataforma por causa del tráfico de eventos.

El enrutador de eventos es un componente fundamental para lograr el mayor rendimiento del SLEE, por tal motivo, su diseño e implementación están basados en un modelo formal para describir el comportamiento de los componentes del entorno de ejecución de lógica de servicio y conseguir una forma eficiente de construir su lógica de funcionamiento. En la siguiente sección se presenta el modelo formal del ER

2.2 Trabajos relacionados

En esta sección se presentan los trabajos relacionados con composición y demás conceptos que hacen parte de esta investigación, se realiza una descripción general del tema relacionado con cada proyecto, también se presentan las respectivas diferencias que existen en cada proyecto con el presente trabajo de grado. Cada trabajo se clasifica en composición estática o dinámica, y en telecomunicaciones o en el dominio de la Web. Se aclara que, si los trabajos no presentan alguna forma de composición dinámica, se clasifican en composición estática, así contemplen un proceso de composición automática, asimismo, si los trabajos además de contemplar el dominio de la Web, también contemplan el de telecomunicaciones, el trabajo se clasifica en el dominio de las telecomunicaciones.

A través de la investigación preliminar realizada durante la realización del anteproyecto, se encontró que el concepto de composición fue concebido inicialmente para el dominio de los WS, de esta manera, se optó por investigar la forma de realizar composición en el dominio de la Web (además de los trabajos de composición en telecomunicaciones), para adquirir el conocimiento necesario y así adaptar los conceptos respectivos al dominio de telecomunicaciones, en particular, la composición dinámica de servicios SIP en un SLEE. También es notable que en cuanto al concepto de composición dinámica existen pocos trabajos de investigación en comparación con el proceso de composición estática y más aún en el dominio de las telecomunicaciones.

2.2.1 Trabajos relacionados con composición estática en el dominio de telecomunicaciones

2.2.1.1 Composición de servicios basada en una capa de arquitectura de redes sobrepuesta para servicios de próxima generación

El paradigma de los Servicios Web ha sido ampliamente usado para la composición de servicios en internet, por otro lado, se han puesto en marcha varios proyectos para integrar la arquitectura orientada a servicios (SOA – Service Oriented Architecture) en el dominio de las telecomunicaciones con el fin de facilitar la inserción rápida de nuevos servicios, como es el caso del grupo de trabajo de composición de servicios basada en una capa de arquitectura de redes sobrepuesta para servicios de próxima generación (NGSON – Next-Generation Service Overlay Networks) de la IEEE. De esta manera, el proyecto presentado en [1] proporciona una arquitectura mejorada de NGSON (E-NGSON – “Enhanced” NGSON) añadiendo características de composición de servicios.

El objetivo de E-NGSON es permitir a los operadores de red y proveedores de servicio tener una solución efectiva para la gestión, control, creación, composición, despliegue y ejecución de servicios. Con el fin de

proveer los conceptos de contenidos generados por el usuario (UGC – User-Generated Contents) y servicios generados por los usuarios (UGS – User-Generated Services) E-NGSON provee herramientas para crear nuevos servicios Web o de telecomunicaciones, permitiendo a los usuarios componer servicios uniendo características de estos dos dominios.

En este proyecto se modifica la arquitectura de NGSON para adicionarle características que permitan adoptar el desarrollo rápido de servicios, así como también el despliegue de los mismos, considerando la reutilización de componentes tanto del dominio de las telecomunicaciones como de la Internet, clasificándose así dentro de la composición estática, de esta manera la principal diferencia de este proyecto con el presente trabajo de grado es que: realiza composición estática con la arquitectura NGSON, sin tener en cuenta procesos que permitan realizar composición dinámica, además no se tiene en cuenta el SLEE.

2.2.1.2 Lineamientos para composición de servicios de telecomunicaciones en un entorno JAIN SLEE basado en software de libre distribución.

En el trabajo [42] se hace un planteamiento sobre la dificultad que presentan las empresas de telecomunicaciones en la actualidad para resolver la demanda de servicios que el modelo Telco 2.0 solicita. Por tanto, se necesita incorporar nuevas tecnologías que permitan mejorar la interoperabilidad y dotar a los sistemas de un mayor grado de agilidad y flexibilidad en el desarrollo de servicios. Dentro de estas tecnologías se encuentra la especificación JAIN SLEE, la cual se presenta como una plataforma robusta para el rápido desarrollo y despliegue de servicios de telecomunicaciones de nueva generación. Debido a que JAIN SLEE es orientado a componentes, facilita la composición de servicios previamente desarrollados, con lo cual reduce el tiempo de lanzamiento al mercado de los nuevos servicios, generando grandes ventajas para los telco. Actualmente, estos procesos son orientados a un comportamiento estático, donde las funciones realizadas utilizan lenguajes basados en el manejo de archivos ejecutables. Teniendo en cuenta este enfoque, se plantean lineamientos para la composición estática de servicios de telecomunicaciones sobre JAIN SLEE utilizando herramientas de libre distribución, con el fin de seleccionar la mejor forma de realizar este proceso. Donde se realiza una identificación, descripción y evaluación de las diferentes herramientas de libre distribución disponibles actualmente, y presenta una base conceptual relacionada con el uso de la especificación JAIN SLEE y sus diferentes capacidades para el desarrollo de servicios de telecomunicaciones.

El principal objetivo de este trabajo es presentar los lineamientos que permitan realizar composición estática de servicios sobre JAIN SLEE, sin embargo no se especifica algún algoritmo que permita realizar dicho proceso, tampoco es considerado el modelo formal de FSM, siendo esta y el hecho de que la composición considerada es estática las principales diferencias de esta investigación y el presente trabajo de grado.

2.2.1.3 Composición de servicios de múltiples tecnologías para el dominio de las telecomunicaciones – Conceptos y experiencias

En telecomunicaciones, los servicios están sujetos a requerimientos específicos tales como soporte para sesiones de comunicaciones end-to-end y mecanismos para la interacción de servicios de diferentes dominios, tales como, telecomunicaciones, empresariales e internet. El objetivo del trabajo planteado en [43] es realizar composición de servicios convergentes, en el sentido de que pueden estar constituidos por componentes de múltiples dominios de servicio, además, estas aplicaciones pueden ser alcanzadas desde múltiples redes usando diferentes tecnologías de acceso. En consecuencia, sería posible, proveer a los usuarios servicios que contengan todos los componentes con los que ellos están familiarizados sin tener en cuenta su dominio de red.

Para realizar composición de servicios convergentes, en [43] se realiza una comparación entre los servicios del dominio de las telecomunicaciones y el dominio de internet donde se encuentra que el enrutamiento de aplicaciones realizado en telecomunicaciones es fundamentalmente diferente de la composición realizada

en los Servicios Web, es decir, en el dominio de los Servicios Web, los servicios son invocados siguiendo un esquema de petición-respuesta (request-response), de esta manera, se pueden invocar los servicios necesarios para dar respuesta a una petición; mientras que en el dominio de las telecomunicaciones se maneja una conexión end-to-end entre los participantes de una sesión, así, cuando se establece una sesión entre múltiples usuarios, cada participante tiene servicios definidos de acuerdo a sus necesidades particulares. En la web, los servicios pueden ser invocados síncrona o asíncronamente, en el dominio de las telecomunicaciones los servicios son invocados secuencialmente y operan asíncronamente. Consecuentemente, se pueden identificar algunos requerimientos para realizar una composición de servicios convergentes:

1. Un mecanismo que permita la interacción con un servicio de telecomunicaciones persistente después de tener respuesta a su invocación. Lo que permitirá que el servicio encargado de realizar la composición esté informado de las actualizaciones que puedan ocurrir en el dominio de las telecomunicaciones, por ejemplo al recibir mensajes SIP.
2. Un mecanismo de invocación que soporte requerimientos en tiempo real.
3. Soporte para gestionar la interacción de servicios que resulten naturalmente entre los servicios compuestos.

Basado en los requerimientos identificados, se propone un sistema para la composición de aplicaciones convergentes usando servicios de la Web, empresariales y telecomunicaciones, basados en los principios de SOA, por lo tanto, todos los servicios son considerados como autónomos y con unidades fácilmente adaptables.

El modelo de composición planteado en [43], tiene como objetivo principal combinar componentes de diferentes dominios, tales como, el dominio de la Web, empresarial y de telecomunicaciones. Las diferencias más relevantes del sistema propuesto con el presente trabajo de grado son: no tiene en cuenta la lógica interna de un SLEE, tampoco está definido bajo la definición formal de FSM y la composición que se maneja no considera coordinación de servicios y fallos en tiempo de ejecución.

2.2.1.4 Diseño de un ambiente de ejecución orquestado basado en JBI

En [18] se presenta un análisis detallado de una arquitectura mejorada de la especificación para integración fundamentada en java (JBI – Java Business Integration) en la cual se integra un motor de servicios basado en JAIN SLEE (debido a su alto rendimiento, soporte de interacciones asíncronas y aprovechamiento de las capacidades de diversos protocolos) para orquestar entidades de servicio tanto del dominio de tecnologías de información como de telecomunicaciones. Para orquestar los servicios atómicos de negocio contenidos en JAIN SLEE, se despliegan procesos en un motor de ejecución BPEL previamente diseñados, de esta manera el ambiente de ejecución orquestado es capaz de integrar el negocio tanto en el dominio de internet como el dominio de las telecomunicaciones.

El contenedor JBI utilizado en [18] es Servicemix (desarrollado por Apache), el cual define una arquitectura basada en los conceptos de orientación a servicios para integrar sistemas a través de componentes JBI que inter-operan intercambiando mensajes por medio de un enrutador. Teniendo integrado el contenedor JBI, el motor BPEL y JAIN SLEE, el proyecto se enfoca en el diseño e implementación de la lógica de un servicio de conferencia multimedia incluyendo los componentes atómicos y organización del flujo de ejecución del servicio para probar el funcionamiento correcto.

El proyecto descrito en [18] ilustra de forma general, las funciones fundamentales del entorno de ejecución JAIN SLEE y el motor de ejecución BPEL, pero no considera la forma de crear y combinar eficientemente servicios atómicos, de esta manera, se considera como la mayor diferencia, con el presente trabajo de grado, el hecho de que en [18] se presenta la descripción de un ambiente orquestado, el cual, mediante el uso de BPEL, provee condiciones muy adecuadas para poder trabajar el concepto de composición y se plantea el flujo de ejecución de un servicio multimedia, sin embargo, no se considera la composición

dinámica de SBB en tiempo de ejecución, haciendo uso de BPEL en el contenedor JBI, por esta razón, dicho proyecto se clasifica dentro de la composición estática.

2.2.1.5 TeamCom: Una plataforma de creación de servicios para redes de próxima generación

En el proyecto descrito en [26] se presenta un ambiente para la creación de servicios sobre la capa de control de las redes de próxima generación, es decir, sobre IMS, que da soporte al desarrollador de aplicaciones para componer servicios basados en componentes reutilizables. Con el fin de permitirle al desarrollador, enfocarse en la lógica de las aplicaciones, en [26] se propone un analizador (parser) para traducir la descripción del proceso de negocio contenida en documentos BPEL a un código java y soportar así el despliegue del servicio en un ambiente de ejecución de servicios basado en JAIN SLEE. Esto conduce a nuevas oportunidades para la creación rápida y eficiente de servicios usando un ambiente de creación de servicios con un alto nivel de abstracción y generación automatizada de servicios. Para agilizar el despliegue de servicios, se propone:

- Abstraer las interfaces de redes heterogéneas.
- Definir los componentes reusables de los servicios de comunicaciones.
- Desplegar un ambiente de creación de servicios combinado con la generación de servicios automatizados empleando componentes reutilizables.
- Usar servicios parciales mediante la cooperación de servidores de aplicación.

De esta manera, el proyecto TeamCom descrito en [26] propone una arquitectura integrada para la creación, despliegue y ejecución de servicios dividida en cuatro partes, el ambiente de creación de servicios que incluye el diseño y composición de nuevos servicios, el despliegue del servicio, el motor de ejecución del servicio que contiene uno o más servidores de aplicaciones basados en JAIN SLEE y la capa de transporte que soporta varias redes de telecomunicaciones, dicha arquitectura facilita en gran medida la composición de servicios sin un conocimiento muy detallado de los protocolos de comunicaciones.

En el proyecto presentado en [26] se propone un sistema que permite componer servicios a partir de componentes de servicio de comunicaciones elementales, es decir, mediante su ambiente de creación, permite que un desarrollador establezca el flujo de ejecución de los servicios, de esta manera, este proyecto se clasifica dentro de la composición estática, siendo esta la diferencia principal con el trabajo de grado descrito en esta monografía, además de que no se utiliza el modelo formal de FSM, a pesar de que los servicios son construidos con base en el entorno JAIN SLEE.

2.2.1.6 Un enrutador de aplicación para composición de aplicaciones SIP Servlets

En [44] los autores describen la aplicación de un AR, basado en la arquitectura de composición con características distribuidas (Distributed Feature Composition – DFC). Ésta es una implementación de ECharts [45], que es un lenguaje de composición derivado del lenguaje estandarizado UML Statecharts, basado en FSM para sistemas dirigidos por eventos. La forma como se realiza la lógica de composición es por medio de la identificación y manejo de las solicitudes de servicio, donde cada suscripción a un servicio se configura con una dirección de origen y una dirección de destino y una descripción de funciones; luego se permite fijar el orden de invocación de las funciones de las aplicaciones con base en la fuente y dirección de destino del establecimiento de la llamada y la relación con la información de suscripción a un servicio. Luego por medio de un algoritmo de enrutamiento se establece la selección de las aplicaciones de acuerdo a la información que se fija en un archivo XML de configuración. Esta arquitectura brinda características de simplicidad al trabajar con una configuración estática de composición y de alto rendimiento al tener la capacidad de soportar un gran número de solicitudes SIP.

El DFC-AR es una implementación basada en un lenguaje derivado de FSM, su propósito es seleccionar aplicaciones que respondan a la solicitud de un usuario, basado en la información fijada en un archivo XML, es decir permite realizar una composición estática, por lo tanto no tiene en cuenta cambios eventuales en

tiempo de ejecución los cuales se pueden tratar con el modelo de las FSM como se realiza en este trabajo de grado.

2.2.1.7 Composición de aplicaciones en un ambiente de SIP Servlets

En [46] se describe la transición del SSAPI v1.0 (especificación de java JSR-116) a la versión 1.1 con Solicitud de la especificación de Java 289 (JSR-289 – Java Specification Request 289) [47] definida por el Proceso de la Comunidad Java (JCP - Java Community Process) para el desarrollo y despliegue de servicios SIP, realizada básicamente porque en la versión 1.0 del SSAPI se encontraron algunas deficiencias para la composición de aplicaciones ya que el manejo de invocaciones de varios Servlets no se podía realizar de forma concurrente, generando así una complejidad en la lógica computacional.

El SSAPI define un modelo basado en contenedores comerciales para la construcción de aplicaciones convergentes de servicios de telecomunicaciones utilizando a SIP como protocolo de señalización entre la red, el contenedor y la interacción con la web. Utiliza un modelo de programación similar al de los HTTP Servlets y permite un control y manejo de cabeceras y mensajes del protocolo SIP, con lo cual hace más sencillo el desarrollo para el programador. En la versión 1.1 se introdujo el AR, encargado de invocar y seleccionar las aplicaciones SIP, el AR es un módulo independiente del contenedor, pero está en constante interacción con éste por medio de peticiones que invocan servicios, dando un orden en la ejecución de aplicaciones a cada petición recibida y logrando así un proceso de composición.

Con la introducción del AR es posible realizar composición dentro del contenedor, permitiendo la interacción de uno o más servicios que proporcionen funciones bien definidas para desarrollar nuevas aplicaciones que permitan enriquecer la experiencia de los usuarios; buscando con esto implementar un esquema de diseño de servicios con funciones independientes y con capacidad de reutilización.

En este proyecto se puede observar la descripción del desarrollo de servicios acorde con el concepto de composición implícito en el SSAPI 1.1, además de la descripción de las características del SSAPI 1.1, pero no se encuentra un modelo que permita describir dicha composición como FSM y no esta aplicado al contexto de un SLEE siendo estas las principales diferencias con el actual trabajo de grado.

2.2.2 Trabajos relacionados con composición dinámica en el dominio de los Servicios Web

2.2.2.1 Selección dinámica de Servicios Web para la composición de Servicios Web confiable

En [12] se presenta un modelo basado en FSM que determina un subconjunto de WS que podrán ser invocados y orquestados en tiempo de ejecución para construir un servicio compuesto y en caso de presentarse una falla ser reemplazado por otro servicio que soporte la ejecución de la operación realizada; para lograrlo, se determina que servicio es más confiable para realizar las operaciones del WS compuesto añadiendo una métrica definida como confiabilidad agregada (AggR – Aggregated reliability).

Este proyecto brinda un gran aporte a este trabajo de grado, puesto que se enfoca en la fase de reconfiguración de un servicio, sin embargo existen diferencias muy significativas, las cuales se mencionan a continuación.

- En [12] se realiza composición dinámica en el dominio de los WS, mientras que en el presente trabajo se considera el dominio de las telecomunicaciones, específicamente los servicios que contemplan las características del protocolo SIP adaptados a la lógica que provee el entorno JAIN SLEE.
- La composición en el contexto de los WS realizada en [12] utiliza servicios atómicos para construir un servicio compuesto a partir de los requerimientos de los usuarios, mientras que en el contexto de JAIN SLEE la composición se realiza a nivel de componentes de servicio (SBB), que no son servicios como tal sino módulos que contienen una lógica para procesar eventos, y mediante su composición se da lugar a la conformación de un servicio.

- En el contexto de JAIN SLEE no se considera un servicio compuesto como se hace en [12] (en el que cada operación es una interfaz que permite delegar un servicio para realizar el respectivo procesamiento), debido a que en el SLEE, los módulos (SBB) que componen un servicio son los que tienen la lógica y efectivamente realizan procesamiento de los eventos.
- En el proyecto presentado en [12] se compone y ejecuta un servicio a partir del flujo del servicio de consulta (servicio compuesto) haciendo uso de los servicios atómicos, mientras que en el presente trabajo de grado se asume que el servicio compuesto ya está predeterminado, y en su ejecución el proceso de composición dinámica adapta un componente candidato permitiendo que el flujo de ejecución continúe, en caso de que uno de los componentes del servicio falle.
- El modelo de composición en el proyecto presentado en [12] se construye para todo el flujo de ejecución del servicio compuesto, mientras que en el presente trabajo de grado el modelo de composición está enfocado directamente sobre el SBB que falla, el anterior y posterior a él, sin considerarse todos los SBB que componen el servicio.
- En el presente trabajo de grado se construye un diagrama de composición dinámica basado en máquinas de estado finito en el que sus estados comprenden componentes implicados en la falla del procesamiento de un evento del servicio, los componentes candidatos y algunos estados de sincronización, sin tener en cuenta el concepto de configuración presentado en [12].
- En [12] se proponen el concepto de configuración y de las cadenas de Markov que permiten seleccionar el camino más viable para componer un servicio, dichos conceptos no son considerados en este trabajo de grado puesto que el enfoque es directamente sobre la reconfiguración de los componentes disponibles. El proceso de selección puede pertenecer al descubrimiento de los servicios donde se puede evaluar los parámetros que permitan determinar si un componente es o no apto para soportar los procesos de un componente particular de un servicio, permitiendo así, que la composición dinámica tenga tiempos de respuesta menores (requerimiento indispensable puesto que la reconfiguración se realiza en tiempo de ejecución) debido a que el procesamiento para adaptar un componente, es más corto.

2.2.3 Trabajos relacionados con composición dinámica en el dominio de telecomunicaciones

2.2.3.1 Una plataforma ágil de servicios para ambientes de telecomunicaciones

En el proyecto descrito en [48], se presenta un sistema que permite la interacción de los servicios tradicionales de la industria de telecomunicaciones para disminuir los límites del despliegue y composición de servicios. Puesto que, tanto los operadores de telecomunicaciones como las terceras partes tienen servicios con diferentes funcionalidades, dependientes de su dominio, es necesario permitir que cada uno de ellos tenga acceso a las funcionalidades del otro, para esto, se introducen dos términos, “de adentro hacia afuera” (IO – Inside-Out) para distribuir los servicios internos al mundo externo y “de afuera hacia adentro” (OI – Outside-In) para integrar servicios externos y enriquecer los servicios inherentes de telecomunicaciones con nuevas funcionalidades.

Con el fin de abordar un amplio rango de servicios, el sistema ofrece una solución genérica para distribuir un servicio por varios canales de acceso y así mismo brindar la posibilidad de mejorar el despliegue y composición de servicios. Por esta razón, el agente de servicios (“Service Broker”) presentado en [48] está integrado en la capa superior de una red de nueva generación, la Open SOA Telco Playgraound (OSTP), basado en el proyecto Open IMS Core, de esta manera el agente de servicios puede interconectarse con varios habilitadores de servicios tales como presencia, localización, mensajería, etc. permitiendo componer servicios de telecomunicaciones abordando los principios de SOA e integrando las nuevas funcionalidades que proporcionen los habilitadores.

El proyecto presentado en [48] describe un sistema que permite la interacción de servicios de telecomunicaciones y de las terceras partes, siguiendo los principios de SOA, permitiendo así a un usuario interactuar con servicios que integran diferentes funcionalidades. De esta manera, este proyecto se puede

clasificar dentro de la composición dinámica, puesto que permite la interacción de diferentes componentes adaptándose al flujo de ejecución requerido en tiempo de ejecución. La diferencia principal con este trabajo de grado, es que en [48] no se considera el proceso de reconfiguración de servicios cuando ocurre una falla en uno de sus componentes, así como tampoco se consideran las capacidades de los servicios de un SLEE.

2.2.3.2 Hacia el enfoque flexible de un protocolo de composición basado en FSM

En [14] se presenta un nuevo enfoque para realizar composición de un grupo de protocolos de comunicación, logrando la comunicación en grupo entre diferentes stack de protocolos para desarrollar nuevas aplicaciones tolerantes a fallos. Para lograr dicho objetivo, propone el diseño de un módulo de composición, que establece un lenguaje particular de comunicación entre los diferentes módulos funcionales de los protocolos para permitir la interacción.

Mediante el uso de las FSM, éste trabajo logra un comportamiento dinámico modelando cada protocolo como un módulo FSM. Los módulos de protocolo se comunican entre sí por medio del envío de señales a través de interfaces, éstas son definidas por diferentes especificaciones de programación; y las señales son descritas como notificaciones que una FSM envía a otra. El módulo recibe la señal por una de sus interfaces de entrada, luego la procesa y dependiendo de la notificación de entrada genera una señal de salida. El protocolo de composición, encargado de la coordinación de los bloques funcionales para cumplir una determinada tarea, utiliza tres módulos de interconexión: los adaptadores, adaplexores y aisladores.

- Adaptadores: son una FSM que permite la comunicación entre dos módulos haciendo un emparejamiento entre la señal de entrada de un módulo y la señal de salida de otro, y viceversa. Dicho procedimiento se puede realizar haciendo uso de un emparejamiento sintáctico basado en ontologías, o un emparejamiento semántico. Para lograr la correspondiente coherencia de las señales se utiliza metadatos para entender el significado de las diferentes señales y realizar así su conversión logrando la comunicación.

- Adaplexores: son módulos de interconexión más complejos. Además de proporcionar la misma funcionalidad que el adaptador, permiten realizar las funciones de un multiplexor para que un módulo de protocolo de una capa pueda ser utilizado por múltiples módulos de otra, es decir, permite realizar una comunicación simultánea sobre un módulo con varios módulos de otra capa funcional.

- Aisladores: permiten la protección de una señal en un momento determinado, evitando problemas de interferencias en la comunicación.

El protocolo fue implementado con el lenguaje de descripción y especificación (SDL - Specification and Description Language). SDL es un lenguaje normalizado por la UIT-T en el sector de las telecomunicaciones.

El modelo de programación utilizado por SDL se basa en el uso de FSM. La comunicación se realiza por intercambio de señales, sin compartir memoria. Cada protocolo se encapsula dentro de un bloque. Los bloques están interconectados entre sí, formando protocolos de más alto nivel. Este enfoque da lugar a un modelo de composición flexible que puede ser usado para las jerarquías de grupos de protocolos de telecomunicaciones. Este trabajo presenta un mecanismo de composición dinámico para protocolos de comunicación utilizando FSM y el uso de ontologías.

El concepto de composición en el trabajo descrito anteriormente, se utiliza para permitir la interacción entre protocolos de comunicación, y no se enfoca en composición de servicios con el fin de brindar mejores soluciones a las peticiones de los usuarios, por lo tanto, la principal diferencia de este proyecto con el presente trabajo de grado, es que no está enfocado en composición de servicios SIP en un SLEE.

Capítulo 3

Caracterización Formal de Composición Dinámica de Servicios SIP en un SLEE

3.1 Introducción

En este capítulo se procede a caracterizar la composición dinámica de servicios SIP en un SLEE utilizando la representación formal de FSM. Para esto, se describe el modelo formal del comportamiento de los componentes definidos en la especificación de JAIN SLEE 1.1 y se abordarán las características y la

descripción del modelo de composición dinámica en el SLEE, generado a partir de la adaptación del modelo de composición aplicado al contexto de los WS basado en FSM [12].

3.2 Composición dinámica en el SLEE

Gracias a la comunicación asíncrona, el enrutador de eventos definido por JAIN SLEE, da inicio a la ejecución de los servicios mediante el disparo de eventos, los cuales llegan al SBB raíz del servicio correspondiente. Una vez activada la lógica de funcionamiento del servicio, se lleva a cabo una comunicación síncrona haciendo uso de las relaciones hijas para lanzar los eventos entre los SBB.

La composición soportada por defecto en JAIN SLEE es estática, puesto que ninguno de los métodos de comunicación (síncrona y asíncrona) permite realizar cambios en tiempo de ejecución si ocurre un suceso que no se haya tenido en cuenta en la etapa de desarrollo, dicha labor es propia de la composición dinámica ya que se encarga de adaptar los cambios que no fueron tenidos en cuenta en fase de diseño.

El SLEE es un entorno orientado a composición, sin embargo es evidente la ausencia de métodos que ayuden a corregir fallos inesperados de sus componentes en tiempo de ejecución, por esta razón en este trabajo de grado, la composición dinámica de servicios en el SLEE es un proceso capaz de adaptarse a cambios inesperados que afecten el funcionamiento normal de un servicio, es a partir de este concepto que se puede considerar y resolver el problema de inactividad inesperada de algún servicio en tiempo de ejecución, en este sentido se puede decir que, si en algún momento un componente de un servicio falla, la composición dinámica será la encargada de encontrar un componente de las mismas características y reordenar el servicio reemplazando el componente y efectuando nuevamente su funcionamiento.

Por lo tanto, teniendo en cuenta que en el SLEE no ha sido planteado un método que responda a las fallas que pueda tener un componente de un servicio en tiempo de ejecución y debido a la gran importancia que tiene en el dominio de las telecomunicaciones un proceso que le permita a un servicio adaptarse a cambios inesperados, no tenidos en cuenta en la etapa de desarrollo, se considera adecuado proporcionar mecanismos que realicen composición dinámica de servicios SIP en el SLEE, con capacidades de reconfiguración. Dicha composición, se propone como un método aplicado a nivel de bloques de construcción de servicios (SBB). Además, como ya se ha definido, solo se tendrán en cuenta las interacciones entre los componentes que pertenezcan a servicios SIP, descartando así la posibilidad de contemplar servicios convergentes formados por la interacción de componentes de diferentes dominios tales como la Web y telecomunicaciones.

Con el fin de llevar a cabo el proceso de composición dinámica de servicios en el SLEE basado en FSM, se han definido unas fases como resultado de la investigación realizada en el presente proyecto.

3.2.1 Fases

La composición dinámica, es un proceso que debe consumir un tiempo mínimo, casi imperceptible, puesto que se lleva a cabo en tiempo real e impacta el rendimiento de los servicios que lo integren. En el entorno SLEE, este criterio es muy importante puesto que integra servicios de telecomunicaciones tal como los servicios SIP, donde los usuarios se ven afectados por los tiempos de respuesta que influyen en los servicios que utilizan [49].

El descubrimiento de servicios es un proceso ligado al concepto de composición puesto que determina los componentes necesarios para poder conformar el flujo de ejecución de un servicio, sin embargo, es necesario realizar esta fase con anterioridad y no inmersa en la composición dinámica para evitar que sus procedimientos afecten el rendimiento de la composición dinámica en el SLEE.

En este trabajo de grado se considera necesario realizar un proceso de selección de componentes en la fase del descubrimiento (anterior a la composición dinámica) con el fin de depurar los candidatos disponibles para un servicio.

A continuación se describe la fase de descubrimiento previa a la composición de servicios y las fases de composición dinámica.

3.2.1.1 Descubrimiento de servicios

El descubrimiento de servicios tiene la finalidad de encontrar servicios según los requisitos que tenga un cliente particular [20, 50, 51], en el caso del entorno JAIN SLEE, en la fase de descubrimiento se necesita encontrar los candidatos que puedan llevar a cabo el funcionamiento de los componentes de un servicio. Para ello es necesario un proceso de selección que permita depurar el proceso de descubrimiento con el fin de entregar los candidatos más viables para llevar a cabo el proceso de composición.

La fase de selección, comprendida en el descubrimiento, tiene la finalidad de definir un conjunto de candidatos aptos para ejecutar los eventos que ejecuta cada componente del flujo de ejecución de un servicio en el SLEE y generar la respuesta acertada, para ello se pueden llevar a cabo técnicas de QoS tal como la confiabilidad contemplada en [12] para obtener una probabilidad de ejecución del componente o técnicas como la sintáctica o semántica que permitan realizar una comparación a nivel de palabras entre los componentes candidatos y los componentes del servicio original.

El conjunto de candidatos definido por la fase de selección es clasificado por servicios y SBB, de tal manera que en el proceso de composición dinámica se puedan identificar los candidatos específicos que pueden reemplazar el procesamiento de un evento realizado por un SBB particular de un servicio.

El proceso de selección requiere de tiempos de respuesta pronunciados como se afirma en [12] por lo tanto no es factible contemplar su ejecución dentro del proceso de composición dinámica, por esta razón se considera necesario realizarla en la etapa de descubrimiento, previa a la composición. En este trabajo de grado el proceso de descubrimiento y su fase de selección se asumen como entradas para llevar a cabo la ejecución de los mecanismos de composición dinámica.

3.2.1.2 Síntesis y orquestación

La composición de servicios, involucra dos conceptos muy importantes, la síntesis y la orquestación[17, 52], la síntesis es un proceso que permite generar un plan para lograr el comportamiento deseado de los servicios, a través de la combinación de múltiples componentes y la orquestación permite coordinar el flujo de control y datos entre los componentes del plan definido en la síntesis.

Los procesos de síntesis y orquestación, al igual que el proceso de descubrimiento, requieren de tiempos elevados de respuesta, como se puede confirmar en [53-56], de esta manera, se considera necesario, realizar estos procesos en la etapa de diseño previa a la ejecución de servicios.

3.2.1.3 Monitoreo

Esta fase se encarga de identificar la falla en los servicios en tiempo de ejecución y enviar la respectiva notificación al SLEE, para ello se requiere que se hayan establecido puntos de control que ayuden en la detección del problema.

La fase de monitoreo, está encargada de detectar fallas tanto en los SBB del servicio original como en los SBB candidatos. Cuando ocurre una falla en un SBB del servicio original, la fase de monitoreo la detecta, y es la encargada de enviar un evento de alarma al SLEE para que sea entregada al módulo de composición dinámica. Aunque la fase de descubrimiento se encarga de depurar los candidatos, estos también pueden fallar al momento de procesar un evento que falló en el flujo de ejecución de un servicio, por lo tanto es necesario realizar el monitoreo de estos cuando se esté realizando el proceso de composición dinámica, de esta manera, si ocurren fallas en los candidatos, la fase de monitoreo se encarga de enviar la notificación a la fase de reconfiguración.

Existen herramientas provistas por JBOSS, que ayudan a la administración y monitoreo de todas las funcionalidades de JAIN SLEE, incluyendo los servicios, eventos y componentes SBB, tales como la consola JMX y JOPR [57]; a partir de estas herramientas, es posible conocer lo que está sucediendo en un servicio determinado de forma externa al SLEE, esto deja entrever una solución automatizada que en colaboración con estas herramientas ayuda al proceso de detección de una falla en los servicios y así permita enviar un evento al SLEE para realizar el proceso de composición dinámica mediante el módulo provisto en este trabajo de grado.

3.2.1.4 Reconfiguración

En esta fase se efectúa la reconfiguración del flujo de ejecución con el fin de que ejecute un evento que no haya podido ejecutar un componente particular de un servicio, pasando dicho evento a cada candidato hasta que uno de ellos tenga una respuesta exitosa, si ninguno de los candidatos tiene una respuesta exitosa, se determina que el servicio no puede ser reconfigurado y es necesario realizar nuevamente un descubrimiento de servicios para seleccionar nuevos candidatos que puedan ser parte de la reconfiguración del servicio.

Antes ejecutarse el algoritmo de reconfiguración, esta fase confirma si un candidato determinado está activo para realizar el procesamiento de un evento que haya fallado en un servicio, confirmando que puede llevar a cabo el procesamiento realizado por el componente correspondiente del servicio a componer. Una vez que se determine cuáles candidatos están activos, esta fase entrega este conjunto a la fase de reconfiguración para iniciar el proceso de ejecución del evento que falló en el servicio.

En el dominio de telecomunicaciones, se requieren procesos que den solución en tiempo real a fallas que puedan ocurrir en el flujo normal del servicio, por esta razón, en este trabajo de grado se implementó específicamente esta fase debido a que se considera como la más crítica del proceso de composición dinámica ya que debe dar respuesta en tiempo de ejecución a un evento que no se haya procesado en el flujo de ejecución normal de un servicio. Esta fase utiliza para su funcionamiento un algoritmo de reconfiguración basado en [12], considerando que en este trabajo de grado, la reconfiguración se realiza de forma atómica, es decir enfocándose en el flujo definido por el componente que falló y los componentes anterior y siguiente, sin considerar todo el flujo de ejecución del servicio.

3.2.2 Modelo de composición en el SLEE

Teniendo en cuenta el modelo formal del enrutador de eventos de la especificación de JAIN SLEE 1.1 [33] y utilizando los conceptos definidos en [12], a continuación se describen formalmente los conceptos de composición dinámica para un SLEE.

3.2.2.1 Definición 1. $Service_{SLEE}$.

Para definir formalmente un servicio en el SLEE es necesario adoptar dos definiciones matemáticas, la definición de evento y de componente SBB, que permiten describir el comportamiento e interacción de los componentes de un servicio.

- **Evento**

Si se dice que $\boldsymbol{EventType}$ es el conjunto de tipos de eventos y que $\boldsymbol{Event}$ es el conjunto de eventos, un evento $\boldsymbol{e}$, es una 5-tupla ordenada:

$$(\boldsymbol{e_1}, \boldsymbol{e_2}, \boldsymbol{e_3}, \boldsymbol{e_4}, \boldsymbol{e_5}) \in \boldsymbol{Z} \times \boldsymbol{EventType} \times \boldsymbol{ActivityContext} \times \boldsymbol{Address} \times \boldsymbol{P}(\boldsymbol{service})$$

Dónde:

$\boldsymbol{e_1}$: es la identidad del evento.

$\boldsymbol{e_2}$: es el tipo de evento.
$\boldsymbol{e_3}$: es el contexto de la actividad en la que se generó el evento.
$\boldsymbol{e_4}$: es la dirección que el SLEE ha determinado para el evento.
$\boldsymbol{e_5}$: es el conjunto de servicios.

- **Componente SBB**

Si se dice que $\boldsymbol{SBBComponent}$ es el conjunto de todos los componentes SBB, un componente SBB $\boldsymbol{c}$ es una 5-tupla:

$$(\boldsymbol{c_1 c_2}, \boldsymbol{c_3}, \boldsymbol{c_4}, \boldsymbol{c_5}) \in \boldsymbol{P}(\boldsymbol{ChildRelation}) \times \boldsymbol{P}(\boldsymbol{EventType}) \times \boldsymbol{P}(\boldsymbol{EventType}) \times \boldsymbol{P}(\boldsymbol{EventType}) \times \boldsymbol{P}(\boldsymbol{EventName})$$

Dónde:

$\boldsymbol{c_1}$: es el conjunto de relaciones hijas del componente SBB.
$\boldsymbol{c_2}$: es el conjunto de tipos de eventos iniciales del componente SBB.
$\boldsymbol{c_3}$: es el conjunto de tipos de eventos recibidos del componente SBB.
$\boldsymbol{c_4}$: es el conjunto de tipos de eventos mask-on-attach del componente SBB.
$\boldsymbol{c_5}$: es el conjunto de nombres de eventos mask-on-attach del componente SBB.

Para un componente SBB dado, $\boldsymbol{c_2}, \boldsymbol{c_4} \subseteq \boldsymbol{c_3}$

Utilizando las anteriores definiciones, un servicio en el SLEE se puede representar mediante una FSM de la siguiente forma:

$$\boldsymbol{Service_{SLEE}} = (\boldsymbol{ServiceEvent}, \boldsymbol{ServiceSBBComponent}, \boldsymbol{sbb_{root}}, \boldsymbol{\delta_{SLEE}}, \boldsymbol{FinalSBBComponent}).$$

Dónde:

- $\boldsymbol{ServiceEvent}$ *es el conjunto de eventos que forman parte del servicio, tal que:*

  $$\boldsymbol{ServiceEvent} \subseteq \boldsymbol{Event},$$

  Donde $\boldsymbol{Event}$ *es el conjunto de eventos del SLEE definido en el modelo formal del enrutador de eventos.*

- $\boldsymbol{ServiceSBBComponent}$ *es el conjunto de componentes SBB que conforman el servicio, tal que:*

  $$\boldsymbol{ServiceSBBComponent} \subseteq \boldsymbol{SBBComponent},$$

  Donde $\boldsymbol{SBBComponent}$ *es el conjunto de componentes SBB del SLEE, definido en el modelo formal del enrutador de eventos.*

  De esta manera, Los estados de ejecución del flujo del servicio son representados a partir de sus componentes en el orden de procesamiento de sus eventos.

- $\boldsymbol{sbb_{root}}$ *es el componente SBB raíz del servicio, representa el inicio del servicio, tal que:*

  $$\boldsymbol{sbb_{root}} \in \boldsymbol{ServiceSBBComponent}.$$

  El disparo de un evento inicial $\boldsymbol{e_i}$ por parte del SLEE (representado por el ER) ejecuta el SBB raíz, con lo cual se inicia el servicio y con él, la ejecución de sus componentes tal como se haya definido durante su implementación.

- $\boldsymbol{\delta_{SLEE}}$ *es la función de transición definida para establecer el flujo de ejecución del servicio en el SLEE. La función de transición se define de la siguiente forma:*

$$\boldsymbol{\delta_{SLEE}} : \boldsymbol{ServiceSBBComponent} \times \boldsymbol{ServiceEvent} \rightarrow \boldsymbol{ServiceSBBComponent.}$$

  La función de transición retorna el nuevo componente SBB encargado de procesar un evento en particular disparado por un componente anterior, es decir, los estados de ejecución del flujo del servicio se definen con la ejecución de sus componentes dependiendo del evento que sea procesado en un componente SBB.

- $\boldsymbol{FinalSBBComponent}$ *se define como el conjunto de estados finales del servicio. Es representado por los componentes SBB en los cuales se termina el flujo del servicio, tal que:*

$$\boldsymbol{FinalSBBComponent} = \{\boldsymbol{finalSbb_0}, \boldsymbol{finalSbb_1}, \dots, \boldsymbol{finalSbb_n}\}.$$
Para todo $\boldsymbol{n} \in \boldsymbol{Z^+}$.Donde
$$\boldsymbol{FinalSBBComponent} \subseteq \boldsymbol{ServiceSBBComponent}.$$

La anterior definición de servicio en JAIN SLEE como FSM, permite modelar el comportamiento normal de un servicio, es decir, su flujo de ejecución para realizar su lógica de funcionamiento. La FSM definida abstrae la lógica establecida en fase de desarrollo de un servicio por medio de la representación del flujo de ejecución de los SBB asumiendo el funcionamiento correcto del servicio.

En la figura 5 se observa el diagrama que representa un servicio en el SLEE como una FSM construida a partir de la definición de $\boldsymbol{Service_{SLEE}}$, donde los estados del servicio son representados como óvalos verdes y los eventos como aristas dirigidas. Se modela un servicio conformado por cuatro SBB, donde cada componente SBB representa un estado del servicio; el SBB raíz inicia la lógica del servicio basada en el disparo de eventos, donde la arista dirigida representa el disparo de un evento desde un SBB a otro, de esta manera el flujo de ejecución se puede construir utilizando la función de transición, partiendo desde el SBB raíz hasta llegar al final del flujo del servicio, representado por los componentes SBB en los cuales se termina la lógica de funcionamiento del servicio, en la figura 5 el $\boldsymbol{SBB_4}$ representa el estado final del servicio.

También se debe mencionar que para el modelado de un servicio, no es necesario conocer la lógica interna de los SBB, sino de los eventos recibidos y disparados entre ellos, con lo que se construye el orden de ejecución de los mismos. Los eventos son entregados y a su vez procesados en un SBB que genera como resultado el disparo de otro evento dirigido a otro SBB para seguir con la lógica del servicio, pero en algunos casos, dependiendo de esta lógica, los eventos disparados son dirigidos al exterior por medio del RA y como resultado generan un nuevo evento entrante debido a una acción en la red, este evento es el dirigido al siguiente SBB en el flujo del servicio; sin embargo, este proceso no es necesario describirlo ni modelarlo en la FSM que representa un servicio puesto que para realizar composición, solo es necesario tener conocimiento de la interacción entre los SBB de un servicio, es decir, es irrelevante tener en cuenta, si un evento $\boldsymbol{e_{in}}$ recibido por un $\boldsymbol{SBB_1}$, fue disparado directamente por el $\boldsymbol{SBB_0}$ anterior o como resultado de un evento $\boldsymbol{e_{out}}$ (lanzado por el $\boldsymbol{SBB_0}$) que interactuó con el RA.

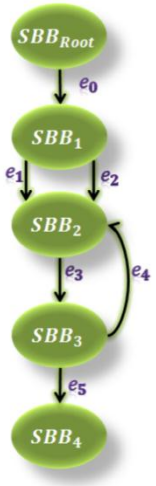

Figura 5. Diagrama de representación de un servicio en el SLEE

El principal objetivo de esta definición es abstraer el orden de ejecución de los SBB a partir de la secuencia de eventos que requieren ser procesados por estos componentes, logrando determinar el flujo normal de ejecución del servicio, dicho flujo ayudará a deducir el comportamiento predeterminado de los servicios de esta plataforma para poder encontrar diferentes alternativas de componer los servicios cuando uno de sus componentes falle.

3.2.2.2 Definición 2. $CandidateServiceSBBComponent$

Se define como el conjunto de componentes SBB candidatos para componer un servicio en el SLEE, este conjunto posee componentes SBB que han sido elegidos para reemplazar los componentes SBB que originalmente conforman un servicio, en el caso de fallar en el procesamiento de un evento en particular.

$$\text{Para todo } \boldsymbol{ServiceSBBComponent} = \{\boldsymbol{sbb_0}, \boldsymbol{sbb_1}, \dots, \boldsymbol{sbb_i}, \dots, \boldsymbol{sbb_n}\} \ \exists \ un \ \boldsymbol{CandidateServiceSBBComponent} = \{\boldsymbol{sbbCandidate_{00}}, \dots, \boldsymbol{sbbCandidate_{i1}}, \dots, \boldsymbol{sbbCandidate_{ik}}, \dots, \boldsymbol{sbbCandidate_{nm}}\}, tal\ que\ \boldsymbol{sbb_i} \rightarrow \boldsymbol{sbbCandidate_{i1}}, \boldsymbol{sbbCandidate_{i2}}, \dots, \boldsymbol{sbbCandidate_{ik}}, \text{ donde } \boldsymbol{0 \leq i \leq n, 0 \leq k \leq m},$$
$$\textit{Para todo } \boldsymbol{i, k, m, n} \in \mathbb{Z}^+. \textit{ Donde}$$
$$\boldsymbol{CandidateServiceSBBComponent} \subseteq \boldsymbol{SBBComponent}.$$
$$\boldsymbol{ServiceSBBComponent} \subseteq \boldsymbol{Service_{SLEE}}.$$

Para todo elemento del conjunto $\boldsymbol{ServiceSBBComponent}$ *pueden existir cero o más elementos similares en el conjunto de* $\boldsymbol{CandidateServiceSBBComponent}$.

Los SBB candidatos son seleccionados de acuerdo a la similaridad que tengan con los SBB del servicio original, basada en la capacidad de soportar los eventos que sean procesados por ellos, con el fin de realizar la lógica de funcionamiento modelada en el flujo del servicio. La selección, hace parte de la etapa de descubrimiento, donde se escogen los componentes candidatos que pueden reemplazar un componente del servicio original por medio de procesos sintácticos o semánticos de comparación y con técnicas de QoS como se realiza en [12] con el concepto de confiabilidad agregada. La etapa de descubrimiento, es un proceso previo a la composición de servicios, por lo tanto, en este trabajo de grado se asume que los componentes SBB candidatos ya están descubiertos debido a que los objetivos del mismo se centran en un proceso de composición dinámica con capacidades de reconfiguración.

En la figura 6 se define una posible relación entre un conjunto de SBB candidatos de un servicio dentro del SLEE, la cual consiste, en que cada componente SBB que conforma el servicio posee un número de candidatos que están en capacidad de reemplazarlo basado en los eventos que procese. Como se aprecia en

la figura 6 se puede considerar que un SBB candidato puede soportar procesamiento de eventos de varios SBB del servicio original, de este modo, pueden existir SBB candidatos que soporten toda la lógica de procesamiento de eventos de un servicio, pero debido a la característica de granularidad fina que posee la arquitectura del SLEE se busca encontrar varios SBB candidatos que soporten uno o varios eventos procesados por los SBB del servicio original y hagan parte de la lógica total del servicio con el fin de generar mayor número de opciones para componer el servicio.

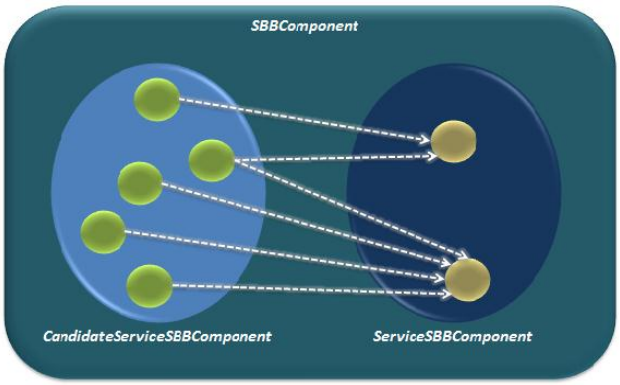

Figura 6. Conjunto de componentes SBB candidatos

Hasta el momento han sido adaptadas las definiciones formales de servicio compuesto y comunidad del modelo de composición de los WS [12] al contexto de los servicios de JAIN SLEE y se han encontrado algunas diferencias entre estos dominios, las cuales se describen a continuación.

En el caso del servicio compuesto, en los WS, es un servicio de consulta que exclusivamente plantea el flujo del servicio, es decir, no puede ejecutar operaciones por sí solo, para esto, es necesario buscar en una comunidad de servicios atómicos (servicios con lógica definida, capaces de ejecutar operaciones) los delegados que soporten las operaciones expuestas en el WS compuesto. En contraste, en el SLEE, un servicio compuesto (concepto abstraído de la definición de $\boldsymbol{Service_{SLEE}}$) está conformado por componentes SBB que interaccionan entre sí por medio de eventos, para ejecutar la lógica de un servicio. Cabe resaltar que en este proyecto es acogida la composición síncrona realizada entre SBB para modelar el comportamiento de un servicio real.

El concepto de comunidad, en el contexto de los WS, la comunidad es un conjunto de WS atómicos capaces de procesar operaciones y además, cada WS atómico de la comunidad soporta al menos una operación del WS compuesto; en el SLEE, este concepto es representado por un conjunto de SBB candidatos definido en la representación formal $\boldsymbol{CandidateServiceSBBComponent}$, donde a cada servicio desarrollado en el SLEE le corresponde un conjunto que contiene SBB con características similares a los SBB del servicio original; La diferencia principal radica en que la comunidad de los WS contiene servicios y cada uno de ellos puede funcionar independientemente, mientras que el conjunto de SBB candidatos en el SLEE contiene componentes SBB, que no necesariamente pueden funcionar como un servicio, sino que podría ser indispensable la interacción con al menos otro SBB para definir la funcionalidad de un servicio.

3.2.2.3 Definición 3. $\boldsymbol{Composition_{SLEE}}$

El SLEE realiza un procesamiento secuencial de los eventos de un servicio, estableciendo así, el orden de ejecución de los componentes SBB, de esta manera se descarta la posibilidad de que ocurran fallas en paralelo o en un mismo instante dentro de un servicio, por lo tanto, se propone modelar la composición dinámica de forma atómica, es decir, teniendo en cuenta el componente SBB que falló al procesar un evento y los SBB anterior y siguiente a este, descartando todos los demás SBB que hacen parte del flujo de ejecución del servicio.

A continuación se presenta la definición formal de la FSM de composición encargada de modelar la etapa de reconfiguración de un servicio en el SLEE:

FSM $\boldsymbol{Composition_{SLEE}}$

Si se dice que $\boldsymbol{SBB_{fail}}$ es un componente SBB que falla en la ejecución de un servicio en el SLEE, entonces la composición a partir del conjunto de SBB candidatos determinados para dicho componente, representados en $\boldsymbol{CandidateServiceSBBComponent}$, que soportan un evento particular definido como $\boldsymbol{event_i}$ en el flujo del servicio representado en $\boldsymbol{Service_{SLEE}}$ se denota así:

$$\boldsymbol{Composition_{SLEE}}(\boldsymbol{SBB_{fail}}, \boldsymbol{event_i}, \boldsymbol{Service_{SLEE}}, \boldsymbol{CandidateServiceSBBComponent}) = (\boldsymbol{EventComp}, \boldsymbol{SBBComp}, \boldsymbol{SynchroState}, \boldsymbol{initialSBBComp}, \boldsymbol{\delta_{Comp}}, \boldsymbol{finalSBBComp})$$

Tal que $\boldsymbol{SBB_{fail}} \in \boldsymbol{ServiceSBBComponent} \wedge \boldsymbol{event_i} \in \boldsymbol{ServiceEvent}$, *donde* $\boldsymbol{ServiceSBBComponent}, \boldsymbol{ServiceEvent} \subseteq \boldsymbol{Service_{SLEE}}$.

Dónde:

- $\boldsymbol{EventComp}$ *se define como el conjunto de eventos de composición que está conformado así:*

$$\boldsymbol{EventComp} = \{\boldsymbol{e_i}, \boldsymbol{e_j}, \boldsymbol{e_{fail}}, \boldsymbol{e_{wrong}}\}$$
$$\textit{para todo } \boldsymbol{i, j} \in \mathbb{Z}^+ \wedge \boldsymbol{e_i}, \boldsymbol{e_j} \in \boldsymbol{ServiceEvent}, \boldsymbol{donde}$$
$$\boldsymbol{ServiceEvent} \subseteq \boldsymbol{Service_{SLEE}}.$$

  Dónde:

  - $\boldsymbol{e_i}$ *representa al evento particular que no pudo procesar un SBB del servicio original.*
  - $\boldsymbol{e_j}$ *representa al evento resultado del procesamiento del evento $\boldsymbol{e_i}$, dependiendo de la lógica del servicio es posible que haya un evento resultado del procesamiento del evento que falló, por lo cual en el conjunto se establece por defecto el evento resultado $\boldsymbol{e_j}$, en caso de no existir se omite.*
  - $\boldsymbol{e_{fail}}$ *es definido como un evento de falla, representa el fallo del procesamiento de $\boldsymbol{e_i}$ por parte de un SBB candidato.*
  - $\boldsymbol{e_{wrong}}$ *es definido como evento de error, representa el fallo definitivo del procesamiento de $\boldsymbol{e_i}$ y se produce cuando ninguno de los candidatos pueda procesarlo.*

- $\boldsymbol{SBBComp}$ *se define como el conjunto de los componentes SBB de composición. Está formado por los SBB involucrados en la falla del servicio como son el SBB que dispara el evento que no es procesado con éxito, el SBB que falla en el procesamiento de dicho evento y el SBB que recibe el evento resultado del procesamiento fallido del evento disparado; además también hacen parte de este conjunto los SBB candidatos del SBB que falla en procesar el evento.*

  Su representación formal se describe a continuación:

$$\boldsymbol{SBBComp} = \{\boldsymbol{sbbComp_{fail}}, \boldsymbol{sbbComp_{previous}}, \boldsymbol{sbbComp_{next}}, \boldsymbol{sbbComp_{candidate_i}}, \dots, \boldsymbol{sbbComp_{candidate_n}}\}$$
$$\textit{Para todo } \boldsymbol{i, j} \in \mathbb{Z}^+ \textit{ . tal que}$$
$$\boldsymbol{sbbComp_{fail}}, \boldsymbol{sbbComp_{previous}}, \boldsymbol{sbbComp_{next}} \in \boldsymbol{Service_{SLEE}} \wedge$$
$$\boldsymbol{sbbComp_{candidate_i}}, \dots, \boldsymbol{sbbComp_{candidate_n}} \in \boldsymbol{CandidateServiceSBBComponent}.$$
$$\textit{Por lo tanto} \quad \boldsymbol{SBBComp} = \boldsymbol{Service_{SLEE}} \cap \boldsymbol{CandidateServiceSBBComponent}.$$

  Dónde:

- $\boldsymbol{sbbComp_{fail}}$ *representa el SBB del servicio original que falla en el procesamiento de un evento particular.*
- $\boldsymbol{sbbComp_{previous}}$ *representa el SBB anterior al SBB que falla en el flujo del servicio original, el cual es el SBB encargado de disparar el evento que no es procesado con éxito.*
- $\boldsymbol{sbbComp_{next}}$ *representa el SBB siguiente al SBB que falla en el flujo del servicio original; el cual es el SBB encargado de recibir el evento resultado del procesamiento del evento que falla. Como se mencionó anteriormente dependiendo de la lógica del servicio es posible que no exista un evento resultado en ese caso se omite del conjunto este componente SBB.*
- $\boldsymbol{sbbComp_{candidate_i}, \dots, sbbComp_{candidate_n}}$ *representan los SBB candidatos del SBB del servicio original que falló en el procesamiento de un evento en particular.*

El conjunto de componentes SBB de composición más los estados de sincronización, definidos en el siguiente ítem, representan los estados de la FSM de composición definida en este modelo.

- $\boldsymbol{SynchroState}$ *se define como el conjunto de estados de sincronización de la composición del servicio; su función fundamental es controlar en el flujo de composición determinando si se realizó con éxito la reconfiguración del servicio, la cual se lleva a cabo por medio de la sustitución del componente que falló, por uno de los candidatos disponibles. También garantiza el seguimiento normal del orden de ejecución preestablecido para dicho servicio. A continuación se presenta su definición formal:*

$$\boldsymbol{SynchroState} = \{\boldsymbol{reconfiguration}, \boldsymbol{un-luck}, \boldsymbol{failure}\}$$

Dónde:

- $\boldsymbol{reconfiguration}$ *representa el estado de composición que controla la etapa encargada de reemplazar el componente SBB del servicio original que fallo por su candidato.*
- $\boldsymbol{un-luck}$ *representa el estado de composición que determina cuando un SBB candidato no pudo procesar con éxito el evento del servicio.*
- $\boldsymbol{failure}$ *representa el estado de composición que determina que la etapa de reconfiguración falló debido a que todos los SBB candidatos disponibles no pudieron procesar el evento con éxito.*

- $\boldsymbol{initialSBBComp}$ *se define como el estado inicial de la composición, el cual es representado por el* $\boldsymbol{sbbComp_{previous}}$ *tal que:*

$$\boldsymbol{initialSBBComp = SbbComp_{previous}, \text{ donde}}$$
$$\boldsymbol{sbbComp_{previous} \in SBBComp}$$

- $\boldsymbol{\delta_{Comp}}$ *se define como la función de transición de la composición del servicio, representa al flujo de la composición de la siguiente forma:*

$$\boldsymbol{\delta_{Comp}}: \boldsymbol{EventComp} \times \boldsymbol{SBBComp} \rightarrow (\boldsymbol{SBBComp} \vee \boldsymbol{SynchroState})$$

La función de transición retorna el nuevo estado en la composición a partir del procesamiento de un evento por parte de un determinado SBB de composición en referencia al flujo de ejecución del servicio o los estados de sincronización utilizados en la lógica de la FSM de composición.

- $\boldsymbol{finalSBBComp}$ *se define como el estado final de la composición, el cual puede tomar valores dependiendo del resultado de la composición, en el caso que la composición sea realizada con éxito será representado por el* $\boldsymbol{sbbComp_{next}}$ *y en el caso que falle será representada por estado de sincronización* $\boldsymbol{failure.}$

3.2.2.4 Relación de los SBB de un servicio con los SBB candidatos

La FSM $Composition_{SLEE}$ plantea modelar la composición a partir del flujo de ejecución del servicio y el conjunto de SBB candidatos para un determinado componente SBB del servicio original que presente alguna falla durante el procesamiento de un evento particular, estos conceptos fueron adaptados del modelo de composición de los WS [12], con la diferencia de que en este modelo, la composición se realiza para un determinado componente que falla en el flujo predeterminado del servicio a partir de sus SBB candidatos dependiendo del evento en particular que se procese, según esto se determina el candidato adecuado para lograr la composición del servicio por medio de la sustitución del componente y permitir así la continuación de la lógica con la interacción hacia el siguiente componente en el flujo del servicio en tiempo de ejecución; mientras en el contexto de los WS se realiza la composición creando diferentes flujos a partir de la ejecución de las operaciones por parte de los WS atómicos en referencia al flujo determinado en el WS compuesto con el objetivo de determinar el flujo de ejecución del servicio desde el inicio hasta el final del mismo para construir el servicio en tiempo real a partir de las solicitudes del usuario.

En la figura 7 se observa el flujo de ejecución de un servicio en el SLEE modelado a partir de la definición de $\boldsymbol{Service_{SLEE}}$, y su respectiva relación con el conjunto de los SBB del servicio original $\boldsymbol{ServiceSBBComponent}$, donde cada SBB que conforma el servicio está ubicado según en el orden que determina el procesamiento de los eventos. Además se presenta el conjunto de los SBB candidatos del servicio $\boldsymbol{CandidateServiceSBBComponent}$ con la relación del SBB del servicio original al que pueden reemplazar.

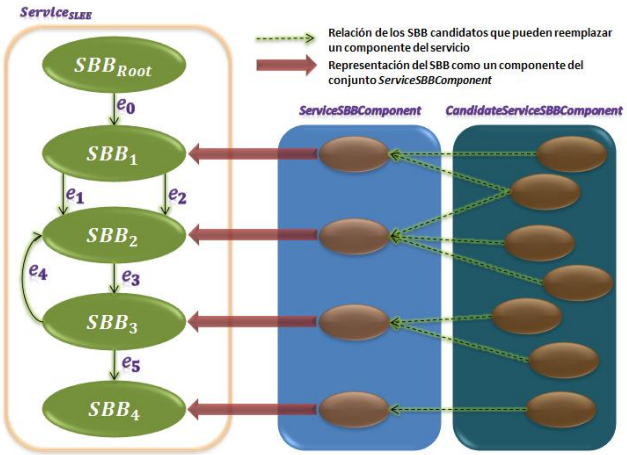

Figura 7. Flujo del servicio y relaciones entre los SBB del servicio original y SBB Candidatos

En la figura 7 se puede observar que para el SBB raíz no existen candidatos, esto es debido a que este componente tiene unas limitaciones para realizar la composición. La primera limitación se deriva del hecho de que el SBB raíz representa al servicio en el SLEE, y es el único que puede recibir los eventos que lance el ER, de esta manera, si el SBB raíz del servicio falla, también lo hace el servicio en sí, la segunda limitación es debida a que dicho SBB tiene una funcionalidad especial, y es la de servir como mediador entre la lógica del servicio y el módulo de composición dinámica para poder realizar la reconfiguración del servicio con el SBB candidato para procesar el evento que falló al ser procesado por el SBB del servicio original, específicamente encargándose de introducir el evento disparado del candidato en el flujo del servicio para continuar con su lógica establecida, esto es descrito con detalle en la sección 4.4.3.

3.2.2.5 Ejemplo de composición dinámica

Con el fin de ilustrar la definición de la composición dinámica $Composition_{SLEE}$, se plantea la falla del procesamiento del evento e_1 realizado por el SBB_2 en el flujo de ejecución del servicio $\boldsymbol{Service_{SLEE}}$ de la figura 7. Dicha falla impide que el evento e_3 sea lanzado, por lo tanto el flujo de ejecución del servicio queda inconcluso. Para evitar la interrupción del flujo de ejecución del servicio, existen tres candidatos que pueden reemplazar el SBB_2, como se puede observar en la figura 7, de esta manera, a partir del flujo de ejecución del servicio y de los candidatos disponibles, en la figura 8 se modela la FSM $Composition_{SLEE}$ para el fallo del componente SBB_2.

En la figura 8 se puede observar que la FSM modela los "caminos" posibles de composición que forman los SBB candidatos para reemplazar el componente que falla, los cuales están controlados por los estados de sincronización que permiten determinar si un SBB candidato reconfigura el servicio con éxito o falló en su objetivo.

La composición se modela en el orden del flujo del servicio, donde el estado inicial está representado por el componente SBB_1, que corresponde al SBB anterior en el flujo del servicio, de allí en adelante, se modela la falla del procesamiento del evento e_1 en el componente SBB_2 y de allí se modela la etapa de reconfiguración en el estado de sincronización $reconfiguration$ donde se genera las diferentes alternativas para reemplazar el componente que falló dependiendo del procesamiento del evento e_1 que pueden realizar los diferentes SBB candidatos y en referencia al flujo del servicio se genera el evento resultante e_3 que debe ser procesado por el componente SBB siguiente, que en este caso corresponde al SBB_3, con lo que se establece un posible estado final que representa el éxito del proceso y en caso de que un componente candidato falle en el procesamiento del evento es representado en el estado de sincronización $un-luck$, de allí regresa al estado de $reconfiguration$ para repetir el proceso con otro SBB candidato y en caso de que todos los candidatos disponibles fallen se representa el fallo definitivo para reconfigurar al servicio por medio del estado de sincronización $failure$, así modelando el otro posible estado final de la composición.

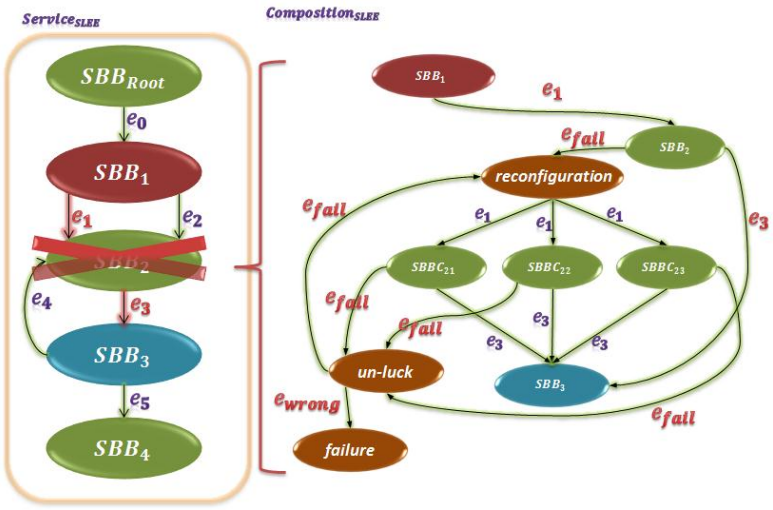

Figura 8. Composición de un servicio en el SLEE a partir de los SBB candidatos

También se puede observar en la figura 8 que existe un evento e_4 que puede ser disparado del SBB_3, sin embargo, esta situación no se ha considerado en el caso del diagrama de composición, ya que para este caso solo se plantea considerar el flujo que admite al evento e_3 como resultado del procesamiento de e_1. De esta manera, no está garantizado que el componente candidato tenga las condiciones apropiadas para soportar el evento e_4, si este evento se lanza puede existir una nueva falla, si esto ocurre, se volverá a detectar el fallo y será repetido el proceso de composición dinámica para el componente SBB_2 pero en este caso

teniendo en cuenta que el evento e_4 fue el que falló. Debido al modelo de enrutamiento de eventos del SLEE caracterizado por ser secuencial, es decir, permite solo procesar un evento a la vez en un SBB y de forma concurrente cuando se ejecutan varios SBB al mismo tiempo, este modelo de composición dinámica se consideró para un componente de servicio a la vez cuando ocurra una falla sobre él.

3.2.2.6 Algoritmo de composición dinámica de servicios

El algoritmo de composición dinámica de servicios en el SLEE ha sido adaptado del algoritmo de selección basado en confiabilidades agregadas (*AR_based_selection*) descrito en [12] (Anexo C), el cual, además de presentar una estrategia de selección, contempla el proceso de reconfiguración a partir de un modelo de composición basado en FSM. Para realizar la adaptación del algoritmo, se consideraron algunas diferencias, las cuales se mencionan en la tabla 1.

Composición dinámica en WS	Composición dinámica en el SLEE	Aporte
Contempla el proceso de selección dentro de la composición dinámica de servicios	El proceso de selección se contempla dentro de la fase de descubrimiento	Evita retardos que involucran las técnicas de selección de servicios
La composición, es realizada a nivel de Servicios Web atómicos, a partir de un WS compuesto (WS de consulta)	La composición se realiza a nivel de componentes SBB	Se aprovechan los beneficios de composición provistos por JAIN SLEE.
		Permite contemplar un modelo de composición atómica
El modelo de composición dinámica contempla todo el flujo de ejecución del WS compuesto	El modelo de composición dinámica es atómico, es decir contempla el flujo de ejecución comprendido por el SBB que falló, el anterior y el siguiente	Evita retardos que se podrían presentar por la construcción del modelo total del servicio, en tiempo de ejecución
No se contemplan estados de sincronización	Se contemplan estados de sincronización	Permiten controlar en el flujo de composición, si se realizó con éxito la reconfiguración del servicio

Tabla 1. Diferencias entre el algoritmo de composición en WS y en el SLEE

A continuación se describe la lógica del algoritmo de composición dinámica presentado en la figura 8 para lograr la reconfiguración de un componente SBB del servicio original por medio de la sustitución de un candidato que realice el procesamiento del evento que falló.

Descripción del algoritmo

Como se ha mencionado anteriormente para realizar la reconfiguración del componente del servicio se debe entregar como entradas del algoritmo: el evento no procesado, el componente SBB que fallo en el procesamiento de dicho evento, los SBB candidatos adecuados para reemplazar al componente que fallo y parte del flujo del servicio necesario para la reconfiguración. A partir de estas entradas realiza una lógica que permite que el evento sea procesado por un componente candidato y en caso de generar un evento que siga en la secuencia del servicio es entregado al SBB correspondiente.

El algoritmo basa su lógica en referencia a la definición de la FSM del modelo de composición dinámica establecido en la sección 3.2.2.3, en el cual se presenta como una adaptación el manejo de estados de sincronismo especiales para realizar la reconfiguración, a continuación se describe la estructura del algoritmo:

- **Entradas:**
  - Evento que no fue procesado con éxito
  - Componente SBB que fallo en el procesamiento del evento
  - Identificación del servicio afectado

- **Salida:**
  - Notificación de que el componente fue reconfigurado con éxito o no

- **Definición de variables:**
  - Estado de sincronización sin suerte (unluck): define cuándo un candidato logra el procesamiento del evento con éxito o no. su valor inicial es sin éxito.
  - El contador de ejecución de candidatos (contCandidate): define el número de candidatos ejecutados, su valor inicial es cero.
  - El conjunto de componentes candidatos que pueden reemplazar al SBB respectivo.
  - El flujo de reconfiguración: parte de flujo del servicio necesario para realizar la adaptación del componente candidato a la lógica del servicio.

- **Definición de métodos:**
  - Método de búsqueda del flujo de reconfiguración, se encarga de revisar en la FSM del flujo del servicio afectado los datos necesarios para realizar la reconfiguración (SBB anterior, SBB que falla, SBB siguiente y eventos implicados) a partir de la comparación de la información de entrada: evento no procesado y el SBB que fallo.
  - Método de solicitud de candidatos, se encarga de obtener el arreglo de candidatos para el componente SBB que fallo, los candidatos ya están previamente clasificados desde la fase de descubrimiento, por tal motivo a partir de la identificación del SBB que falla se escoge el conjunto correspondiente.

A partir del valor del estado unluck y el número de candidatos ejecutado se decide realizar la lógica definida para la reconfiguración de la siguiente manera:

- Si **unluck** es igual a ***sin éxito*** *y* ***contCandidate*** *es* ***menor*** *al número total de candidatos.*

  Se inicia el estado de reconfiguración del componente a partir de la ejecución de cada candidato en el orden entregado para reemplazar el componente que fallo, por lo cual se dispara el evento fallido hacia el candidato y se determina si es procesado con éxito por medio de un evento de respuesta, luego se determina por medio del flujo de reconfiguración de que exista evento resultante, en caso de tenerlo se obtiene el evento resultante del candidato y se direcciona hacia el siguiente SBB encargado de procesarlo; en caso de no tener evento resultante se notifica que la reconfiguración termino. Al final se entrega la notificación de éxito como salida, pero en caso de fallar el candidato el valor de Unluck seguirá estando en ***sin éxito***, con lo cual se repite de nuevo la lógica de reconfiguración.

- Si **unluck** es igual a ***sin éxito*** *y* ***contCandidate*** *es* ***igual*** *al número total de candidatos.*

  Se pasa al estado de falla definitiva, ya que todos los candidatos fallaron y se notifica la reconfiguración no fue exitosa.

A continuación se muestra la representación formal del algoritmo de composición dinámica:

Algoritmo de Composición Dinámica
Nombre: *Reconfiguration*
Descripción: reconfigura un componente de un servicio cuando ocurre una falla en el procesamiento de un evento.

INPUTS: $(inputEventFail, SBBCompFail, IdService)$
OUTPUT: $isSBBReconfig$
Begin

1. $unlukState = false$
2. $contCandidate = 0$
3. $FlowReconfig = searchSLEEService(InputEventFail, SBBCompFail, IdService)$
4. $CandidateSBB = getCandidateSBB(SBBCompFail)$

// State reconfiguration

5. if (! $unlukState$ && $contCandidate < CandidateSBB.lengt$) then
6. SBB=$CandidateSBB.next()$
7. **fireEvent** ($inputEventFail$, SBB)
8. $contCandidate++$
9. **wait** ($responseEvent$)
10. **if** ($responseEvent.stateProcess$) then
11. **if** ($FlowReconfig.eventResult != null$) then
12. $outputEvent = SBB.getresultEvent()$
13. $SBBNext = FlowReconfig.getSBBNext()$
14. **fireEvent** ($outputEvent, SBBNext$)
15. **end**
16. $isSBBReconfig$:"$Sucsess\ Reconfiguration$"
17. **return** $isSBBReconfig$
18. **else**

// State unluck

19. break 3
20. end
21. **else**

// State failure

22. $isSBBReconfig$:"$Failure\ Reconfiguration$"
23. **return** $isSBBReconfig$
24. **end**

Figura 9. Algoritmo de composición dinámica en el SLEE

Capítulo 4
Evaluación y análisis de resultados

4.1 Introducción

En este capítulo se presenta detalladamente la implementación y evaluación del algoritmo de composición dinámica de servicios SIP en el SLEE; como ya se ha mencionado, este trabajo de grado se enfoca en una de las fases definidas para la composición dinámica, esta es la fase de reconfiguración de un componente candidato al flujo de ejecución del servicio (definida en la sección 3.2.1.4), esta decisión fue tomada debido a que se considera la fase de reconfiguración como la más crítica del proceso de composición dinámica porque adecúa en tiempo real el flujo de ejecución del servicio para que continúe el funcionamiento normal de éste, cuando ocurra una falla de uno de sus componentes.

4.2 Caso de estudio

En este trabajo de grado se contempla la implementación de un módulo que permite realizar composición dinámica de servicios SIP en el SLEE, dicho módulo está basado en la definición formal de $\boldsymbol{Composition_{SLEE}}$ realizada en la sección 3.2.2.3 y el algoritmo adaptado de [12], presentado en la sección 3.2.2.6. Con el fin de llevar a cabo el proceso de evaluación del algoritmo, se considera necesario el desarrollo de un servicio SIP compuesto de varios SBB, los cuales son propensos a fallos en tiempo de ejecución.

Con el fin de llevar a cabo el proceso de composición dinámica enfocándose en la fase de reconfiguración, se han considerado algunos tópicos, mencionados a continuación:

- El algoritmo adaptado de [12], tiene algunas modificaciones con el fin de adecuarse al entorno SLEE, además de la composición a nivel de componentes considerada, también se han agregado unos estados especiales de sincronización que ayudan a definir el comportamiento de la FSM de composición dinámica en el SLEE.

- El algoritmo de composición dinámica necesita ser ejecutado por un módulo que contenga los candidatos para poder controlarlos, además, el servicio a componer, necesita de un elemento adicional a su lógica que intervenga en el control del evento resultante de la reconfiguración, para poder enviar dicho evento al componente correspondiente, según el flujo de ejecución del servicio.

- Los candidatos para todos los componentes SBB del servicio a componer, deben haber pasado por un proceso de descubrimiento y selección, en este trabajo de grado, dichas fases se asume que ya han sido realizadas de tal forma que los candidatos para cada SBB del servicio se consideran aptos para llevar a cabo la reconfiguración y por lo tanto se encuentran dentro del módulo de composición dinámica.

El módulo de composición dinámica provisto en este trabajo de grado considera servicios que utilicen los elementos ofrecidos por el adaptador de recursos SIP para llevar a cabo el proceso de reconfiguración, sin embargo, el SLEE define un nuevo paradigma para llevar a cabo la lógica de sus servicios, la cual básicamente funciona de la misma manera para todos ellos (eventos procesados por componentes con funcionalidades específicas) y además adapta los recursos externos a él (mediante adaptadores de recursos); gracias a esto, el módulo de composición dinámica es fácilmente adaptable a cualquier tipo de servicio dentro del SLEE, simplemente definiendo en el componente indicado los tipos de evento que puede procesar.

4.3 Arquitectura de referencia de composición dinámica

En la figura 10 se muestra la arquitectura general del proceso de composición dinámica de servicios SIP, en ella se observa que contenidos en el SLEE se encuentran el módulo de composición y los servicios SIP (utilizan las primitivas provistas por el AR SIP) que pueden comunicarse por medio del enrutador de eventos. Gracias a la comunicación asíncrona, Asumiendo la fase de monitoreo las fallas pueden ser detectadas y notificadas al módulo por medio de un evento de alarma cuando se presente en algún servicio un problema en uno de sus componentes para procesar un evento en particular, para efecto de pruebas, las fallas han sido simuladas en el servicio a partir del envió del evento de alarma directamente desde el componente SBB afectado, de esta manera, si existe un candidato que pueda dar una respuesta exitosa, el módulo de composición, envía un evento de respuesta al servicio SIP para que continúe el flujo de ejecución normal.

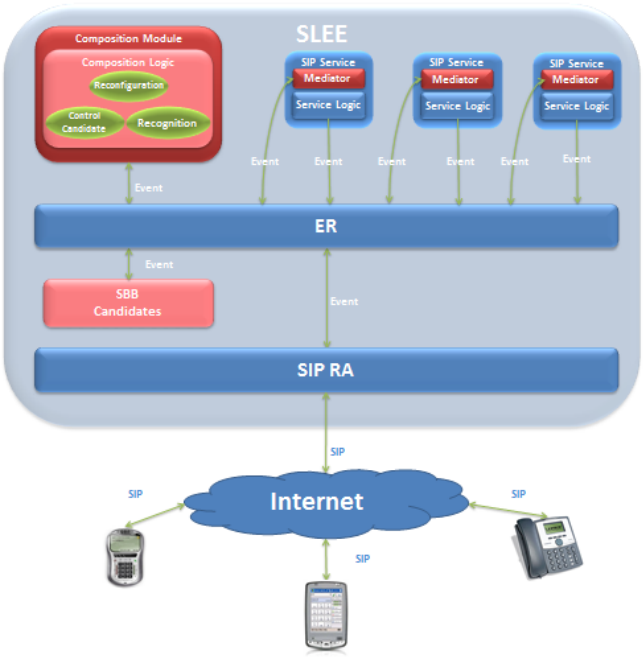

Figura 10. Arquitectura de referencia de composición dinámica

Con el fin de lograr el proceso de composición dinámica en este trabajo de grado se proveen dos componentes, el módulo de composición y el mediador, los cuales se pueden ver de un color rojo en la arquitectura. El módulo de composición, es el componente encargado de llevar a cabo la reconfiguración del servicio, éste, recibe una notificación de la falla y realiza el procedimiento para obtener la respuesta adecuada; el mediador, es el componente encargado de dirigir un evento resultante del proceso de reconfiguración al componente indicado del servicio. En la arquitectura, también se observa el módulo SBB Candidates, este módulo es el que contiene todos los candidatos disponibles para llevar a cabo la reconfiguración.

En la arquitectura se pueden observar, además de los componentes provistos y el módulo SBB Candidates, otros componentes que son necesarios para realizar el proceso de composición dinámica, estos son, el ER, eventos externos al SLEE y el SIP RA. El enrutador de eventos, es el componente que permite realizar la comunicación (asíncrona) entre el módulo de composición y los servicios; los eventos externos, son específicamente peticiones SIP que necesitan ser procesadas en los servicios del SLEE y el SIP RA, es el componente que permite adaptar las peticiones SIP externas, a la lógica de enrutamiento de eventos del entorno.

4.4 Implementación de referencia

La implementación de la arquitectura se realizó utilizando el servidor Mobicents JSLEE 1.1 v2.3.0, el cual se ejecuta en el ambiente de ejecución Java (JRE – Java Runtime Environment) 1.6 sobre el sistema operativo Linux Ubuntu 9.04, como se puede observar en la figura 11. Para el desarrollo del software necesario, tal como el mediador (MedatorSbb.jar) y el módulo de reconfiguración (ReconfigSbbSLEE-DU.jar) se utilizó la herramienta Eclipse Helios v3.6.1 a la que se le adicionó el Plug-in EclipSLEE v1.2.6.

En la figura 11 también se observa el AR SIP 11 utilizado para procesar las peticiones SIP lanzadas por el softphone X Lite 4.0 y adecuarlas a la lógica orientada a eventos de JAIN SLEE

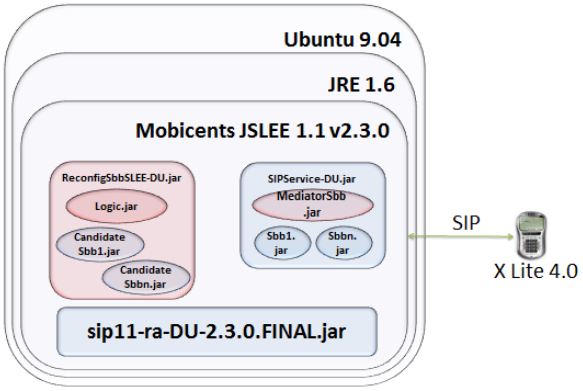

Figura 11. Modelo de despliegue

El componente Logic.jar, representa a toda la lógica implementada en el módulo que permite llevar a cabo el algoritmo de composición dinámica, es decir a los SBB Recognition, Control Candidate y Reconfiguration.

4.4.1 Módulo de composición

El módulo de composición (Composition Module), es el componente que contiene toda la lógica que propone el algoritmo de composición dinámica (detallado en la sección 3.2.2.6), como se muestra en la figura 10, este módulo puede componer dinámicamente varios servicios, haciendo uso del ER para el envío y recepción de eventos (comunicación asíncrona).

La lógica del módulo de composición inicia cuando se recibe un evento *AlarmEvent* que tiene información de la falla ocurrida en uno de los servicios SIP. El módulo de composición extrae el evento que falló en el servicio y lo envía a uno de los candidatos, si este no lo procesa exitosamente, el evento se transfiere a otro SBB candidato. Este proceso se repite hasta que uno de los candidatos procese exitosamente el evento, o bien hasta que se determine que ninguno de ellos puede procesarlo como se estableció en la sección 3.2.2.3.

4.4.1.1 Lógica de composición

El sub-módulo de lógica de composición permite realizar la lógica del algoritmo de composición para llevar a cabo los mecanismos de reconfiguración que se contemplan dentro de la composición dinámica propuesta en este trabajo de grado, la cual consta del reconocimiento de la notificación de falla, el proceso de control de candidatos, el procesamiento del evento que fallo y adaptación del flujo del servicio implicado. La lógica está repartida en tres componentes (SBB), los cuales se describen en detallé a continuación:

- *Reconfiguration*

  Este SBB es el componente principal del módulo, se ha definido como el SBB raíz, ya que es el encargado de establecer la comunicación con el mediador del servicio y los candidatos para realizar las funciones más importantes de la reconfiguración, que a continuación son enunciadas:

- Recibir la notificación de falla del servicio afectado, el cual está definido en un evento personalizado llamado *AlarmEvent* (evento construido de forma particular dentro del SLEE, que contiene la información del evento y SBB que fallaron, el evento a procesar en el servicio).
- Recibir la información necesaria del flujo del servicio afectado para realizar la reconfiguración y los candidatos del SBB a sustituir, esta información es entregada por medio del evento *CandidateEvent* (evento personalizado que contiene el flujo necesario para la reconfiguración y un arreglo de los Componentes SBB candidatos).
- Disparar el evento no procesado del servicio afectado, hacia los candidatos en el orden que ha sido preestablecido en una fase previa (Descubrimiento), luego verificar si el candidato correspondiente lo procesó o no, con éxito, por medio del evento personalizado *ProcessEvent* (evento que notifica, si el procesamiento del evento, se realizó con éxito en el SBB candidato, y en caso de generar evento resultante lo guarda como un atributo).
- En caso de recibir un evento resultante por parte del candidato, se encarga de establecer la comunicación con el componente mediador para entregar dicho evento al SBB siguiente en el flujo del servicio. El SBB *Reconfiguration* se encarga de disparar un evento *ReconfigEvent* (evento personalizado que contiene el evento resultante del procesamiento del candidato y el id del SBB que debe procesarlo) al mediador para adaptar el flujo del servicio. El mediador envía un evento *InfoEvent* (evento personalizado que notifica si el disparo del evento resultante al SBB correspondiente), en respuesta al disparo del evento resultante al SBB correspondiente del servicio afectado.
- Encargado de notificar en consola si el proceso de reconfiguración fue realizado con éxito o no.

Nota: Este componente debe tener definidos todos los eventos que se van a enviar como respuesta al servicio SIP a componer, con el fin de poder utilizar su disparador.

- *Recognition*

Este componente es definido como un hijo del SBB Reconfiguration encargado de realizar la siguiente función:

- Reconocimiento de la falla ocurrida del servicio afectado a partir del conocimiento de la identificación del servicio, el componente SBB y evento respectivo donde se presentó la falla, con esta información, se consulta en una tabla de perfil el flujo de ejecución del servicio y se determina el flujo necesario para realizar la reconfiguración, el cual contiene el SBB que falla en el servicio, el SBB anterior y el SBB siguiente y los eventos implicados.

Se aclara que el flujo de ejecución establecido en el perfil se creó manualmente puesto que los objetivos de este trabajo de grado no plantean crearlo en tiempo de ejecución. Dicho flujo es determinado a partir del modelo de comportamiento de un servicio en el SLEE representado por la FSM definida en la sección 3.2.2.1, el cual es necesario para realizar composición dinámica en el SLEE.

- *Control Candidate*

Este componente es definido como un hijo del SBB Recognition encargado de realizar la siguiente función:

- Se encarga de buscar el conjunto de candidatos aptos para ejecutar el evento que falló del SBB original del servicio afectado a partir de la información entregada por medio del evento FlowEvent (evento personalizado que contiene el flujo necesario del servicio para la reconfiguración: SBB que falla, SBB anterior, SBB siguiente y los eventos relacionados).

Este componente facilita la búsqueda de los SBB candidatos puesto que cada uno de ellos es un SBB hijo del SBB *Control Candidate*, esto permite establecer arreglos de SBB candidatos que sean aptos para hacer el procesamiento que realiza un SBB del servicio original, de esta manera, se puede escoger el

arreglo que contenga los SBB apropiados. En general este proceso podría consumir un tiempo considerable puesto que la cantidad de candidatos disponibles puede ser muy grande, para evitar este problema, se determina que tanto los candidatos como los SBB del servicio original tengan un id que permita identificarlos rápidamente.

Para ejecutar el proceso de búsqueda, en este trabajo se asume que los candidatos han tenido un proceso de selección en la fase de descubrimiento, donde se determina que tan apto es un candidato para reemplazar un componente del servicio original, a través de procesos sintácticos o semánticos, o con técnicas de QoS como se realiza en [12] con el concepto de confiabilidad agregada. De esta forma se evita realizar un proceso de selección dentro del módulo de composición que retarde la reconfiguración del servicio. Este proceso de selección, permite obtener una lista de candidatos definida previamente, la cual permite identificar los componentes que pueden procesar un evento de un servicio y por lo tanto, clasificarlos de tal manera que en el proceso de reconfiguración se utilicen aquellos componentes que específicamente procesen el evento que requiera el proceso de reconfiguración.

Partiendo del conocimiento de los componentes SBB que realizan la lógica de composición, a continuación se explica cómo se forma el flujo de ejecución:

- ✓ Se recibe el evento *AlarmEvent en el SBB Reconfiguration*, enviado desde el servicio que ha experimentado una falla en tiempo de ejecución.
- ✓ El evento *AlarmEvent* es transferido al componente SBB *Recognition* encargado de buscar el flujo del servicio requerido para realizar la reconfiguración. Como resultado de esta búsqueda se envía la información en el evento *FlowEvent* dirigido al SBB *ControlCandidate*.
- ✓ El SBB *ControlCandidate,* a partir de la información del evento FlowEvent, se encarga de seleccionar el conjunto de SBB candidatos correspondientes para procesar el evento que fallo, y se envía el evento *CandidateEvent* hacia el SBB *Reconfiguration* (comunicación síncrona) con la información de los candidatos disponibles para el SBB particular a reemplazar en el procesamiento del evento del servicio original e información del flujo necesario, específicamente, el SBB que falla, el SBB anterior, el SBB siguiente y los eventos implicados.
- ✓ El SBB *Reconfiguration* con la información de los candidatos, el evento no procesado del servicio afectado y el flujo requerido para la reconfiguración, procede a realizar la ejecución de los candidatos, iniciando con el envío del evento que falló a uno de los candidatos, luego para verificar si el evento fue procesado con éxito o no, el candidato envía un evento *ProcessEvent* a *Reconfiguration* para confirmar el estado del procesamiento del evento, y en caso de fallar este procede a reenviar el evento que falló a otro de los candidatos, repitiéndose el proceso hasta que se determine que alguno de los candidatos ejecutó exitosamente el evento o que ninguno de ellos pudo hacerlo.
- ✓ Cuando un candidato procese exitosamente el evento de fallo, si es necesario, el SBB *Reconfiguration* envía el evento personalizado *ReconfigEvent* al servicio SIP que experimentó la falla (específicamente al mediador del servicio) con el resultado del procesamiento (el evento resultante y el SBB al que va dirigido) y luego que el mediador realice el disparo del evento al SBB correspondiente, se envía confirmación por medio del evento *InfoEvent* a *Reconfiguration*. Si no es necesario, simplemente se despliega un mensaje en consola indicando que el evento fue procesado exitosamente.
- ✓ Si ninguno de los candidatos pudo procesar exitosamente el evento, se ejecuta en consola un mensaje que notifica este suceso, y se determina que el servicio no puede ser reconfigurado puesto que en la fase de descubrimiento no se seleccionó un candidato pertinente para realizar el proceso.

Nota: las funciones del algoritmo fueron divididas entre los componentes del módulo de composición debido al manejo de eventos necesarios y con el fin de dar escalabilidad al sistema.

4.4.2 SBB candidatos

Otro módulo necesario para la composición es el SBB *Candidates*, el cual contiene todos los SBB candidatos. Cada uno de los SBB candidatos es hijo del SBB *Control Candidate*, facilitando de esta manera la identificación de los SBB candidatos que pueden procesar un evento que proviene de un SBB particular de un servicio, el cual ha sufrido una falla.

Como ya se mencionó, el módulo de composición dinámica puede hacerse cargo de la composición de varios servicios que estén en ejecución en el SLEE. Para esto, el evento personalizado *AlarmEvent* enviado desde el servicio hasta el módulo de composición, tiene información del servicio, evento y SBB implicado, con esta información se consulta en una tabla de perfil el flujo de ejecución (FSM) del servicio particular identificando los elementos necesarios del SBB que falló (eventos de entrada y salida) y se limita el proceso de reconfiguración a los SBB candidatos que soporten dichos eventos.

Una vez determinados los candidatos que pueden procesar un evento particular, se lanza al primero de ellos el evento que necesita ser procesado, si el procesamiento del evento no se realiza con éxito se procede a lanzar el evento al siguiente SBB candidato, repitiéndose el procedimiento hasta que se determine que el procedimiento se realizó exitosamente o que ninguno de los candidatos pudo realizarlo, cada candidato en su lógica tiene establecido el envió del evento *ProcessEvent* para notificar el estado del procesamiento del evento.

4.4.3 Mediador

Con el fin de tener un control de todos los eventos que llegan al servicio SIP, se ha construido un módulo mediador que contiene básicamente un SBB que permite realizar el control de eventos. Como ya se ha mencionado, cada servicio del SLEE contiene un SBB raíz que da inicio a toda la lógica del servicio, y en general, todos los eventos que lleguen al servicio se reciben únicamente en este SBB, de esta manera, es necesario para la composición dinámica del servicio que el SBB raíz controle cada uno de los eventos de entrada, permitiendo identificar si es o no un evento de respuesta a una falla ocurrida en el servicio, razón por la cual se provee un mediador adaptado a la lógica del servicio.

Para facilitar el proceso de control de eventos, el SBB raíz que contiene el mediador, tiene como hijos a todos los SBB que componen la lógica del servicio, de esta manera, cuando llegue un evento respuesta a una falla (evento personalizado que trae información necesaria para identificar el SBB siguiente en el flujo de ejecución), se identifica el SBB al que va dirigido el evento y se lo envía haciendo uso de la relación hija respectiva.

- **Control de eventos**

  Con el fin de controlar los eventos dirigidos al servicio SIP se ha construido un SBB raíz llamado *Mediator*, este SBB recibe todos los eventos dirigidos al servicio y determina si es el evento personalizado *ReconfigEvent* o un evento perteneciente a la lógica del servicio. Si el evento va dirigido a la lógica de composición, el *Mediator* lo deja pasar sin realizar ningún procesamiento, si el evento es el *ReconfigEvent*, el cual lleva información del evento resultante del proceso de reconfiguración y el SBB al que va dirigido, el *Mediator* envía el evento resultante al componente correspondiente y envía un evento personalizado de confirmación llamado *InfoEvent* hacia el módulo de composición, este evento se encarga de notificar que el evento resultante fue disparado para seguir con el flujo normal del servicio, hasta allí se da por terminada la reconfiguración del componente. La lógica de composición realizada no garantiza que después de la reconfiguración no vuelvan a ocurrir fallas, en este caso el proceso de composición se repetiría hasta solucionar el problema.

En la figura 10, se puede apreciar que además de los componentes provistos por este trabajo de grado, existen otros elementos que intervienen en el proceso de composición, estos son el ER, el SIP RA y los eventos externos al SLEE, a continuación se describen cada uno de ellos.

4.4.4 Enrutador de Eventos

Como ya se ha mencionado, el enrutador de eventos permite realizar una comunicación asíncrona dentro del SLEE, en este trabajo de grado se aprovecha dicha característica para el envío del evento personalizado que indica al módulo de composición que ha ocurrido una falla en un servicio particular, cuando el módulo de composición ya haya realizado el proceso de análisis respectivo, y si es necesario, se envía el evento personalizado de respuesta a la falla al servicio, haciendo uso nuevamente del enrutador de eventos. Este elemento también gestiona todos los eventos SIP externos al SLEE, reenviándolos al servicio que los soporte

4.4.5 Eventos externos al SLEE

Como ya se ha definido, en este trabajo de grado se hace uso del protocolo SIP, de esta manera, se utilizan softphone que permiten generar peticiones SIP que puedan procesar los servicios contenidos en el SLEE y que puedan ser reconfigurados, el softphone utilizado es el X-Lite 4 provisto por la corporación Counterpath, que permite una descarga gratuita del software con las funcionalidades básicas para realizar una llamada de voz, video y también permite realizar mensajería instantánea.

4.4.6 Adaptador de recursos SIP

Este elemento, como se muestra en la arquitectura (Figura 11) adapta todas las peticiones SIP realizadas por recursos externos al SLEE, a la lógica orientada a eventos del entorno, permitiendo realizar composición dinámica de servicios SIP en el SLEE.

4.4.7 Servicio de prueba (CallMsgService)

Con el fin de evaluar el algoritmo de composición dinámica se desarrolló el servicio *CallMsgService* haciendo uso de las funcionalidades provistas por el AR SIP, el servicio consta básicamente de dos módulos, como se observa en la figura 12, el módulo que contiene todos los SBB que permiten llevar a cabo la lógica del servicio (Service Logic) y el mediador adaptado al flujo de ejecución para el control de sus eventos.

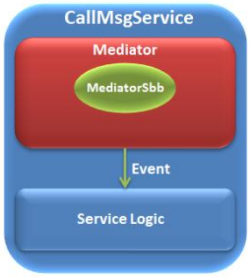

Figura 12. Servicio de prueba

Como se puede observar en la figura 12, el mediador del servicio *CallMsgService* posee un SBB MediatorSbb, que es el SBB raíz del servicio y padre de todos los SBB que componen la lógica del servicio, éste ayuda a controlar los eventos de entrada al servicio como se mencionó en la sección 4.4.3.

4.4.7.1 Lógica del servicio

En esta sección se detalla la FSM del servicio SIP CallMsgService, su lógica (contenida en el sub-módulo Service Logic) consta 5 SBB que permiten simular una falla en cualquiera de sus componentes.

El servicio CallMsgService tiene 5 funcionalidades bien definidas: la primera de ellas es permitir el registro de usuarios, una vez registrados, pueden utilizar dos funcionalidades más, llamar o enviar mensajes instantáneos entre cualquiera de los usuarios registrados. Si se ejecuta una llamada se utiliza una

funcionalidad más que permite llevar la cuenta del tiempo de la llamada y si se inicia una conversación mediante mensajes instantáneos existe una funcionalidad que permite contar los mensajes que ha enviado cada usuario. Para efecto de pruebas, cada funcionalidad ha sido definida en un SBB diferente, permitiendo así la simulación de una falla en cualquiera de ellos.

Figura 13. FSM del servicio *CallMsgService*

En la figura 13 se observa que la FSM del servicio tiene dos posibles flujos de ejecución, uno definido por la realización de la llamada y el otro por el envió de mensajes instantáneos, consecuentemente se concluye en dos estados finales representados por los componentes donde termina el flujo. También se aprecia los SBB que constituyen el servicio de prueba CallMsgService, a continuación se detalla cada uno de ellos.

- *Register*

  Este componente se encarga del registro de los usuarios y gestión de las siguientes dos funcionalidades del servicio (mensajería instantánea o llamada de voz). Es decir, permite re-direccionar el evento SIP de mensajería (*Message*) o los eventos SIP implicados con una llamada (*Invite o Bye*) al componente que permita realizar el respectivo proceso.

- *Message*

  Este componente se encarga de recibir un evento SIP *Message* y realizar el proceso que permite obtener la información para re-enviar el contenido del mensaje al respectivo host de destino lanzando un evento SIP *Message*. Una vez enviado el mensaje, envía un evento personalizado (*MessageCounterEvent*) al siguiente componente para realizar el conteo de mensajes.

- *B2BUA*

  Este componente permite gestionar el proceso de una llamada telefónica entre dos usuarios registrados, iniciándola mediante el evento SIP de entrada *Invite* o finalizándola con el evento SIP de entrada *Bye*. Cada evento SIP de entrada es re-direccionado al host de destino correspondiente permitiendo iniciar o finalizar respectivamente un diálogo SIP. Una vez finalizada una llamada, el componente envía un evento personalizado *CallBillingEvent* con la información del tiempo utilizado en la llamada y del usuario que la inició.

- *CallBilling*

  Permite realizar la tarificación a cada usuario que ejecuta una llamada almacenando su tiempo de duración.

- *MessageBilling*

  Realiza la tarificación de la mensajería instantánea por medio del conteo de mensajes enviados por cada uno de los usuarios. La información del usuario que envía un evento llega al componente *MessageBilling* por medio del evento personalizado *MessageCounterEvent*.

4.4.8 Ejemplo de Reconfiguración de servicios en el SLEE

Para llevar a cabo un ejemplo de reconfiguración se describirá primero el comportamiento que se quiere lograr por medio de este proceso basado en el concepto de composición dinámica para los servicios dentro de un entorno de ejecución de lógica de servicio y se describirá en detalle la lógica de los mecanismos de composición dinámica que se desarrollaron para lograr el objetivo.

Los servicios en el SLEE tienen un flujo normal de ejecución de sus componentes para realizar sus funciones determinadas, debido a cambios inesperados en el momento de la ejecución es posible que ocurran fallas que alteren este flujo y no permitan llevar a cabo sus objetivos funcionales, por tal motivo una solución propuesta es la reconfiguración que permita corregir el problema y continuar con el flujo normal del servicio. En la figura 14 se puede apreciar la FSM que representa el comportamiento deseado de los servicios en este entorno utilizando el concepto de composición dinámica para adaptarlos a cambios en tiempo de ejecución.

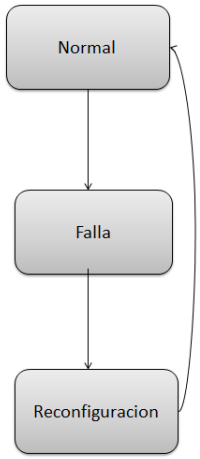

Figura 14. FSM del comportamiento deseado del servicio en el SLEE

En la figura 15 se observa un ejemplo de reconfiguración del servicio de prueba para solucionar una falla ocurrida en la funcionalidad de tarificación de llamada, la cual se ejecuta al terminar la llamada por medio de un *Bye* y procede a almacenar el tiempo de duración de esta en el perfil del usuario que la realizo, este caso se modela un procesamiento incorrecto del evento *Bye* perteneciente a la lógica del componente *B2BUA y* generando como consecuencia que el evento *CallBillingEvent* no sea disparado hacia el SBB *CallBilling*. También en la figura 15, se aprecia (lado izquierdo) el orden de ejecución de los componentes del servicio de prueba hasta el punto que ocurre un fallo y desde allí inicia la lógica de composición dinámica establecida en el módulo *ReconfigSbbSLEE* (lado derecho) por medio de la ejecución de sus componentes a partir del disparo de alarma *AlarmEvent* y concluyendo con el evento de confirmación *InfoEvent* del componente mediador. A continuación se describe el flujo de composición dinámica realizado para la reconfiguración del componente SBB *B2BUA* en el orden de enumeración del disparo de eventos:

1. El mediador se encarga de transferir el evento Bye al SBB raíz del servicio: el SBB Register.

2. El SBB Register se encarga de direccionar el evento Bye hacia el componente encargado de procesarlo, en este caso el SBB B2BUA, donde ocurre la falla al tratar de procesarlo.
3. Al ocurrir la falla, un evento AlarmEvent es disparado al módulo de composición para notificar el problema que se ha presentado, y trae consigo el evento que no fue procesado con éxito.
4. El SBB raíz del módulo: SBB Reconfiguration lo transfiere hacia un componente llamado SBB Reconigtion, encargado de reconocer la falla, y de buscar en la FSM del flujo de servicio que ha sido establecida en perfiles y obtener el flujo necesario para llevar acabo la reconfiguración: el SBB anterior, el SBB siguiente en el flujo y los eventos implicados entre ellos.
5. El SBB Reconigtion dispara un evento FlowEvent, el cual lleva la información del flujo requerido para la reconfiguración hacia el SBB ControlCandidate.
6. El SBB ControlCandidate después de recibir el evento FlowEvent, se encarga de seleccionar el conjunto de candidatos correspondientes y garantizar que estén activos, luego dispara un evento CandidateEvent hacia el SBB Reconfiguration, este evento tiene como atributos el conjunto de candidatos y el evento FlowEvent.
7. El SBB Reconfiguration se encarga de ejecutar el algoritmo de composición, por lo cual recupera el evento no procesado (Bye) y lo dispara hacia a un candidato SBB CandidateB2BUA para que sea procesado con éxito.
8. El SBB CandidateB2BUA proceso con éxito el evento y ahora dispara un evento personalizado ProcessEvent, encargado de notificar el estado de proceso del evento y en este caso de entregar el evento resultante CallBillingEvent al SBB Reconfiguration.
9. El SBB Reconfiguration detecta que el procesamiento del Bye fue llevado con éxito y dispara el evento ReconfigEvent con el evento resultante hacia el SBB mediator para que termine la reconfiguración del servicio.
10. El SBB Mediator se encarga de direccionar el evento resultante CallBillingEvent hacia el SBB correspondiente en el flujo de reconfiguración, en este caso el SBB CallBilling.
11. El SBB Mediator termina la reconfiguración del servicio y envía un evento de notificación InfoEvent al módulo de composición.

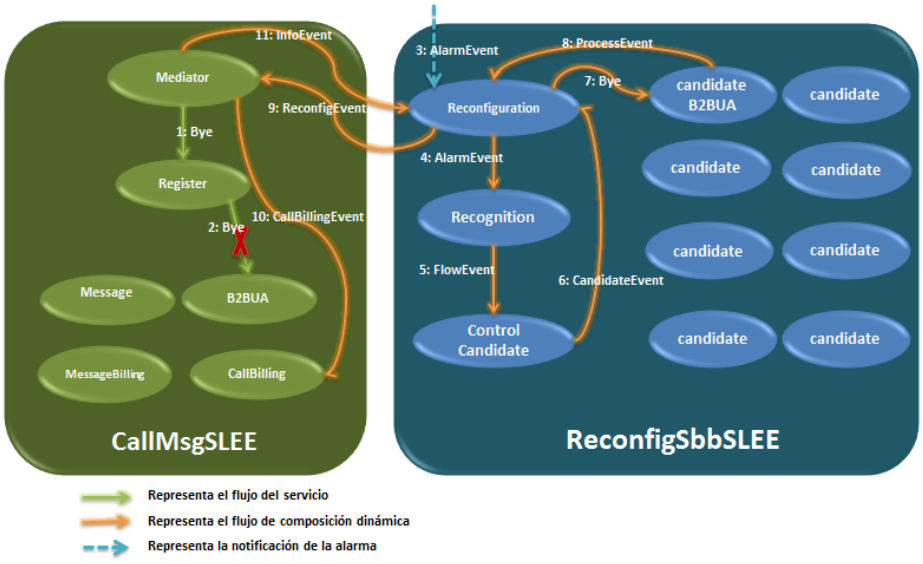

Figura 15. Lógica de composición dinámica

4.5 Evaluación del algoritmo

En esta sección se presenta la metodología de experimentación utilizada, las medidas utilizadas como criterios de evaluación, las pruebas de desempeño realizadas, análisis y resultados logrados para establecer el rendimiento del algoritmo de composición dinámica en el SLEE.

4.5.1 Metodología de experimentación

La evaluación del algoritmo por medio del módulo de composición se realizó haciendo uso del método experimental logrando así, medir su desempeño por medio del diseño de un plan de pruebas basado en la medida de tiempos de respuesta en un escenario experimental, los cuales son expresados en tablas y gráficas para su fácil análisis y compresión de los resultados obtenidos utilizando un criterio de evaluación que permite comparar con valores de referencia y generar las conclusiones acertadas del comportamiento observado.

Para medir el rendimiento del algoritmo se procedió a tomar los tiempos críticos del proceso de composición dinámica de la siguiente manera:

1. Se establecieron puntos de control en el módulo de composición, el mediador y en el servicio CallMsgService, los cuales permiten tomar los tiempos necesarios para realizar la evaluación del algoritmo de composición dinámica.
2. Se midió el rendimiento del algoritmo de composición dinámica comparando los tiempos que tardan las funcionalidades del servicio *CallMsgService* cuando se ejecutan normalmente, y cuando ocurren fallas y el módulo de composición reconfigura el servicio, completando exitosamente las funcionalidades del servicio.
3. Se midió el rendimiento del algoritmo de composición dinámica, variando el número de componentes SBB candidatos necesarios para procesar con éxito un evento que falla en una reconfiguración del servicio de prueba, es decir, se varía el número de candidatos que fracasan en el intento de procesar el evento que fallo.
4. Se midió el rendimiento del algoritmo de composición dinámica, variando el número de servicios para reconfigurar en forma simultánea utilizando un candidato para procesar con éxito el evento que fallo.

4.5.2 Banco de pruebas

Para realizar las pruebas, se contó con las cuatro funcionalidades del servicio *CallMsgService*, la funcionalidad de registro, la de mensajería instantánea, la de llamada de voz y la funcionalidad de tarificación de llamada para las cuales se mide el tiempo de respuesta del flujo de ejecución normal y el tiempo de respuesta del flujo de ejecución con reconfiguración simulando las fallas en el procesamiento de los eventos Register, Message, Invite, Bye por parte de los SBB Register, Message y B2BUA respectivamente.

El conjunto de SBB candidatos para realizar la reconfiguración son implementados como versiones similares a los SBB del servicio original con diferentes identificadores para cada uno de ellos, por ejemplo para el SBB Register del servicio de prueba se desarrollaron 20 candidatos con la notación siguiente: CandidateOneRegister, CandidateTwoRegister, etc. para realizar la prueba de variación de candidatos para una reconfiguración del servicio en el procesamiento el evento Register. De igual manera fueron desarrollados un número finito de candidatos para los demás componentes.

Para la prueba que exige la variación del número de servicios del SLEE, con el fin de medir los tiempos que tomó el módulo en reconfigurar varios servicios en forma simultánea, se desarrollaron versiones similares del servicio de prueba *CallMsgService* hasta un número total de 10 servicios.

Para realizar el proceso de reconfiguración, el servicio *CallMsgService* se despliega con algunos de los componentes SBB del servicio, en mal estado, de esta manera, cuando ocurra la falla, el componente

implicado lanza un evento personalizado llamado *AlarmEvent*, dicho evento es transferido al módulo de composición a través del enrutador de eventos, donde se puede obtener la información del servicio, componente SBB y evento de entrada implicado en la falla presentada, a partir de ahí se inicia la lógica de composición para reconocer la falla, hallar el flujo necesario del servicio para realizar la reconfiguración, escoger y entregar los candidatos correspondientes al componente encargado de ejecutar el algoritmo y terminar con la ejecución del mediador del servicio.

El escenario experimental para realizar las pruebas se llevó a cabo en un computador con procesador AMD Turion X2 de 2.1 GHz, 3 gigas de RAM y con el sistema operativo Linux-Ubuntu 9.04 de 32 bits al cual se enviaba peticiones SIP desde otro computador donde se ejecutaron dos softphone X-Lite 4 (provisto por la corporación Counterpath) hacia la plataforma Mobicents JAIN SLEE 1.1 version 2.3.0 donde se implementaron el módulo de composición, el mediador y el servicio *CallMsgService* en el lenguaje Java haciendo uso del plugin EclipSLEE versión 1.2.6 provisto por la comunidad de desarrolladores de esta plataforma.

4.5.3 Criterios de evaluación

Con el fin de determinar la eficiencia del algoritmo, se realizaron pruebas de rendimiento que permiten obtener los tiempos de respuesta de este, Para ello se tienen en cuenta los criterios de la tabla 1.

Descripción	Criterio	Considerado para
Requerimientos de tiempo típicos en un servidor de aplicaciones de telecomunicaciones según la prueba de laboratorio realizada en [43]	20 ms para aplicaciones simples	Este criterio se utiliza para definir el umbral que limita al algoritmo de composición dinámica en el SLEE
	200 ms para aplicaciones complejas	
Requerimientos de tiempo definidos por la recomendación G.1010 de la ITU [58]	Voz, preferido < 150 ms	Este criterio se utiliza con el fin de realizar un análisis de los umbrales resultantes del algoritmo de composición dinámica, respecto a los requerimientos en un entorno real
	Voz, límite < 400 ms	
Tiempos definidos para la comunicación síncrona y asíncrona según [41]	Síncrona, tiempo de ida, 1.03 ms	Este criterio se utiliza para analizar los tiempos que necesita el módulo de composición, según los eventos que se envíen hacia los componentes implicados.
	Síncrona, tiempo de regreso, 0.47 ms	
	Asíncrona, tiempo de ida, 13 ms	
	Asíncrona, tiempo de regreso, 1.6 ms	

Tabla 2. Criterios de evaluación

4.5.4 Plan de pruebas

Con el fin de evaluar el rendimiento del algoritmo de composición dinámica se diseñó un plan de pruebas sobre el módulo de composición dinámica y el mediador, los cuales contemplan la fase de reconfiguración del proceso de composición dinámica. Para realizar las pruebas se tuvo en cuenta el servicio *CallMsgService,* al cual, se le modificaron sus componentes SBB con el fin de que fallen en tiempo de ejecución.

En la tabla 2 se describen las pruebas de rendimiento realizadas al módulo de composición y mediador.

Prueba	Descripción
PR1	Determina el retardo generado por todo el proceso de reconfiguración. Para esta prueba, primero se mide el tiempo que tardan normalmente las funcionalidades del servicio *CallMsgService*, y después se mide el tiempo que toman las funcionalidades cuando ocurre una falla en los SBB que las ejecutan y es ejecutado el módulo de composición para corregirlas. Para esta prueba se asume que el primer candidato ejecuta exitosamente el evento que falló.
PR2	Determina el tiempo que tarda el módulo de composición para ejecutar exitosamente un evento que haya fallado en el servicio *CallMsgService.* Para esta prueba se varía el número de candidatos que procesan dicho evento.
PR3	Determina el tiempo que tarda el módulo de composición en reconfigurar en forma simultánea los servicios. Para esta prueba se varía el número de servicios que necesitan ser reconfigurados en el mismo instante.

Tabla 3. Plan de pruebas

4.5.5 Resultados y discusión

En esta sección se presentan los resultados de las pruebas descritas en el banco de pruebas, así como también el análisis de cada una de ellas.

Con el fin de tomar las muestras de tiempo, se utilizó la función de java *System.CurrentTimeMillis()* la cual permite determinar el instante de tiempo en milisegundos cuando ésta es ejecutada, de esta manera se definieron líneas de código como puntos de control tanto al inicio como al final de la lógica del servicio de prueba, mediador y módulo de composición, para medir el tiempo de respuesta a partir de la diferencia de las muestras de tiempo para cada módulo.

4.5.5.1 Prueba de rendimiento PR1

En esta sección se presentan los resultados de la comparación de tiempos realizada al servicio *CallMsgService* cuando se ejecutan normalmente todas sus funcionalidades y cuando estas mismas presentan una falla en el componente SBB respectivo. A continuación se menciona como se realizó las medidas sobre algunas de las funcionalidades utilizadas para esta prueba como fueron la de mensajería y la de llamada de voz.

La prueba de mensajería consiste en la ejecución del evento *Message* por parte del SBB *Message* (ver figura 13), cuando se realiza normalmente el procesamiento del evento, el SBB *Message* envía el evento *MessageCounterEvent* al SBB *MessageBilling*, cuando se simula la falla del procesamiento del evento *Message,* el SBB *Message* envía el evento personalizado *AlarmEvent* al módulo de composición, el cual, con el primer candidato realiza la ejecución exitosa del evento *Message*, y envía el evento personalizado *ReconfigEvent* al servicio, en el cual, el mediador se encarga de enviar el evento resultante *MessageCounterEvent* al SBB MessageBilling.

La prueba de llamada de voz consiste en la ejecución del evento *Invite* por parte del SBB *B2BUA* como se puede ver en la figura 13, si se realiza normal el procesamiento del evento, se establece una llamada entre 2 softphone, si falla el procesamiento del evento, el SBB B2BUA envía el evento personalizado *AlarmEvent* al

módulo de composición dinámica, el cual ejecuta exitosamente el evento con el primer candidato, permitiendo de esta manera que la llamada sea establecida.

La diferencia entre las pruebas de las 2 funcionalidades anteriormente explicada consiste en que para la falla de la funcionalidad de mensajería el módulo de composición envía un evento resultante al mediador del servicio, y para la falla de la funcionalidad de llamada de voz, el procesamiento del evento Invite no genera evento resultante sino que solo crea la sesión entre los usuarios, de tal forma no necesita comunicación con el mediador cuando se realiza la reconfiguración , ya que el proceso termina en el procesamiento del evento por parte del candidato. A partir de la descripción de estas dos funcionalidades, queda claro la forma de realizar las pruebas para las demás funcionalidades, ya que la principal diferencia en los flujos de ejecución era el caso de que exista evento resultante o no, por tal motivo para la funcionalidad de registro su ejecución termina en el componente Register, y para la funcionalidad de tarificación de llamada, que se ejecuta con el evento Bye, se genera evento resultante y termina con el procesamiento del evento disparado *CallBillingEvent* hacia el siguiente SBB *CallBillingEvent.*

Para realizar esta prueba se tomaron diez muestras de tiempo para cada funcionalidad y se calculó el promedio de tiempo en que cada funcionalidad llevó a cabo todas sus tareas, el promedio se calculó a partir de la suma de los tiempos de respuesta de cada funcionalidad sobre el número de muestras de tiempo. Se tomaron en total cuarenta muestras para la ejecución normal de cada funcionalidad y cuarenta muestras para la ejecución de las funcionalidades con reconfiguración, dando un total de ochenta muestras para esta prueba.

A continuación se observa parte de una notificación en consola del servidor donde se resaltan el tiempo inicial y final para la ejecución normal de la funcionalidad de mensajería.

```
15:27:21,479 INFO  [MediatorSbb] recibio un Message el servicio
15:27:21,484 INFO  [RegisterSbb] Ingresando al RegisterSBB-event
Message
15:27:21,484 INFO  [RegisterSbb] Tiempo incio de mensajeria
instantanea: 1308515241484
15:27:21,492 INFO  [MessageSbb] Ingreso Event message
15:27:21,492 INFO  [MessageSbb] Message
```

…

```
15:27:21,535 INFO  [MessageBillingSbb] Con los siguientes datos:
15:27:21,536 INFO  [MessageBillingSbb] Name: 222
15:27:21,537 INFO  [MessageBillingSbb] Uri: sip:222@
192.168.0.11:1024;rinstance=41010a6b3f2801b4
15:27:21,538 INFO  [MessageBillingSbb] minutes: 274
15:27:21,538 INFO  [MessageBillingSbb] messages: 1
15:27:21,539 INFO  [MessageBillingSbb] Tiempo final de mensajeria
instantanea: 1308515241539
```

El tiempo total de esta ejecución es: `1308515241539 ms - 1308515241484 ms = 55 ms`

De esta misma manera se tomó el tiempo para cada una de las muestras de todas las funcionalidades.

A continuación en la tabla 3 y la figura 16 se muestran los resultados obtenidos.

Muestra #	**Reg Normal t(ms)**	**Reg Reconf t(ms)**	**Msg Normal t(ms)**	**Msg Reconf t(ms)**	**B2B Normal t(ms)**	**B2B Reconf t(ms)**	**Bye Normal t(ms)**	**Bye Reconf t(ms)**
1	90	82	55	155	151	346	70	192
2	13	43	27	165	58	121	17	95

3	15	72	38	267	18	185	20	173
4	17	59	46	110	17	107	45	179
5	12	44	54	169	18	85	35	168
6	8	78	26	108	22	73	59	83
7	13	81	21	87	22	92	24	132
8	19	53	28	127	20	54	64	189
9	10	53	58	147	13	73	38	175
10	11	45	20	153	23	77	53	185
Promedio	**20,8**	**61**	**37,3**	**148,8**	**36,2**	**121,3**	**42,5**	**157,1**

Tabla 4. Resultados de la prueba PR1

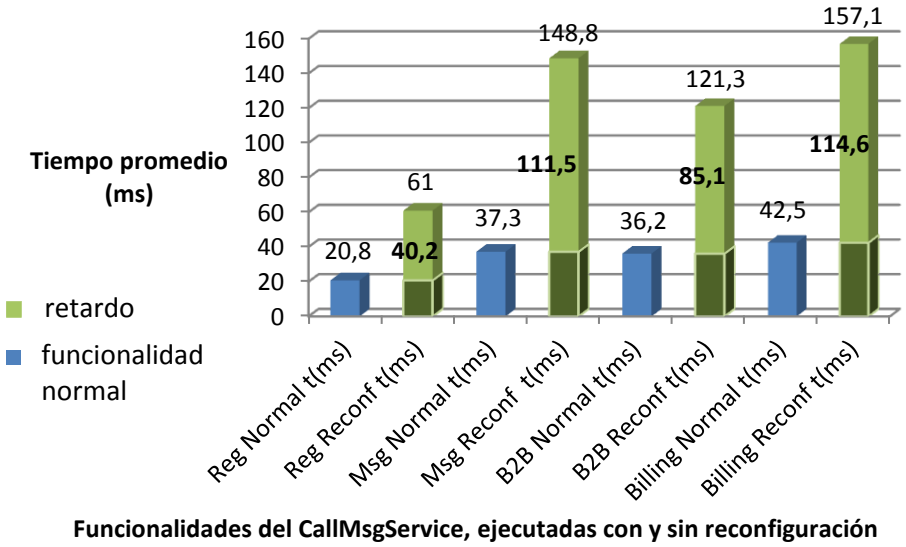

Figura 16. Resultados gráficos de la prueba PR1

En la figura 16, se observa que como se esperaba, existe un retardo (diferencia entre los promedios de tiempo de la funcionalidad normal y la funcionalidad con reconfiguración) generado por el módulo de composición a cada una de las funcionalidades estudiadas, esto es debido a que el módulo de composición dinámica hace uso de las comunicaciones síncrona y asíncrona para poder realizar la reconfiguración del servicio, que como se afirma en [41] (ver tabla 1), generan un retardo diferente, particularmente, la comunicación asíncrona se lleva a cabo en un tiempo más prolongado que la comunicación síncrona, debido a que el envió del evento dura más tiempo al usar el ER. Por otra parte, el retardo generado por el procesamiento realizado por el módulo, según el requerimiento de tiempo necesario para los servicios de telecomunicaciones, según [43] el cual muestra en la tabla 1, se puede establecer como una aplicación que funciona por debajo del umbral de 200 ms establecido para las aplicaciones complejas.

También se puede observar en la figura 16 que el proceso de reconfiguración de la funcionalidad de mensajería tiene un retardo de 111.5 ms, mayor que el proceso de reconfiguración de la funcionalidad de llamada por voz, cuyo retardo es de 85.1 ms. Esto se debe a que, como ya se mencionó, el procesamiento del evento *Message* por parte del SBB *Message* origina el evento *MessageCounterEvent* el cual se entrega al *SBB MessageBilling* mientras que la funcionalidad de llamada no origina un evento resultante, de esta manera, cuando ocurre una falla para la funcionalidad de mensajería, además del envío del evento *AlarmEvent* al módulo de composición, y el procesamiento realizado por este, (también realizados por la funcionalidad de llamada de voz), es necesario el envío del evento resultante desde el módulo de

composición al servicio (haciendo uso de la comunicación asíncrona) el cual lo recibe y lo procesa el mediador y procede a enviarlo (haciendo uso de la comunicación síncrona) al SBB *MessageBilling.*

De lo anterior se puede deducir que el mediador genera un retardo de 26.4 ms para la funcionalidad de mensajería, respecto a la funcionalidad de llamada por voz, el cual aumenta el tiempo total del proceso de reconfiguración, igualmente para el caso de la funcionalidad de llamada el tiempo de reconfiguración es más corto que de la tarificación de llamada, debido a que la funcionalidad de tarificación necesita el mediador para manejar un evento resultante, donde se aprecia que el mediador genera un retardo de 29.5 ms del tiempo total de este proceso. Por lo tanto se puede apreciar aproximadamente del análisis de la gráfica que el mediador genera un retardo promedio de 28 ms y el módulo de composición genera un retardo promedio de 75.85 ms, según los promedios de las funcionalidades de registro y llamada de voz. De esta manera, se puede decir que, del análisis de las funcionalidades que implican la intervención del mediador, y la que no, el retardo más grande en el proceso de reconfiguración, se genera por el módulo de composición.

Respecto a los criterios de evaluación, se puede decir que el algoritmo de composición dinámica teniendo en cuenta el reenvío y no reenvío de evento resultante hacia el servicio y considerando que el primer candidato ejecuta con éxito el evento que falló en el servicio, está por debajo del umbral de 200 ms establecido para aplicaciones complejas en un servidor de aplicaciones en el dominio de telecomunicaciones. Además se puede decir que los retardos generados, no solo se deben al procesamiento que realiza el módulo de composición, sino a los procesos de comunicación síncrona y asíncrona necesarios para llevar a cabo dicho procesamiento. Por otro lado, según los requerimientos de tiempo definidos por la recomendación G.1010 de la ITU [58], la funcionalidad de llamada con reconfiguración, considerada en la prueba de laboratorio establecida en este trabajo de grado, no sobrepasa el umbral preferido de 150 ms para entornos reales.

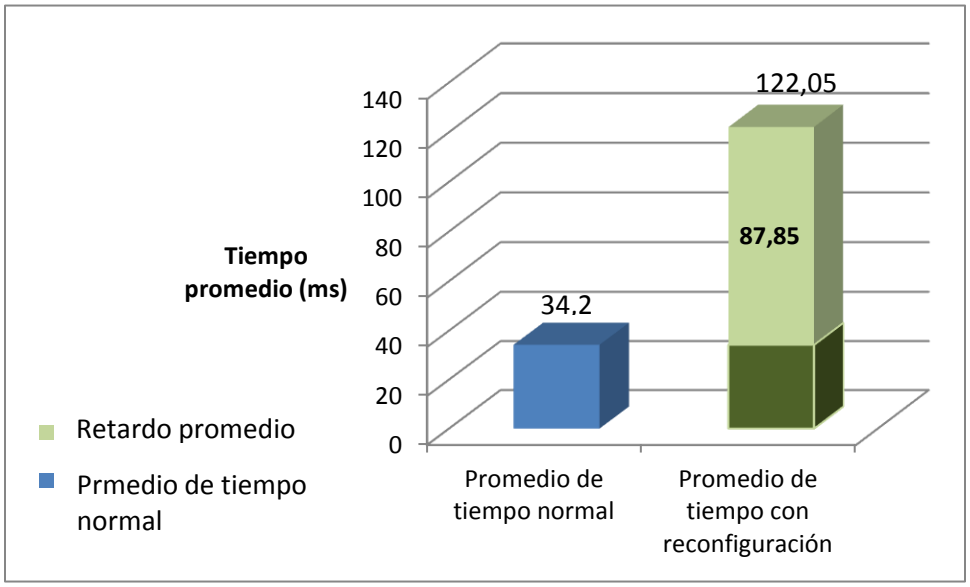

Figura 17. Promedio de tiempo de las funcionalidades del CallMsgService con y sin reconfiguración

En la figura 17 se puede observar la gráfica que muestra el promedio del tiempo que necesitan las funcionalidades del servicio *CallMsgService* para ejecutarse normalmente (34,2 ms), y el promedio de tiempo que necesitan para ejecutarse cuando necesitan del proceso de reconfiguración (122,05 ms), el cual se realiza considerando que el primer candidato procesa exitosamente el evento que haya fallado. Se puede observar que el retardo de tiempo promedio generado es de 87.85 ms.

En conclusión, con esta prueba se puede afirmar que aun realizando el proceso de reconfiguración, el cual genera un retardo de 122.05 ms, el servicio se mantiene por debajo de los tiempos máximos de 200 ms para aplicaciones complejas establecidos para el retardo en [43].

4.5.5.2 Prueba de rendimiento PR2

En esta sección se presentan los resultados de las pruebas de tiempo realizadas al módulo de composición variando el número de candidatos para procesar con éxito el evento que falló en una reconfiguración del servicio *CallMsgService*. Para esta prueba se consideró la falla del procesamiento del evento Register realizada por el SBB Register (ver figura 13) y un número total de 20 candidatos.

Para realizar esta prueba, el SBB Register fracasa al procesar el evento Register y procede a enviar el evento *AlarmEvent* al módulo de composición. El módulo reconoce la falla y obtiene los candidatos correspondientes para realizar la reconfiguración a partir de la ejecución del algoritmo descrito en la sección 3.2.2.6, en el cual, para lograr el procesamiento con éxito del evento, envía el Register (obtenido como un atributo del evento *AlarmEvent*) hacia los candidatos, si el primero de ellos no lo procesa, se lo envía al segundo, si éste no lo procesa, se lo envía al tercero, este proceso se repite hasta que uno de los candidatos procese correctamente el evento y dé por terminado la reconfiguración del servicio.

Con el fin de determinar el comportamiento del algoritmo, en esta prueba se varió el número de candidatos que fracasa en el procesamiento de un evento que falla en una reconfiguración del servicio de la siguiente forma:

- en la primera ejecución, el primer candidato ejecuta correctamente el evento,
- la segunda vez, el primer candidato falla, y el segundo candidato ejecuta correctamente el evento,
- la tercera vez, los dos primeros candidatos fallan, y el tercer candidato ejecuta correctamente el evento,
- la cuarta vez, los tres primeros candidatos fallan, y el cuarto candidato ejecuta correctamente el evento.
- así sucesivamente.

Para realizar esta prueba se tomaron diez muestras de tiempo por cada vez que se varió el número de candidatos que fracasaron al procesar el evento Register y se calculó el promedio de tiempo en cada caso, el promedio se calculó a partir de la suma de los tiempos de respuesta para cada caso en que se varió los candidatos sobre el número de muestras de tiempo. Se tomó un total doscientas muestras para la reconfiguración, variando el número de candidatos.

A continuación se observa parte de una notificación en consola del servidor donde se resaltan el tiempo inicial y final para la reconfiguración del servicio con 2 candidatos.

```
01:08:29,685 INFO  [MediatorSbb] recibio un register el servicio
01:08:29,689 INFO  [RegisterSbb] Ingresando al RegisterSBB-event
Register
01:08:29,691 INFO  [RegisterSbb] Se disparo un punto de control-
simular falla
01:08:29,697 INFO  [ReconfigurationSbb] Se recibe el evento
AlarmEvent -Notificacion de falla de servicio
01:08:29,719 INFO  [ReconfigurationSbb] Tiempo de final de
AlarmEvent-inicio de la reconfiguracion-inicio del
modulo1308550109718

…

01:08:29,760 INFO  [ReconfigurationSbb] evento resultante :null
01:08:29,760 INFO  [ReconfigurationSbb] Se realizo la
reconfiguracion con exito-no tiene evento resultante para
redireccionar al servicio
01:08:29,760 INFO  [ReconfigurationSbb] Tiempo de final del
algoritmo-tiempo final de la reconfiguracion-tiempo final del
modulo1308550109760
```

El tiempo total de esta ejecución es: `1308550109760 ms - 1308550109718 ms = 42 ms`

De esta misma manera se tomó el tiempo para cada número de SBB candidatos

A continuación, en la tabla 4 se muestran los resultados obtenidos.

		Muestra #										**Promedio**
		1	2	3	4	5	6	7	8	9	10	
Número de candidatos	**1 t(ms)**	40	21	30	17	15	16	16	59	15	37	**26,6**
	2 t(ms)	21	43	46	20	28	35	82	59	36	35	**40,5**
	3 t(ms)	78	59	113	37	36	94	53	48	60	46	**62,4**
	4 t(ms)	51	44	56	33	50	59	52	52	36	82	**51,5**
	5 t(ms)	58	42	37	60	38	57	38	11	45	41	**42,7**
	6 t(ms)	57	48	44	49	57	49	61	59	48	72	**54,4**
	7 t(ms)	66	161	93	49	66	62	103	90	53	59	**80,2**
	8 t(ms)	171	62	96	90	67	80	67	129	54	100	**91,6**
	9 t(ms)	138	93	61	149	81	98	64	64	140	115	**100,3**
	10 t(ms)	139	70	85	79	103	101	122	83	87	145	**101,4**
	11 t(ms)	126	170	102	76	117	162	73	73	135	91	**112,5**
	12 t(ms)	113	113	149	142	134	165	170	90	152	130	**135,8**
	13 t(ms)	101	81	157	84	84	181	118	164	149	149	**126,8**
	14 t(ms)	176	189	126	169	133	133	131	166	166	116	**150,5**
	15 t(ms)	112	181	129	126	190	174	192	141	86	194	**152,5**
	16 t(ms)	224	180	157	251	131	192	145	160	102	104	**164,6**
	17 t(ms)	111	119	106	112	130	119	168	103	110	120	**119,8**
	18 t(ms)	120	227	133	144	192	168	211	168	180	217	**176**
	19 t(ms)	138	106	131	166	115	105	131	130	256	191	**146,9**
	20 t(ms)	323	181	240	115	120	109	130	163	133	186	**170**

Tabla 5.Resultados de la prueba PR2

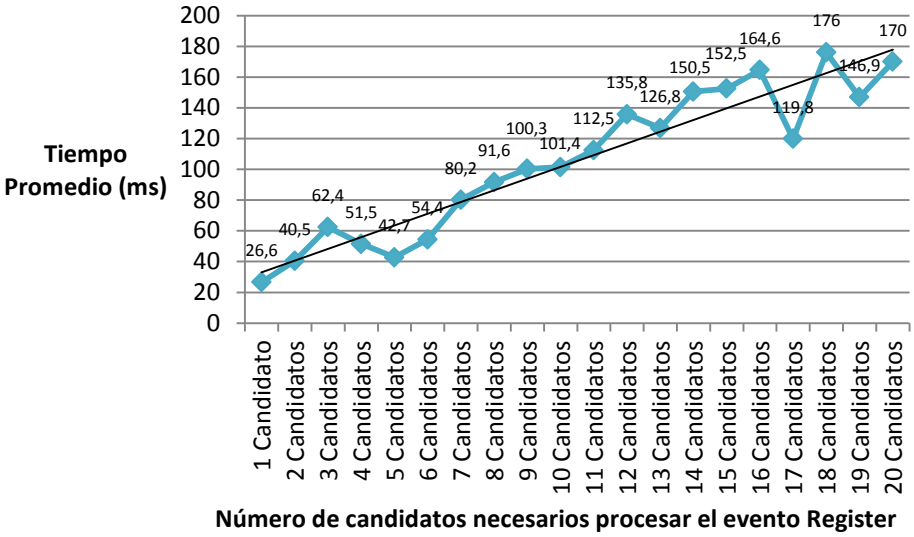

Figura 18. Resultados gráficos parciales de la prueba PR2

En la figura 18, se muestra la gráfica obtenida a partir de los datos de la tabla 4, en la cual se obtiene el tiempo de respuesta del candidato sin tener en cuenta todo el proceso de reconfiguración, el cual comprende desde que se dispara el evento no procesado del servicio hacia el candidato, hasta que se recibe la notificación por parte del mismo. En la figura 18, se observa como varían los tiempos de respuesta respecto al aumento de los candidatos para realizar el procesamiento del evento del componente SBB (Register). En esta grafica se puede apreciar un comportamiento lineal del algoritmo, ya que a medida que aumenta el número de candidatos para realizar con éxito el procesamiento del evento, el tiempo aumenta proporcionalmente como se aprecia en los puntos de la gráfica que representan los promedios de los tiempos de respuesta para cada variación del número de candidatos, considerándose cierto margen de error que se visualiza con los picos en la gráfica debido al incremento y disminución del tiempo al variar el número de candidatos, lo cual se debe a factores de rendimiento del sistema en el cual se realizaron las pruebas, como por ejemplo: la variación de velocidad de procesamiento de equipo, saturación de la memoria del sistema, congestión de envió de peticiones SIP por parte de los terminales y la red hacia el servidor provocado por la inestabilidad del entorno experimental utilizado.

Se observa que en el algoritmo, los tiempos de respuesta seguirán variando proporcionalmente al número de candidatos necesarios para procesar con éxito el evento, con lo cual, entre mayor sea el número de candidatos aumenta la probabilidad de generar tiempos por encima del valor de referencia (200 ms) definido como aceptable para los tiempos de respuesta de los servidores de aplicaciones del entorno de la telecomunicaciones para la ejecución de servicios complejos mencionados en los criterios de evaluación.

Con el fin de estimar el tiempo promedio que se genera cuando se varía el número de candidatos considerando todo el proceso de reconfiguración, se generó la gráfica de la figura 19 a partir de la suma de una constante a los valores de la gráfica de la figura 18. La constante se obtuvo haciendo la diferencia entre el valor de tiempo promedio de la funcionalidad de registro con reconfiguración que considera todo el proceso de reconfiguración con un candidato, cuyo valor según la figura 16 es de 61 ms y el tiempo promedio de la misma funcionalidad contemplando solamente la ejecución del candidato, cuyo valor según la figura 18 es de 26,6 ms para un solo candidato, de esta manera la constante es de 34,4 ms.

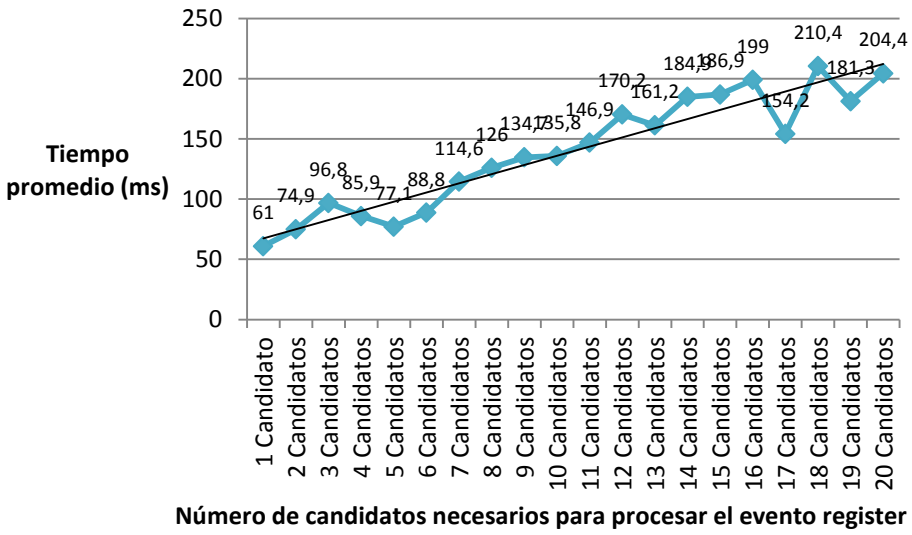

Figura 19. Resultado total de la prueba PR2

En la figura 19 se puede apreciar el tiempo de respuesta total de la reconfiguración, el cual comprende desde la llegada del evento *AlarmEvent* hasta que se notifica si el proceso de reconfiguración fue realizado con éxito o fracaso para cada caso en que se varía el número de candidatos para procesar con éxito el evento que falló. Se aprecia que al tener en cuenta todo el proceso de reconfiguración, los tiempos aumentaron, pero se mantiene la relación proporcional del aumento del tiempo respecto al número de candidatos, el cual se aprecia en el comportamiento lineal de la gráfica. También se observa que el algoritmo de composición dinámica establece un rendimiento aceptable para un número menor o igual a 15 candidatos donde no se evidencia un margen de error considerable, con un promedio de 186.9 ms, el cual está por debajo del valor de referencia (200 ms) establecido en los criterios de evaluación para tiempos aceptables para requerimientos de servidores de telecomunicaciones en la ejecución de servicios complejos.

4.5.5.3 Prueba de rendimiento PR3

En esta sección se presentan los resultados del tiempo que requiere el algoritmo para reconfigurar en forma simultánea varios servicios; para esto los servicios desplegados en el servidor tienen una funcionalidad que falla al momento de ejecutarse.

Para llevar a cabo esta prueba se realizaron 10 versiones similares del servicio *CallMsgService,* las cuales fallan en el procesamiento del evento *Register,* por lo tanto, el componente *SBB Register* procede al envió del evento AlarmEvent al módulo de composición dinámica, el cual es procesado exitosamente con el primer candidato.

En la tabla 5 se muestra el resultado de la prueba PR3, en esta se muestran los tiempos que requiere el algoritmo para reconfigurar simultáneamente desde un servicio, hasta diez servicios, para los cuales se tomaron 5 muestras de tiempo transcurrido desde el primer servicio hasta el último que fue reconfigurado.

Para realizar esta prueba se tomaron cinco muestras de tiempo por cada vez que se varió el número de servicios desplegados y se calculó el promedio de tiempo en cada caso, el promedio se calculó a partir de la suma de los tiempos de respuesta para cada caso en que se varió el número de servicios sobre el número de muestras de tiempo. Se tomó un total cincuenta muestras para las reconfiguraciones, variando el número de servicios desplegados.

A continuación se observa parte de una notificación en consola del servidor donde se resaltan el tiempo inicial y final para la reconfiguración del servicio con 2 candidatos.

```
23:28:20,595 INFO  [RegisterSbb] Se disparo un punto de control-
simular falla
23:28:20,598 INFO  [Mediator1Sbb] recibio un register el servicio
23:28:20,604 INFO  [Register_1Sbb] Ingresando al RegisterSBB-
event Register
23:28:20,605 INFO  [Register_1Sbb] Se disparo un punto de
control-simular falla
23:28:20,609 INFO  [ReconfigurationSbb] Se recibe el evento
AlarmEvent -Notificacion de falla de servicio
23:28:20,609 INFO  [ReconfigurationSbb] Tiempo de final de
AlarmEvent-inicio de la reconfiguracion-inicio del
modulo1308889700609
```

…

```
23:28:20,736 INFO  [ReconfigurationSbb] Se proceso con exito el
evento por el candidato
23:28:20,736 INFO  [ReconfigurationSbb] evento resultante :null
23:28:20,736 INFO  [ReconfigurationSbb] Se realizo la
reconfiguracion con exito-no tiene evento resultante para
redireccionar al servicio
23:28:20,736 INFO  [ReconfigurationSbb] Tiempo de final del
algoritmo-tiempo final de la reconfiguracion-tiempo final del
modulo1308889700736
```

El tiempo total de esta ejecución es: `1308889700736 ms - 1308889700609 ms = 137 ms`

De esta misma manera se tomó el tiempo para cada número de servicios desplegados.

A continuación, en la tabla 6 se muestran los resultados obtenidos.

		Muestra #					Promedio
		1	2	3	4	5	
Servicios	**1 t(ms)**	74,4	55,4	64,4	51,4	49,4	**59**
	2 t(ms)	161	137	107	107	135	**129,8**
	3 t(ms)	148	158	158	155	155	**155,2**
	4 t(ms)	198	213	208	170	157	**189,6**
	5 t(ms)	1202	461	235	281	342	**504,6**
	6 t(ms)	274	276	197	325	269	**268,6**
	7 t(ms)	301	289	1026	365	702	**537**
	8 t(ms)	282	331	369	338	286	**321,6**
	9 t(ms)	1028	825	364	354	358	**586,2**
	10 t(ms)	1252	700	469	302	386	**622,2**

Tabla 6. Resultados de la prueba PR3

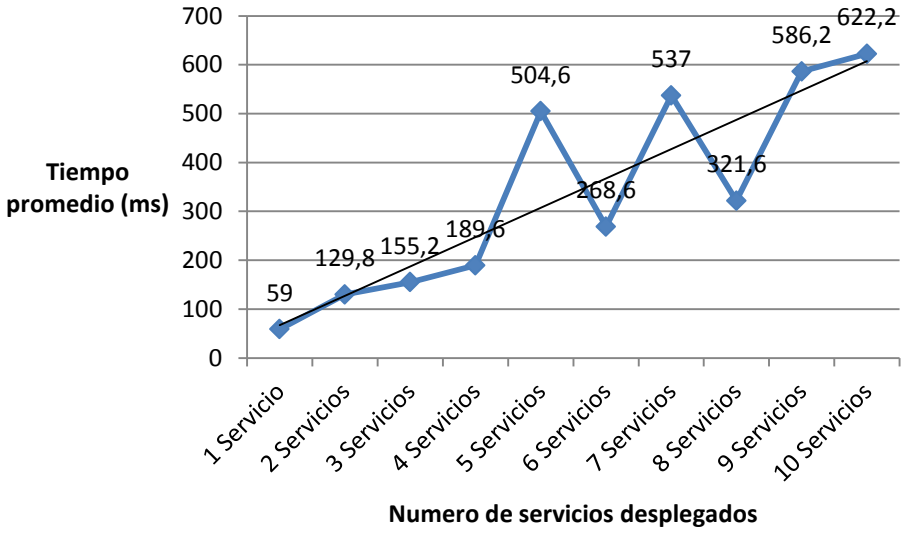

Figura 20.Resultados gráficos de la prueba PR3

En la figura 20 se puede observar que a medida que se aumenta el número de servicios que necesitan del proceso de reconfiguración, considerando la funcionalidad de registro, también lo hace el tiempo promedio necesario para llevar a cabo dicho proceso. Aunque existen algunos picos pronunciados, debido a la variación del consumo de recursos del equipo donde se realizó la prueba, es evidente el comportamiento lineal del algoritmo, permitiendo, de esta manera deducir que a medida que aumente el número de servicios que necesiten del proceso de reconfiguración, el tiempo también lo hará en una forma proporcional a dicho número.

Considerando los requerimientos de tiempo típicos en un servidor de aplicaciones de telecomunicaciones según [43], propuesto en la tabla 1, se puede decir que hasta 4 servicios que ejecuten la funcionalidad de registro, el tiempo de reconfiguración para el último servicio es aceptable dentro del tiempo de 200 ms considerado para las aplicaciones complejas; a partir de 5 servicios, el tiempo necesario para reconfigurar el último servicio, ya sobrepasa el umbral de tiempo de 200 ms. Se debe tener en cuenta que se puede variar el número de servicios reconfigurados que estén por debajo del tiempo de referencia ya mencionado, dependiendo de la funcionalidad del servicio que se ejecute, se podría obtener un número menor de servicios bajo el umbral establecido de 200 ms, es decir, el proceso también depende de la complejidad del servicio.

Con el fin de estimar los promedios de tiempo que genera el algoritmo cuando varios servicios con diferentes funcionalidades necesitan del proceso de reconfiguración simultáneamente, se generó la tabla 7, en la que a cada funcionalidad (excepto la de Register) tiene un tiempo promedio para cada número de servicios, que se calculó sumando una constante a los tiempos promedios de la funcionalidad de Register, calculados en la tabla 6. La constante para cada funcionalidad se obtuvo de la tabla 16, restando el promedio de tiempo de cada funcionalidad (considerando el proceso de reconfiguración), con el promedio de tiempo de la funcionalidad de Register. Por ejemplo, para la funcionalidad de Message, la constante es igual a: 148,8 ms – 61 ms = 87,8 ms, de esta manera, cuando dos servicios necesitan de reconfiguración, considerando la funcionalidad de Register, el tiempo promedio estimado es 129,8 ms + 87,8 ms = 217,6 ms.

Funcionalidad con reconfiguración	Tiempo promedio según el número de servicios									
	1 servicio	2 servicios	3 servicios	4 servicios	5 servicios	6 servicios	7 servicios	8 servicios	9 servicios	10 servicios
Register	59	129,8	155,2	189,6	504,6	268,6	537	321,6	586,2	622,2
Message	146,8	217,6	243	277,4	592,4	356,4	624,8	409,4	674	710
B2B	119,3	190,1	215,5	249,9	564,9	328,9	597,3	381,9	646,5	682,5
Billing	155,1	225,9	251,3	285,7	600,7	364,7	633,1	417,7	682,3	718,3
Promedio	**120,05**	**190,85**	**216,25**	**250,65**	**565,65**	**329,65**	**598,05**	**382,65**	**647,25**	**683,25**

Tabla 7. Promedios de tiempo para varios servicios con diferentes funcionalidades

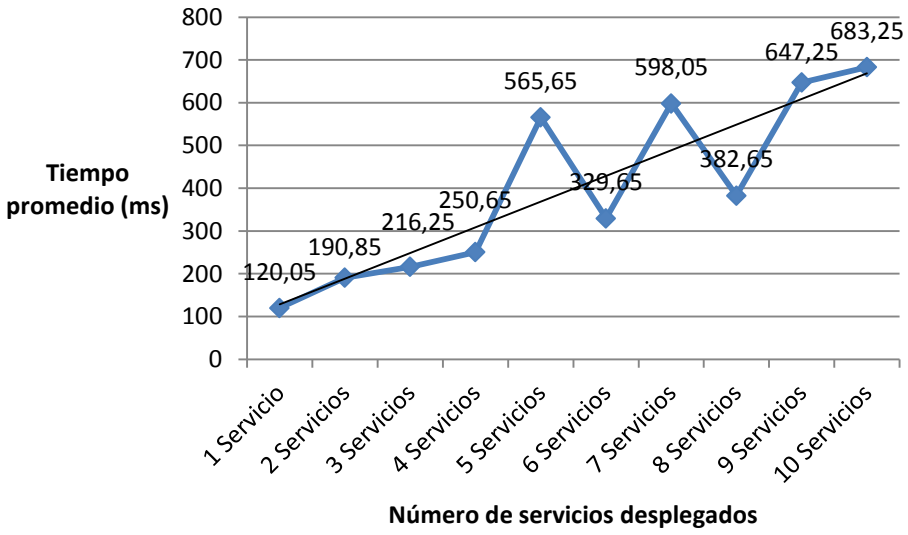

Figura 21. Resultados gráficos estimados para varios servicios con diferentes funcionalidades

Según los requerimientos de tiempo típicos en un servidor de aplicaciones de telecomunicaciones [43], propuesto en la tabla 1 y considerando que los servicios que requieren una reconfiguración simultánea podrían ejecutar cualquier funcionalidad, se puede estimar que el tiempo promedio de reconfiguración para el último servicio, aceptable dentro del tiempo de 200 ms considerado para las aplicaciones complejas, se tiene hasta un máximo de dos servicios simultáneos (190,85 ms), a partir de 3 servicios, el tiempo necesario para reconfigurar el último servicio, ya sobrepasa el umbral de tiempo de 200 ms.

Capítulo 5
Aportes, trabajo futuro y conclusiones

5.1 Aportes

- La formalización de la composición dinámica en el contexto de JAIN SLEE 1.1, en el cual se modela el comportamiento adecuado del proceso, enfocado hacia la reconfiguración de servicios, por medio de la representación formal de FSM, la cual permite abstraer de una forma muy aproximada el funcionamiento de servicios de telecomunicaciones, tales como los servicios SIP.

- Un algoritmo de composición dinámica para servicios SIP basado en FSM enfocado en realizar reconfiguración de componentes de construcción de servicios dentro de un entorno basado en JAIN SLEE 1.1 en el caso de ocurrir cambios en tiempo de ejecución que afecten el funcionamiento de un servicio.

- Un módulo de composición dinámica y un componente mediador para servicios SIP desarrollado dentro de la plataforma Mobicents basada en la arquitectura de JAIN SLEE 1.1 para ejecutar el algoritmo propuesto para la reconfiguración de servicios en el momento de ocurrir fallas en tiempo de ejecución en el procesamiento de un evento en un componente SBB.

- Una arquitectura de referencia para la reconfiguración de servicios en tiempo de ejecución por medio de mecanismos de composición dinámica.

5.2 Trabajo futuro

Se presentan varias ideas de trabajos futuros con el fin de continuar con el presente trabajo de grado e integrar los resultados con los aportes generados hasta el momento.

- Proponer mecanismos de descubrimiento de servicios basados en técnicas de sintáctica, semántica o QoS, con el fin de encontrar y seleccionar SBB candidatos con características similares a los SBB de los servicios alojados dentro del SLEE y poder así llevar a cabo el proceso de composición dinámica enfocado hacia la reconfiguración.

- Proponer mecanismos de síntesis y orquestación de servicios que permitan generar el plan para lograr el comportamiento deseado de los servicios a partir de la interacción de los componentes SBB y posteriormente coordinar el flujo de control y datos de dichos componentes.

- Implementar la arquitectura de referencia de composición dinámica, integrando las cuatro fases propuestas (Descubrimiento, síntesis y orquestación, monitoreo y reconfiguración de servicios) que permiten realizar composición dinámica en el SLEE.

- Adaptar el modelo de SOA a un entorno de ejecución de lógica de servicio integrando el concepto de composición dinámica para mejorar la forma de realizar composición de servicios de telecomunicaciones.

5.3 Conclusiones

- Se caracterizó la composición dinámica de servicios SIP para un entorno de ejecución de lógica de servicio basado en la arquitectura de JAIN SLEE 1.1, describiendo los atributos más relevantes de

cómo realizar este proceso, teniendo en cuenta las características del SLEE, dando como resultado la descripción de las fases de composición necesarias, y un modelo formal de composición utilizando las FSM para representar el comportamiento deseado de componer dinámicamente, enfocado en la reconfiguración de servicios a nivel de componentes SBB.

- Se proporcionaron mecanismos que permiten realizar composición dinámica en el SLEE, utilizando FSM, enfocada en la fase de reconfiguración, ya que ésta se consideró como la más crítica puesto que se encarga de la adaptación de cambios que alteran el funcionamiento normal de los servicios en tiempo de ejecución, tales como posibles fallas en los componentes SBB.

- A partir de la investigación realizada en este trabajo de grado se consideró que la fase de síntesis y orquestación al igual que la fase de descubrimiento, requieren de tiempos elevados de respuesta, afectando de forma negativa el desarrollo de algún mecanismo de estas fases, para su funcionamiento en tiempo de ejecución dentro de un SLEE, debido que los tiempos de respuestas para la ejecución servicios de telecomunicaciones deben ser mínimos para lograr un nivel aceptable de calidad. De esta manera se concluye que estos procesos deben ser realizadas en una etapa previa a la reconfiguración de los servicios.

- Teniendo en cuenta el modelo general de enrutamiento de eventos utilizado dentro del SLEE y el uso de adaptadores de recursos, el desarrollo de los servicios de telecomunicaciones considerando diferentes protocolos, es adaptado a la lógica de eventos del SLEE, de esta manera, los servicios son construidos de forma similar dentro de este entorno, por tal motivo los mecanismos de composición dinámica establecidos para los servicios SIP son fácilmente adaptables a diferentes contextos de servicios.

- Se generó un algoritmo de composición dinámica de servicios SIP en el SLEE basado en FSM y enfocado en reconfiguración, a partir de la adaptación de un algoritmo de selección, que contempla características de reconfiguración del contexto de los Servicios Web, teniendo en cuenta las características establecidas en este trabajo de grado para componer dinámicamente en el SLEE y las diferencias con el algoritmo planteado para el dominio de los WS, las cuales son: la selección de componentes en el SLEE es contemplado antes del proceso de reconfiguración mientras que en el dominio de los WS se contempla durante dicho proceso, la composición dinámica en el SLEE se realiza a nivel de componentes SBB mientras que en los WS se realiza a nivel de Servicios Web atómicos, el modelo de composición dinámica del SLEE es atómico, mientras que en los Servicios Web se contempla todo el flujo de ejecución del WS compuesto, en el SLEE se contemplan estados de sincronización mientras que en los WS no se contemplan.

- Se midieron los tiempos de repuesta del proceso de reconfiguración realizados sobre un servicio SIP de prueba a partir de la evaluación del rendimiento del algoritmo de composición dinámica por medio de la implementación de un módulo de composición dinámica para servicios SIP en la plataforma de Mobicents basada en la arquitectura de JAIN SLEE 1.1. En esta evaluación se consideró el desarrollo de un componente mediador para el servicio debido a que la plataforma presenta ciertas restricciones con la comunicación asíncrona de eventos, dificultando la entrega de los eventos resultantes del SBB candidato hacia el SBB siguiente en el flujo de ejecución del servicio de prueba.

- Comparando los promedios de los tiempos de respuesta de la ejecución normal de las funcionalidades del servicio de prueba respecto a los promedios de los tiempos de respuesta con reconfiguración de las mismas funcionalidades se concluye que, debido al retardo generado por el procesamiento realizado en el módulo de composición, según el requerimiento de tiempo necesario para los servicios de telecomunicaciones definido en los criterios de evaluación, la reconfiguración se puede establecer como un proceso que funciona en torno al umbral de 200 ms

establecido para las aplicaciones complejas. Considerando así, un rendimiento aceptable para la ejecución del algoritmo de composición.

- Se observó que en el algoritmo, los tiempos de respuesta varían proporcionalmente al número de candidatos, según la prueba de medida de los tiempos de respuesta de la reconfiguración de un servicio, variando el número de candidatos necesarios para procesar con éxito el evento que falló, con lo cual se concluye que presenta un comportamiento lineal, teniendo en cuenta que con el aumento del número de candidatos necesarios, los tiempos de respuesta siguen subiendo hasta sobrepasar el umbral de 200 ms, por lo que se establece que para un número menor o igual a 15 candidatos utilizados en la reconfiguración de un servicio se obtienen tiempos de respuesta por debajo del umbral de 200 ms.

- Se observó un comportamiento lineal, a partir de los tiempos de respuesta de reconfiguraciones simultáneas de servicios que utilizaron un candidato para procesar con éxito el evento que falló, confirmando los resultados de la prueba PR2, con lo cual se puede concluir que, debido al comportamiento lineal que muestra el algoritmo de composición dinámica, su rendimiento es aceptable, ya que los tiempos de respuesta varían proporcionalmente al número de servicios que debe reconfigurar, de esta manera, si el número de servicios aumenta, es evidente que el proceso de reconfiguración tarda más tiempo en completar su labor.

Bibliografía

[1] C. Makaya, et al., "Services Composition based on Next-Generation Service Overlay Networks Architecture", New Technologies, Mobility and Security (NTMS) 4th IFIP International Conference on 2011.

[2] H. Lu, Y. Zheng, and Y. Sun, "The next generation sdp architecture: Based on soa and integrated with ims", Proceedings of the 2008 Second International Symposium on Intelligent Information Technology Application - Volume 03. Washington, DC, USA: IEEE Computer Society, 2008.

[3] N. Blum, et al., "An open carrier controlled service environment for user generated mobile multimedia services", Proceedings of the 7th International Conference on Advances in Mobile Computing and Multimedia, ser. MoMM '09. New York, NY, USA: ACM, 2009.

[4] B. Medjahed, A. Bouguettaya, and A.K. Elmagarmid, "Composing web services on the semantic web", The VLDB Journal, vol. 12, November, 2003.

[5] A. Brogi, S. Corfini, and R. Popescu, "Semantics-based compositionoriented discovery of web services", ACM Trans. Internet Technol., vol. 8, October, 2008.

[6] Á. Cruz, "Una nueva convergencia: ¿Java en la red?", InfoWorld, August 3, 2005, [Online]. Available:http://iworldcommx.web124.discountasp.net/iw_SpecialReport_read.asp?iwid=3827&back=2&HistoryParam=U. [Accessed March 16, 2011].

[7] O.C. Limited, "A SLEE for all Seasons", May 11, 2007.

[8] J. Rosenberg, "SIP: Session Initiation Protocol", RFC 3261, June, 2002, [Online]. Available:http://www.ietf.org/rfc/rfc3261.txt. [Accessed March 16, 2011].

[9] C.G.R. Milagros Fernández Gavilanes, Jesús Vilares, Ferro, Jorge Graña Gil, Antonio Blanco Ferro, "Teoría de Autómatas y Lenguajes Formales: Una aproximación práctica desde dos paradigmas de programación", Facultad de Informática, Universidade da Coruña, Campus de Elviña.

[10] R. Brena, "Automatas y lenguajes: un enfoque de diseño", Tec de Monterrey, México, 2003.

[11] P.Á. Mauricio Espinoza-Mejía, "El ciclo de vida de un servicio Web compuesto: virtudes y carencias de las soluciones actuales", Universidad de Zaragosa, España, 2007.

[12] E.-P.L. San-Yih Hwang, Chien-Hsiang Lee, Cheng-Hung Chen, "Dynamic Web Service Selection for Reliable Web Service Composition", IEEE Services Computing, October 24, 2008.

[13] H. Jingjing, et al., "A Service Composition Model with Characteristic of Transaction Based on Finite State Machine", Hu School of Software, Beijing Institute of Technology, School of Mathematics, Capital Normal University, Beijing, Beijing Laboratory of Intelligent Information Technology, School of Computer Science, Beijing Institute of Technology, Beijing, China, IEEE International Conference on Computer and Electrical Engineering, 2008: p. 450-454.

[14] R. Ekwall, et al., "Towards Flexible Finite-State-Machine-Based Protocol Composition", Ecole Polytechnique Federale de Lausanne (EPFL), 2004.

[15] J.C. Corrales, "Composición Semántica De Servicios Web", Universidad del Cauca, Colombia, 2006.

[16] N. Guermouche and C. Godart, "Toward Data Flow Oriented Services Composition", Enterprise Distributed Object Computing Conference, 2008. EDOC '08. 12th International IEEE September 26, 2008: p. 379-385.

[17] D.C. Daniela Berardi, Giuseppe De Giacomo, Maurizio Lenzerini, Massimo Mecella, "Automatic Service Composition Based on Behavioral Descriptions", Int. J. of Cooperative Information Systems, 2005.

[18] Y. Bo, et al., "The Design of an Orchestrated Execution Environment Based on JBI", International Forum on Computer Science-Technology and Applications, 2009.

[19] D.M. Vladimir Tosic, Bernard Pagurek, "Dynamic Service Composition and Its Applicability to E-Business Software Systems – The ICARIS Experience", Department of Systems and Computer Engineering, Carleton University, 2000.

[20] V.A.H. Quintero, *Arquitectura de Referencia para la Composición de Servicios en Ambientes de Computación Ubicua*, in *Departamento de Telemática*. 2010, Universidad del Cauca: Popayán.

[21] M. Malek and N. Milanovic, "Current Solutions for Web Service Composition", Humboldt University, IEEE Internet Computing, vol. 8, no. 6, pp. 51-59, Nov./Dec, 2004.

[22] J.A. Cubillos, "Composición Semántica de Servicios Web", Grupo de Ingeniería Telemática, Universidad de Cauca Colombia, 2004.
[23] S. Bessler, et al., "An Orchestrated Execution Environment for Hybrid Services", Kommunikation in Verteilten Systemen Bern, Schweiz, 2007.
[24] J.G. Jorge Giraldo, Ovalle, Demetrio A, "Integración de Procesos de Negocio Basados en Servicios Web: Coreografía y Satisfacción de Restricciones", Revista de Ingenierías Universidad de Medellín, 2008.
[25] G. Jie, et al., "A Choreography Approach for Value-Added Services Creation", Wireless Communications, Networking and Mobile Computing, 2009. WiCom '09. 5th International Conference, 2009.
[26] A. Lehmann, et al., "TeamCom: A Service Creation Platform for Next Generation Networks", Fourth International Conference on Internet and Web Applications and Services, 2009.
[27] W3C, "Web Services Choreography Description Language Version 1.0", 2004, [Online]. Available:http://www.w3.org/TR/2004/WD-ws-cdl-10-20041217/. [Accessed March 16, 2011].
[28] W3C, "Web Service Choreography Interface (WSCI) 1.0", August 8, 2002, [Online]. Available:http://www.w3.org/TR/wsci/. [Accessed March 16, 2011].
[29] R. S, *Modelización*. Universidad Alianza. 1995, Madrid.
[30] W.A.U. Carlos Andrés Lopez, "Diseño de aplicación para manipulación de grafos con algoritmos heurísticos de análisis", Universidad Tecnológica de Pereira., December 15, 2008.
[31] J. Xiao and R. Boutaba, "QoS-Aware Service Composition and Adaptation in Autonomic Communication", Selected Areas in Communications, IEEE Journal on, 2005.
[32] D.A.K. G. Kapitsaki, I.E. Foukarakis, G. N. Prezerakos, D.I. Kaklamani, I.S. Venieris, "Service Composition State of the art and future challenges", Mobile and Wireless Communications Summit, 2007.
[33] D. Ferry, "JAIN SLEE (JSLEE) 1.1 Specification, Final Release", Sun Microsystems, Inc, Open Cloud, 2008.
[34] O.C.L. University of OTAGO, "About JAIN SLEE", JAINSLEE.ORG, July, 2008, [Online]. Available:http://www.jainslee.org/slee/slee.html. [Accessed April 20, 2011].
[35] OpenCloud, "FSM TOOL 0.8.5 DEVELOPER'S GUIDE", Copyright © 2009 OpenCloud Limited. All rights reserved., 2009.
[36] V.H.R. Villegas, "Aplicaciones de telefonía sobre JAIN SLEE", Departamento de Ingeniería de Sistemas y Automática Área de Ingeniería Telemática, Escuela Técnica Superior de Ingenieros, España.
[37] J.-L.D. Simon Znaty, Roland Geldwerth, "SIP : Session Initiation Protocol", EFORT, 2005.
[38] O.C. Limited, "Rhyno Application Server", 2009, [Online]. Available:http://opencloud.com/products/rhino-application-server/real-time-application-server/. [Accessed March 3, 2010].
[39] A.B. Alexandre Mendonça, Bartosz Baranowski, Douglas Silas, Eduardo Martins, Ivelin Ivanov, and Jared Morgan, "Mobicents JAIN SLEE User Guide", Mobicents, 2009, [Online]. Available:http://www.mobicents.org/. [Accessed March 7, 2010].
[40] I. Ivanov, "Mobicents: JSLEE for the People, by the People", Java.net, March 14, 2006, [Online]. Available:http://today.java.net/article/2006/03/07/mobicents-jslee-people-people. [Accessed April 20, 2011].
[41] M.d.P.J. Chimachaná and J.O.P. Peña, "Mecanismo para la integración de la tecnología JAIN SLEE en un entorno IMS para la creación y prestación de servicios de valor agregado", Departamento de Telemática, Universidad del Cauca, Popayán, 2010.
[42] J.A.R. Ramirez and J.D.R. Medina, "Lineamientos para composición de servicios de telecomunicaciones en un entorno JAIN SLEE basado en software de libre distribución.", Departamento de Telemática, Universidad del Cauca, Popayán, September, 2010.
[43] J. Niemöller, et al., "Multi-Technology Service Composition for the Telecommunication Domain - Concepts and Experiences", Next Generation Mobile Applications, Services and Technologies (NGMAST), 2010 Fourth International Conference on 2010: p. 34-41.
[44] E.C.a.K.H. Purdy, "An Application Router for SIP Servlet Application Composition", IEEE International Conference on Communications, 2008.
[45] E.m. code, "Welcome to Echarts", December 11, 2009, [Online]. Available:www.Echarts.org. [Accessed April 20, 2011].

[46] E.C.a.K.H. Purdy, "Application Composition in the SIP Servlet Environment", IEEE International Conference on Communications, 2007.
[47] Y.C. Mihir Kulkarni, "SIP Servlet Specification, version 1.1", Oracle Corporation, 2008.
[48] L. Lange, A. Blotny, and T. Magedanz, "An agile service platform for telecommunication environments", New Technologies, Mobility and Security (NTMS), 4th IFIP International Conference on 2011.
[49] M. Femminella, et al., "Scalability and Performance Evaluation of a JAIN SLEE-Based Platform for VoIP Services", 2009.
[50] A.G. Neiat, et al., "An Agent-based Semantic Web Service Discovery Framework", International Conference on Computer Modeling and Simulation, 2009.
[51] M. Sellami, S. Tata, and B. Defude, "Service Discovery in Ubiquitous Environments: Approaches and Requirements for Context-Awareness", Fourth International Conference on Internet and Web Applications and Services, ICIW, Venice, Italy, 2009.
[52] N.I.a.F.L. Moüel, "A survey on service composition middleware in pervasive environments", CoRR, vol. abs/0909.2183, 2009.
[53] S.R.P.a.A. Fox, "Sword: A developer toolkit for web service composition," in Proceedings of the 11th International WWW Conference (WWW2002), Honolulu, HI, USA, 2002.
[54] U.K. N. Lin, and E. Sirin, "Web service composition with user preferences", Proceedings of the 5th European semantic web conference on The semantic web: research and applications, ser. ESWC'08. Berlin, Heidelberg: Springer-Verlag, 2008.
[55] B.P. R. Masuoka, and Y. Labrou, "Task computing - the semantic web meets pervasive computing", In Proceedings of 2nd International Semantic Web Conference (ISWC2003), Sanibel Island, 2003.
[56] S.H.a.F. Mavaddat, "A graph-based approach to web services composition", Applications and the Internet, 2005. Proceedings. The 2005 Symposium, jan.-4 feb, 2005.
[57] A. Mendonça, et al., "Mobicents JAIN SLEE User Guide", 2011.
[58] ITU-T, "SERIES G: TRANSMISSION SYSTEMS AND MEDIA, DIGITAL SYSTEMS AND NETWORKS", ITU-T Recommendation G.1010, 2001.

Anexo A

Instalación de herramientas

A continuación se muestra la instalación de los módulos necesarios para llevar a cabo la composición dinámica de servicios SIP para un entorno ejecución de lógica de servicio, basada en FSM, en el sistema operativo Windows XP service pack 3.

Las herramientas instaladas son las que se muestran a continuación:

- Mobicents JAIN SLEE 2.3.0.FINAL
- Eclipse Helios v3.6.1
- Plug-in EclipSLEE v1.2.6

A.1 Instalación de Mobicents JAIN SLEE

Mobicents JAIN SLEE es la primera y única plataforma de código abierto certificada por JAIN SLEE 1.1, provee alta escalabilidad, un servidor de aplicaciones orientado a eventos con un modelo de componentes robusto y un ambiente de ejecución tolerante a fallos. Esta herramienta se puede descargar en la siguiente dirección:

http://sourceforge.net/projects/mobicents/files/, fecha de consulta 1-feb-2011.

Antes de comenzar con la instalación, se deben tener en cuenta los siguientes requisitos:

- Suficiente espacio en disco duro, se necesitan al menos 300MB una vez descomprimido mobicents.
- Es recomendable tener al menos 2GB o 4 GB de memoria RAM para sistemas operativos de 32 o 64 bit.
- Se requiere un JDK versión 6.

En el caso de este proyecto se cuenta con un sistema operativo Windows XP service pack 3, instalado sobre una máquina virtual utilizando la Herramienta VirtualBox cuyos recursos asignados fueron: Disco duro 80GB (33GB de espacio libre en la unidad *C:*\), memoria RAM de 1.08GB DDR3 y JDK 6u23.

Para mayor información se puede descargar la guía de usuario [ref Mobicents_SLEE_Container_User_Guide] en la siguiente dirección:

http://www.mobicents.org/slee/docs.html, fecha de consulta 2-feb-2011.

Para la instalación de esta plataforma se debe descomprimir el archivo .zip en la ubicación deseada, en este caso se descomprimió en *C:**Mobicents* utilizando la herramienta WinRAR, y posteriormente se debe crear la variable de entorno *JBOSS_HOME* apuntando al directorio que contiene todos los archivos de la plataforma mobicents, este directorio contiene al subdirectorio *bin.* El directorio que contiene el subdirectorio *bin* en este caso este caso es *jboss-5.1.0.GA* por lo tanto se creó la variable de entorno apuntando a la dirección *C:**Mobicents**jboss-5.1.0.GA* como se muestra en las figuras 22 y 23.

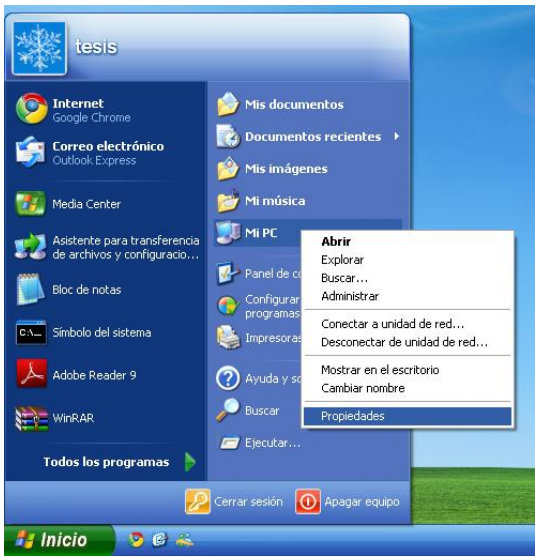

Figura 22. Abrir herramienta para crear variables de entorno

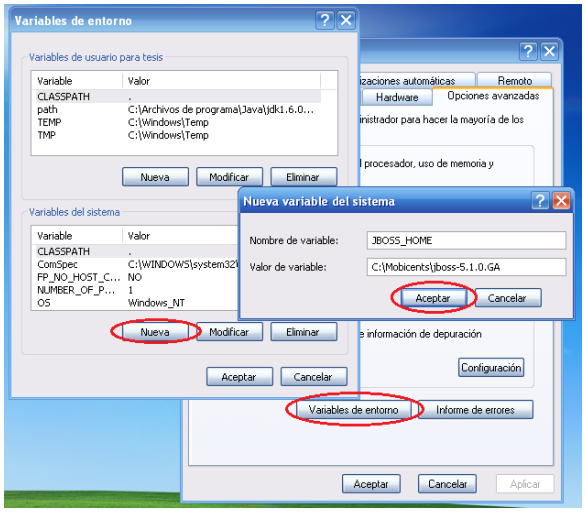

Figura 23. Creación de la variable de entorno

Una vez instalado, se puede iniciar el servidor desde la consola de Windows ubicándose en el subdirectorio *bin*, en este caso la ubicación es *C:\Mobicents\jboss-5.1.0.GA\bin*, una vez ahí se ejecuta el comando *run.bat*. Para detener el servidor, se abre una nueva consola de Windows, una vez ubicado en el subdirectorio *bin* se ejecuta el comando *shutdown.bat – s*, si el servidor fue detenido correctamente se deberán ver las líneas remarcadas en rojo de la figura 23.

Se recomienda agregar la variable de entorno JAVA_HOME indicando la ubicación del directorio donde se encuentra el JDK que se está utilizando, en este caso, *C:\Archivos de programa\Java\jdk1.6.0_23*, para que no haya problemas con el comando shutdown.bat – s

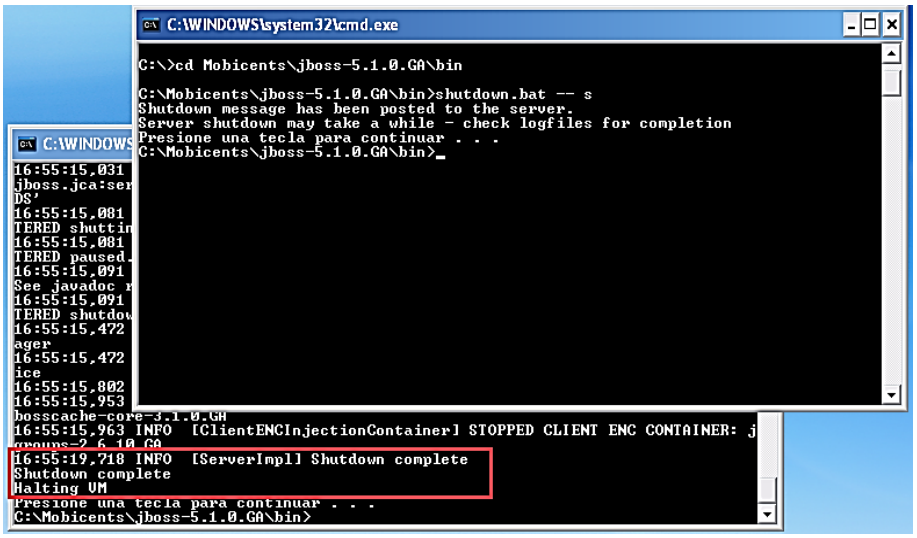

Figura 24. Servidor detenido correctamente

A.2 Instalación de Eclipse Helios v3.6.1

Eclipse es el entorno de desarrollo que permite la creación de servicios JAIN SLEE, una vez instalado el Plug-in necesario. Para usar esta herramienta, se debe descomprimir en alguna ubicación deseada el archivo .zip, que se puede descargar del siguiente enlace:

http://www.eclipse.org/downloads/, fecha de consulta 1-feb-2011.

Una vez descomprimido se ejecuta el archivo *eclipse.exe* que se encuentra dentro del directorio *eclipse*, y se elige una ubicación para el directorio donde se guardarán los proyectos que se van a realizar (*workspace*).

A.3 Instalación del Plug-in EclipSLEE v1.2.6

El Plug-in de JAIN SLEE para eclipse, EclipSLEE, está diseñado para la creación de los siguientes componentes [ref http://docs.jboss.org/mobicents/slee/2.3.0.FINAL/tools/eclipslee/user-guide/en-US/html_single/#preinstall_requirements_and_prerequisites]

- Especificación de perfiles
- Eventos
- Bloques de construcción de servicio (SBB)
- Archivos descriptores XML
- Unidades desplegables

Para la instalación de este Plug-in se pueden seguir dos procedimientos como se muestra en [ref http://docs.jboss.org/mobicents/slee/2.3.0.FINAL/tools/eclipslee/user-guide/en-US/html_single/#preinstall_requirements_and_prerequisites], en este caso se optó por el procedimiento manual, descrito a continuación:

- Descargar el archivo JAR de EclipSLEE de la siguiente dirección: http://mobicents.googlecode.com/svn/downloads/sip-servlets-eclipse-update-site/plugins/, fecha de consulta 4-feb-2011.
- Copiar el archivo JAR al subdirectorio *plugins* de la carpeta donde fue descomprimida la herramienta Eclipse.
- Reiniciar Eclipse.

Anexo B
Instalación del RA SIP 2.3.0

Se describe a continuación el proceso de instalación del adaptador de recursos SIP de Mobicents JAIN SLEE 2.3.0:

- Se descarga el código fuente del RA SIP usando un cliente de Subversión, como se muestra en la figura 25. Para este caso se utilizó la versión TortoiseSVN 1.6.1.2 para Windows XP OS 32 bits, el cual se puede descargar del siguiente enlace:
  http://tortoisesvn.net/downloads.html consulta el 16 de febrero de 2011

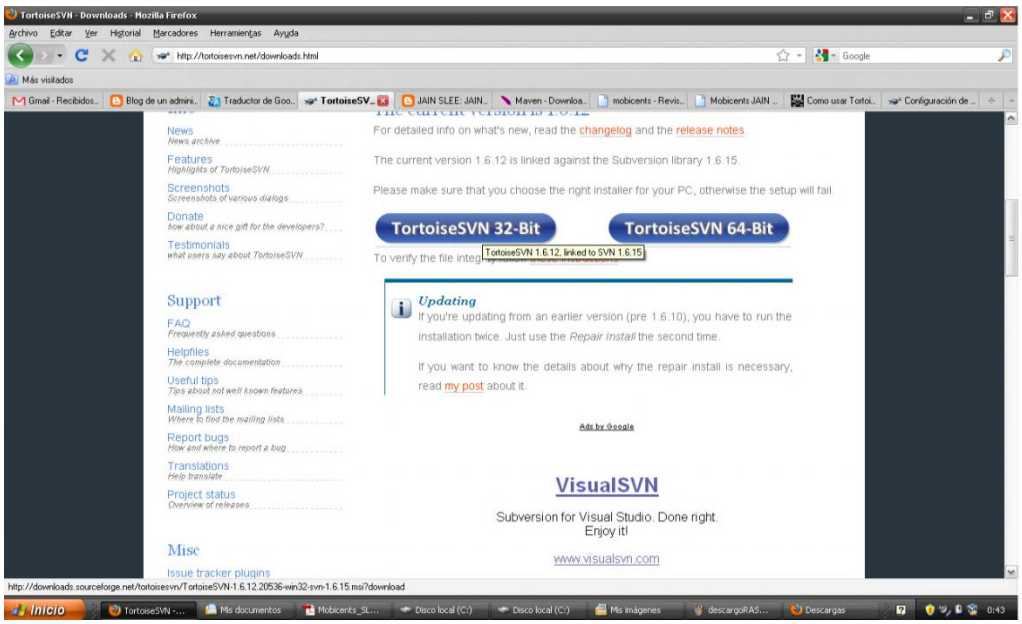

Figura 25. Descarga de TortoiseSVN

El cliente TortoiseSVN se instala normalmente y luego se procede a crear un repositorio SVN para descargar el código fuente del RA SIP de la siguiente manera:

1. Se crea una carpeta en la cual se aloja el código fuente del RA SIP y se realiza la descarga por medio de TortoiseSVN. Para eso, se creó la carpeta ubicada en *C:\RA_SIP,* luego se da clic derecho sobre la carpeta y se selecciona la opción *SVN obtener,* tal como se muestra en la figura 26.

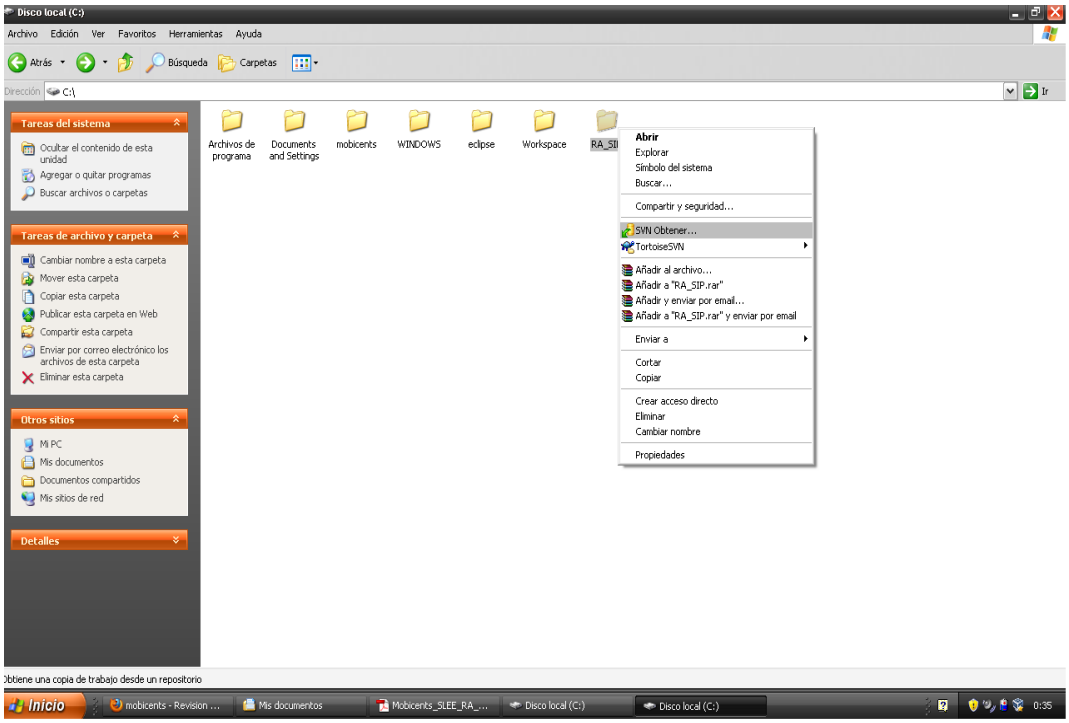

Figura 26. Repositorio local SVN

2. Luego se ingresa la URL del repositorio de donde se descarga el código fuente del RA SIP y se acepta la descarga, como se muestra en la figura 27. La URL de descarga por SVN es la siguiente:

   http://mobicents.googlecode.com/svn/tags/servers/jain-slee/2.x.y/resources/sip11/2.3.0.FINAL consulta el 16 de febrero de 2011.

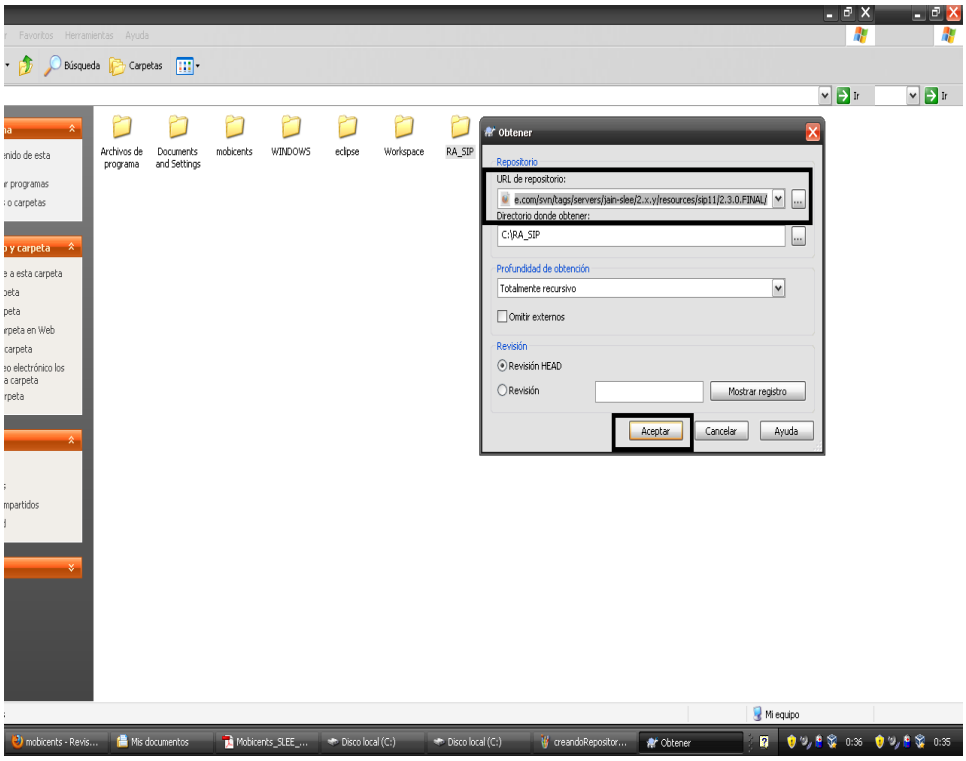

Figura 27. Descarga desde el servidor Subversion

- Luego de terminar la descarga en la carpeta RA_SIP, se procede a descargar e instalar el plugin de MAVEN en el equipo para generar la unidad de despliegue e instalarla en la plataforma de Mobicents. Maven es una herramienta de línea de comandos que hace parte del proyecto Apache, permite crear los directorios de proyectos JAVA, compilar y generar la unidad de despliegue e instalarla. Se utilizó la versión Maven 3.0.2, se descargó el paquete ZIP de la siguiente enlace:

  http://www.apache.org/dyn/closer.cgi/maven/binaries/apache-maven-3.0.2-bin.zip
  Consulta el 16 de febrero de 2011

- A continuación se procede a instalar el plugin de Maven sobre Windows XP OS 32 bits, para ejecutarlo en la consola de comandos. La instalación se realizó de la siguiente forma:

  1. Se descomprime el paquete ZIP en un directorio central, en este caso la ubicación es *C:\apache-maven-3.0.2.*

  2. Se adiciona la siguiente variable de entorno del sistema:

     ***Nombre de variable**: M2_HOME*
     ***Valor de variable:** C:\apache-maven-3.0.2 (esta es la ubicación donde se encuentra Maven).*

  3. *Luego se adiciona la siguiente variable de entorno del sistema:*

     ***Nombre de variable**: M2*
     ***Valor de variable:** `%M2_HOME%\bin`*

4. A continuación se modifica la variable de entorno del sistema: *Path, a la cual se le agrega en el valor de la variable*: %M2% (se antepone de primero a todos los valores actuales del *Path* para que sea reconocida como comando global).

5. *Luego se agrega a la misma variable de entorno del sistema: Path, en valor de la variable*: *%JAVA_HOME%\bin* (en caso que no lo tenga). Se guardan todos los cambios.

6. Luego se abre una consola de comandos de Windows, para comprobar que Maven quedo instalada correctamente se ejecuta el siguiente comando: *mvn –version* y se debe desplegar un mensaje como el que se aprecia en la figura 28.

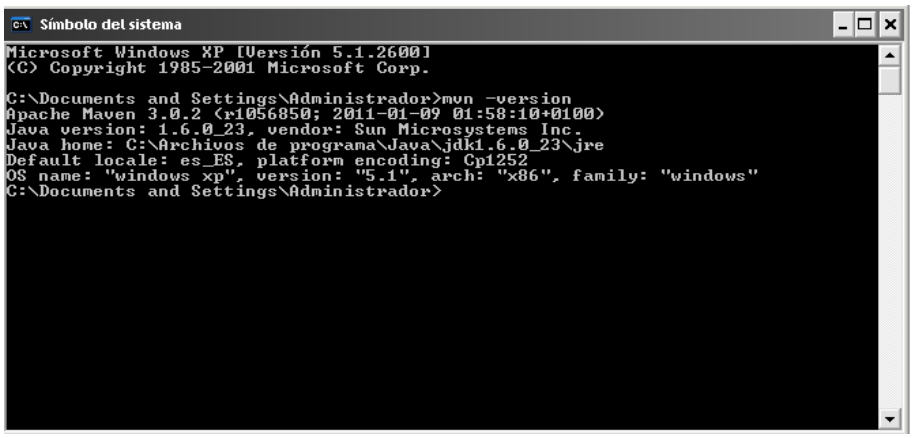

Figura 28. Instalación de Maven

- Para instalar el RA SIP se abre una consola de comandos de Windows y se ubica en el directorio donde se encuentra el código fuente del adaptador de recursos en este caso *C:\SIP_RA* y se procede a ejecutar el siguiente comand*o:* mvn install, luego de terminar el proceso de instalación se desplegar en pantalla un mensaje de proceso finalizado con éxito como se aprecia en la figura 29.

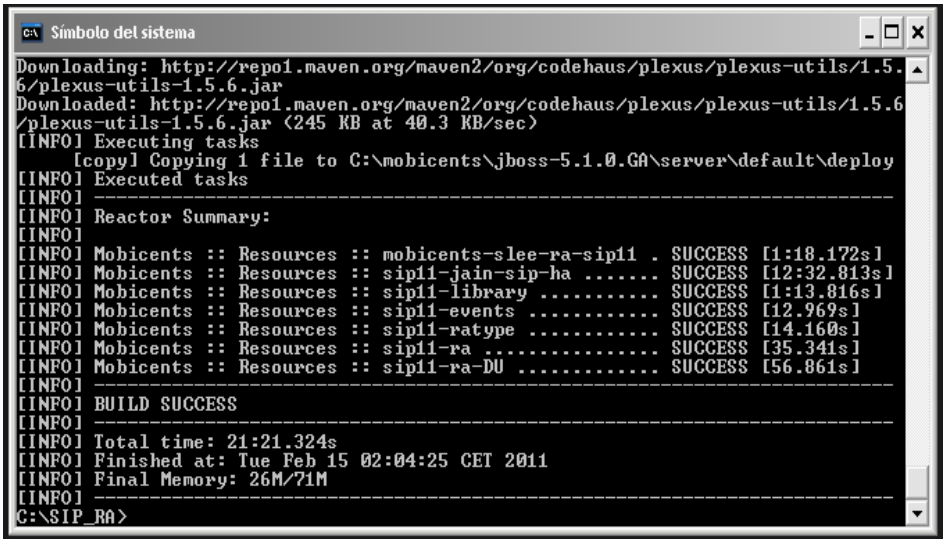

Figura 29. Instalación de RA SIP

Luego de terminar el proceso de instalación de RA SIP en Mobicents, para comprobar que ha sido instalada correctamente se verifica que en la carpeta *deploy,* donde se alojan las unidades de despliegue de componentes de esta plataforma (en este caso su ubicación corresponde

C:\mobicents\jboss-5.1.0.GA\server\default\deploy) que se encuentre el archivo JAR del RA SIP (su nombre corresponde: *sip11-ra-DU-2.3.0.FINAL*) u otra forma de comprobar es revisar desde la consola de administración JMX que se encuentre el adaptador de recursos SIP instalado (Para ingresar a la consola se debe haber arrancado el servidor y escribir en un browser el siguiente enlace: *http://localhost:8080/jmx-console*), tal como se muestra en la figura 30.

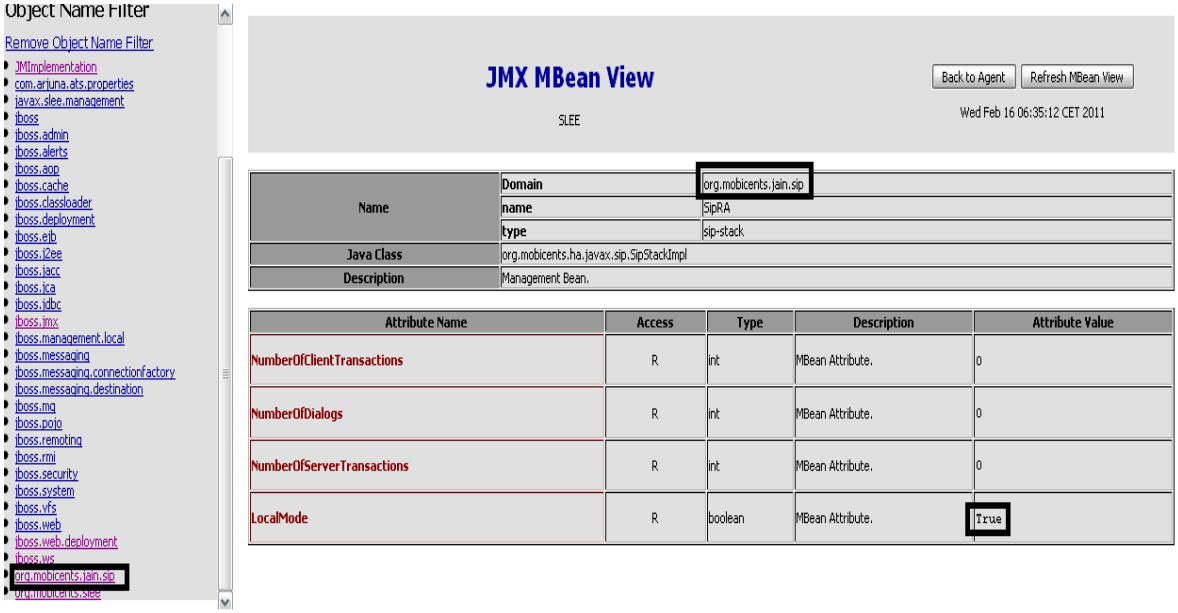

Figura 30. Despliegue del RA SIP en Mobicents JAIN SLEE

Anexo C

Entorno experimental de pruebas

A continuación se describen las características técnicas del entorno experimental utilizado para realizar las pruebas de desempeño sobre el módulo de composición y componente mediador para evaluar el rendimiento del algoritmo. Además se anexaran en forma digital las versiones de las herramientas utilizadas.

E.1 **Descripción del Entorno experimental de pruebas**

Figura 31. Entorno experimental de pruebas

En la figura 31 se observa el entorno experimental de pruebas realizadas, que consiste en un equipo configurado como un servidor de aplicaciones JSLEE 1.1 y otro equipo configurado como cliente SIP para realizar las peticiones al servidor por medio de una red local inalámbrica (WLAN – conexión a internet - 4MB/s). Los equipos: servidor y cliente serán descritas sus características a continuación:

1. **PC Servidor:**

Para configurar como equipo servidor se utilizó un computador portátil HP DV4 1225dx con las siguientes características:

- Memoria RAM 3 GB DDR2
- Procesador AMD TURION X2 - velocidad: 2.1 Ghz.
- Tarjeta de red inalámbrica Fast-Ethernet Broadcom 802.11g.
- Sistema operativo: Linux Ubuntu 9.04 – plataforma de 32 bits.
- JDK 1.6

Sobre este equipo se instaló y configuró una plataforma Mobicents JAIN SLEE 1.1 versión 2.3.0 Final (anexado en forma digital) y un adaptador de recursos SIP11 versión 2.3.0 (anexado en forma digital) de Mobicents.

Para el desarrollo del servicio de prueba, modulo de composición, componente mediador, modulo de SBB candidatos se utilizó el IDE Eclipse Helios v3.6.1 a la que se le adicionó el Plug-in EclipSLEE v1.2.6 (Anexo A).

2. **PC Cliente:**

Para configurar como equipo cliente se utilizó un computador portátil Toshiba Satellite L515 con las siguientes características:

- Memoria RAM 3 GB DDR3.
- Procesador Intel Core I3 - velocidad: 2.1 Ghz.
- Tarjeta de red inalámbrica Fast-Ethernet Realtek RTLL8187SE 802.11b/g.
- Sistema operativo: Windows 7 Home Premium – plataforma de 64 bits.
- JDK 1.6

Sobre este equipo se instaló y configuro dos clientes SIP (softphone) X-Lite versión 4 para realizar la generación de peticiones SIP para realizar pruebas sobre funcionalidades de llamada de voz, mensajería instantánea y registro de usuarios y su respectivo tarificación de la llamada y conteo de mensajes.

Anexo D

Selección dinámica de Servicios Web para la composición de Servicios Web confiable

Anexo E

Reconfiguración de servicios en ambientes de servicios de telecomunicaciones basados en JSLEE

Anexo F

Algoritmo de reconfiguración dinámica de servicios de telecomunicaciones en JSLEE